Forcepoint Next Generation Firewall and Amazon GuardDuty

Integration Guide

Integration Guide

Ultan Casey
21 Aug 2020
Public

forcepoint.com

Table of Contents

Version	Date	Author	Notes
0.1	6 May 2020	Ultan Casey	First draft
0.2	8 May 2020	Mattia Maggioli	Review
0.3	13 May 2020	Neelima Rai	Added troubleshooting chapter
0.4	13 May 2020	Mattia Maggioli	Review
0.5	21 August 2020	Ultan Casey	Replaced AWS deployment method with CloudFormation
0.6	21 August 2020	Mattia Maggioli	Review

Summary

This guide provides step by step instructions to set up an integration between the Forcepoint Next Generation Firewall (NGFW) and Amazon GuardDuty.

This integration enables automated export in real-time of Amazon GuardDuty findings into Forcepoint NGFW so that IP addresses identified within Amazon GuardDuty findings can be used in security policies of Forcepoint NGFW.

The code and instructions provided enable system administrators to automatically:

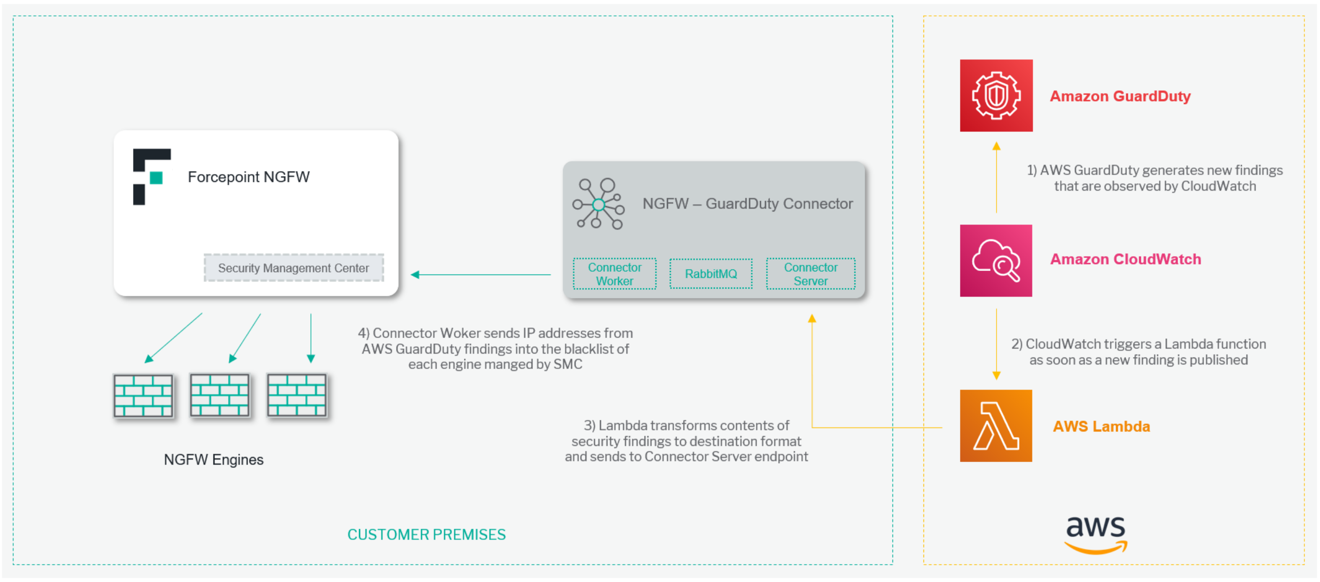
→ Track GuardDuty events using CloudWatch

→ Convert format and export security findings using a Lambda function

→ Receive the security findings with an on-premise connector

→ Add IP addresses into a user-defined list inside the Forcepoint NGFW Security Management Center (SMC)

A description of the workflow between the components involved in this POC is depicted in this diagram:


Forcepoint NGFW
Security Management Center
NGFW Engines
NGFW – GuardDuty Connector
Connector Worker
RabbitMQ
Connector Server
4) Connector Woker sends IP addresses from AWS GuardDuty findings into the blacklist of each engine manged by SMC
3) Lambda transforms contents of security findings to destination format and sends to Connector Server endpoint
CUSTOMER PREMISES
Amazon GuardDuty
1) AWS GuardDuty generates new findings that are observed by CloudWatch
Amazon CloudWatch
2) CloudWatch triggers a Lambda function as soon as a new finding is published
AWS Lambda
aws

Caveats
The integration described in this document was developed and tested with the following product versions:

→ Forcepoint NGFW 6.7.2

→ Forcepoint SMC 6.7.3

This interoperability uses:

→ **Amazon GuardDuty**: a threat detection service that continuously monitors for malicious activity and unauthorized behavior

→ **AWS CloudWatch**: a monitoring and observability service

→ **AWS Lambda:** a service which lets you run code without provisioning or managing servers

→ **AWS CloudFormation:** a service which provides one click deployment of entire stacks of AWS resources

→ **Connector Server**: a component hosted at customer premises exposing an endpoint used to receive security findings from AWS Lambda

→ **RabbitMQ**: a message broker used to handle queuing of ingestion tasks

→ **Connector Worker:** a task handler which controls the ingestion of IP addresses into the user-defined list using the SMC API

Implementation options

Two implementation options are provided in this document for the on-premise portion of the integration.

1. **Docker** – Uses a docker image where the integration component is already installed with all necessary dependencies: the user only has to edit one configuration file and run the container on an existing docker setup.
2. **Traditional** – requires manual deployment of the integration component inside a clean host machine.

The **docker** version of the connector server and worker requires the following files:

→ **fp-ngfw-aws-guardduty-docker-v1.zip** available at this link: https://frcpnt.com/fp-ngfw-aws-guardduty-docker-latest

and has been tested working with the following requirements:

→ Docker 19.03.6

→ Docker Compose 1.25.5

→ The host machine meets the minimum hardware requirements of 2GB of RAM and 20GB of storage

The **traditional** version of the connector server and worker requires the following files:

→ **fp-ngfw-aws-guardduty-v1.zip** available at this link: https://frcpnt.com/fp-ngfw-aws-guardduty-latest

and has been tested working with the following requirements:

→ CentOS 7.6 with at least 2 GB RAM and 20 GB disk

→ Python 3.6

→ The following Python modules:

- flask v1.1.1
- requests v2.23.0
- celery v4.4.2
- pyyaml v5.3.1
- fp-NGFW-SMC-python v0.7.0b20

Both implementations require

→ **fp-ngfw-aws-guardduty-lambda-v1.zip** available at this link: https://frcpnt.com/fp-ngfw-aws-guardduty-lambda-latest

→ **fp-ngfw-aws-guardduty-cloudformation-v1.zip** available at this link: https://frcpnt.com/fp-ngfw-aws-guardduty-cloudformation-latest

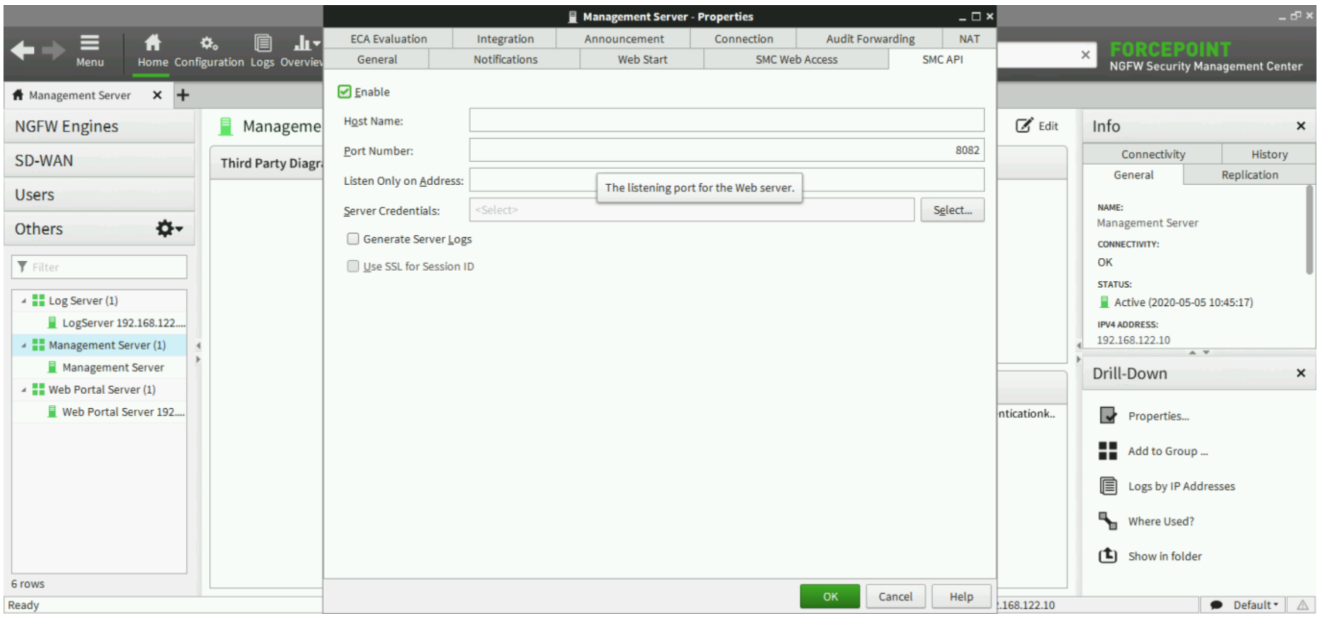
→ **Amazon GuardDuty** disabled during installation of the CloudFormation stack.

→ A public IP assigned to the **Connector Server** hosted on-premise either inside a docker container or a local machine.

Configure API Client in Forcepoint NGFW SMC

To enable the Connector to ingest new IP addresses into the blacklist of SMC an API client must first be created and API must be enabled.

1. Login to **Forcepoint SMC** and from the homepage navigate to **Configuration**.
2. From the sidebar open **Administration > Access Rights > API Clients.**
3. Select **New** to create a new API client.
4. Provide a name for your new Client. In this example we will use **Amazon GuardDuty**.
5. Under permissions provide the client with unrestricted permissions.
6. Save the **Authentication Key** in a secure location as it will be necessary in the next steps of this document, then click **Ok**.
7. Once again from the homepage of the SMC navigate to **Others.**
8. Right click **Management Server** and select **Properties.**
9. Open the **SMC API** tab and enable the checkbox there to enable the API.



☑ Enable
Host Name:
Port Number: 8082
Listen Only on Address:
The listening port for the Web server.
Server Credentials: <Select> Select...
☐ Generate Server Logs
☐ Use SSL for Session ID

NAME: Management Server
CONNECTIVITY: OK
STATUS: Active (2020-05-05 10:45:17)
IPV4 ADDRESS: 192.168.122.10

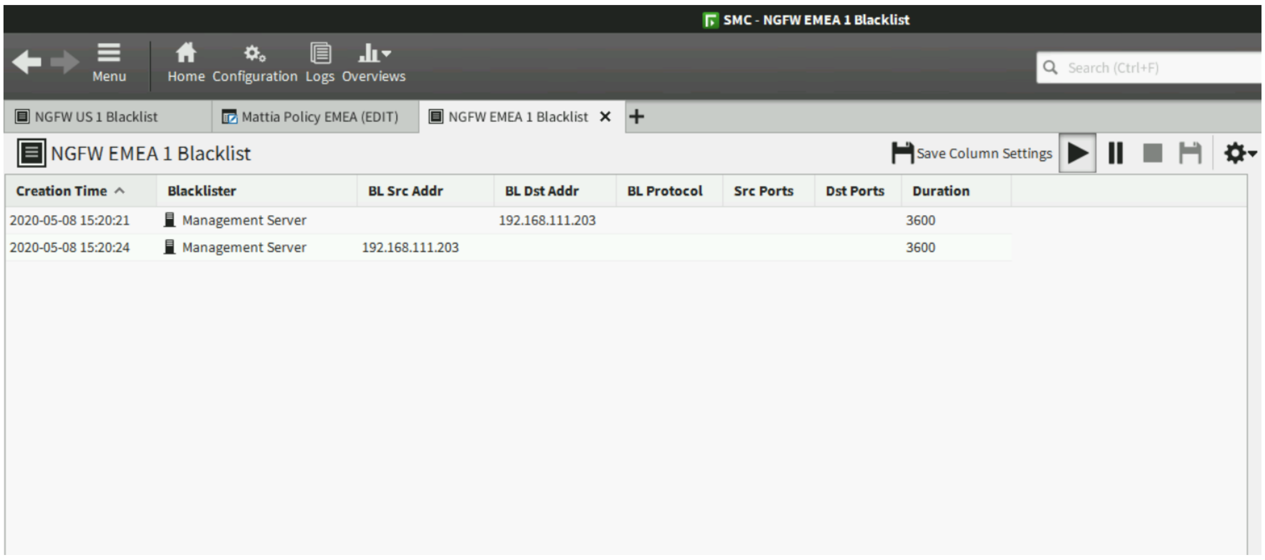
Lifecycle of IP addresses received from Amazon GuardDuty

Each NGFW engine managed by the SMC maintains its own separate blacklist of IP addresses. Blacklist can be configured into policies which are then applied to the engines.

When a relavent event occurs within **GuardDuty** which contains an IP address, this is sent using **AWS Lambda** to the **Connector Server** which places it in the queue to be added to the NGFW engines. The worker then pulls this from the queue and adds it to each engine. It is possible to exclude named engines whose blacklist will not receive the IP address: this is explained in the **Appendix** of this document.

The addresses which have been added to the blacklist remain within the blacklist for the duration defined in the configuration file. After this time is expired they are automatically removed.

Any new additions to the blacklists are immediately available to rules utilising the blacklists.



Creation Time	Blacklister	BL Src Addr	BL Dst Addr	BL Protocol	Src Ports	Dst Ports	Duration
2020-05-08 15:20:21	Management Server		192.168.111.203				3600
2020-05-08 15:20:24	Management Server	192.168.111.203					3600

Implementation - Docker

In order to set up the Connector do as follows:

1. Login to the docker registry with the following command

   docker login docker.frcpnt.com

   User: fp-integrations
   Pass: t1knmAkn19s

2. Download and extract the **fp-ngfw-aws-guardduty-docker-v1.zip** file available at this link: https://frcpnt.com/fp-ngfw-aws-guardduty-docker-latest to the downloads folder of the machine.

3. Create a configuration file named **config.yaml** with the following contents, editing the values as described in the Appendix of this document. A sample is also contained within the config folder included with the downloaded files.

```
# Connector config
host: 0.0.0.0
port: 5000
log_level: info

# SMC config
smc_endpoint: http://<SMC_ENDPOINT_IP_OR_DOMAIN>
smc_port: 8082
smc_api_key: <SMC_API_KEY>
smc_api_version: 6.7
blacklist_duration: 3600
exclude_engines:
  - NGFW 1
  - NGFW 2
```

Upload the file to a secure location where it can be accessed over http/https (e.g. a web server, an AWS S3 bucket, Azure Blob Storage): this will make sure the configuration is not lost in case the docker container is decommissioned.

4. Retrieve the URL to the file and store it in a safe location as it will be used in the next steps of this document
5. Open the **docker-compose.yml** file and replace both instances of **<CONFIG_FILE_URL>** with the URL of the config file just uploaded.
6. From the command line run the following command:

   ```
   docker-compose up -d
   ```

7. Configure the necessary network and security settings in your infrastructure to expose the **Connector** endpoint using a public IP, so that the endpoint can be reached by AWS Lambda.

Implementation - Traditional

1. Download and extract the **fp-ngfw-aws-guardduty-v1.zip** file available at this link: https://frcpnt.com/fp-ngfw-aws-guardduty-latest to the downloads folder.
2. Create a configuration file named **config.yaml** with the following contents, editing the values as described in the Appendix of this document. A sample is also contained within the config folder included with the downloaded files.

```
# Connector config
host: 0.0.0.0
port: 5000
log_level: info

# SMC config
smc_endpoint: http://<SMC_ENDPOINT_IP_OR_DOMAIN>
smc_port: 8082
smc_api_key: <SMC_API_KEY>
smc_api_version: 6.7
blacklist_duration: 3600
exclude_engines:
  - NGFW 1
  - NGFW 2
```

Upload the file to a secure location where it can be accessed over http/https (e.g. a web server, an AWS S3 bucket, Azure Blob Storage): this will make sure the configuration is not lost in case the services are moved elsewhere.

3. Retrieve the URL to the file and store it in a secure location as it will be used in the next steps of this document
4. From the command line navigate to the downloads folder and the extracted files.
5. Run the following command:

sudo chmod +x ./install.sh

6. Open both the **start_server.sh** file and **start_worker.sh** file with a text editor and replace <CONFIG_URL> with the URL of the config file you uploaded.
7. Next install everything executing this command:

   sudo ./install.sh

8. When prompted, type **y**.
9. Configure the necessary network and security settings in your infrastructure to expose the **Connector** endpoint using a public IP, so that the endpoint can be reached by AWS Lambda.

Configure Amazon resources using CloudFormation

To enable events to be sent from GuardDuty to the on-premise connector we must now configure the AWS components involved in this integration:

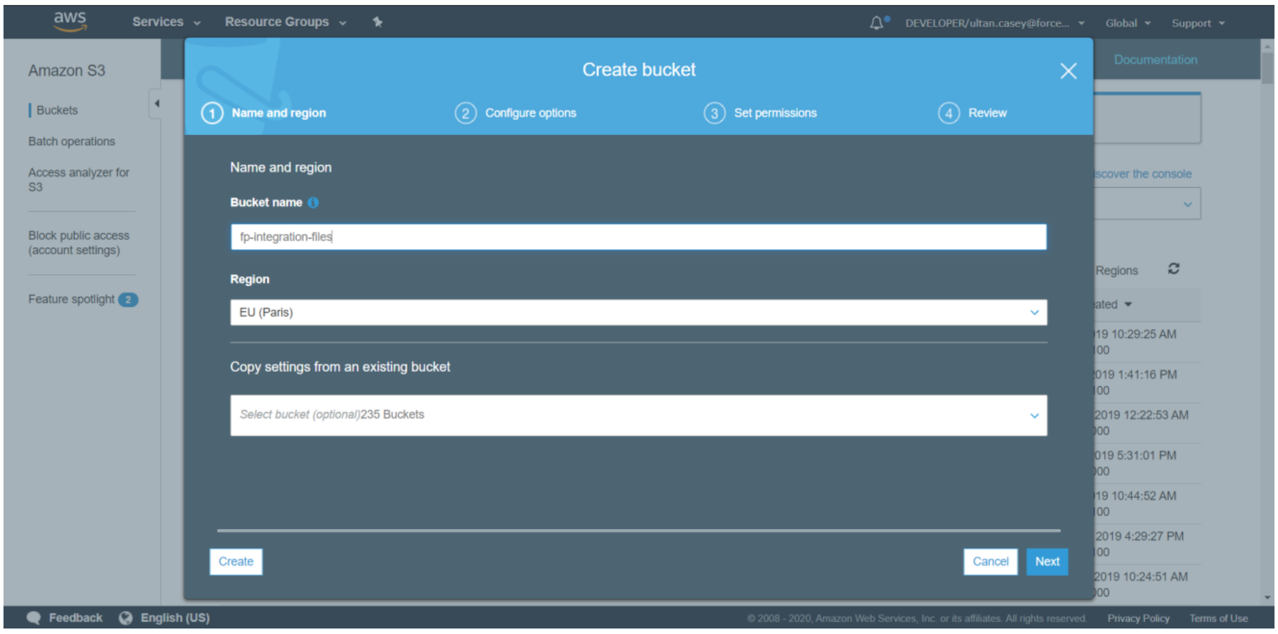
→ Amazon GuardDuty which detects events

→ a CloudWatch rule which triggers an AWS Lambda function upon the creation of a new finding in Amazon GuardDuty

→ an AWS Lambda function which pushes the finding to our integration component

This section requires **fp-ngfw-aws-guardduty-lambda-v1.zip** file available at this link: https://frcpnt.com/fp-ngfw-aws-guardduty-lambda-latest and **fp-ngfw-aws-guardduty-cloudformation-v1.zip** file available at this link: https://frcpnt.com/fp-ngfw-aws-guardduty-cloudformation-latest

1. Download both required zip files to the downloads folder.
2. Extract **fp-ngfw-aws-guardduty-cloudformation-v1.zip** which contains **fp-ngfw-aws-guardduty-cloudformation-v1**
3. Navigate to the AWS management console and from there open the **S3** control panel.
4. Click on **Create Bucket**. Set the name of the bucket to **fp-integrations-files** and leave the rest as default. Click **Create.**



Name and region

Bucket name: fp-integration-files

Region: EU (Paris)

Copy settings from an existing bucket

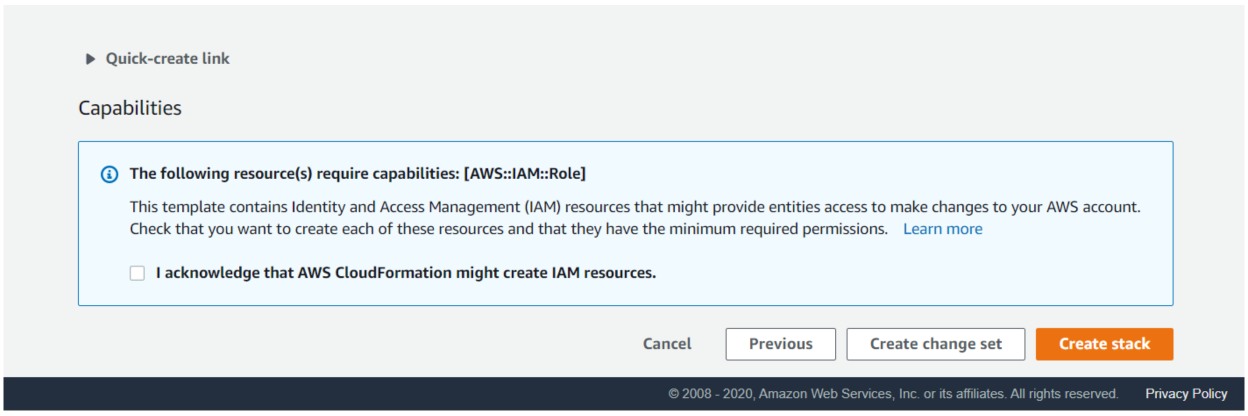
Select bucket (optional)235 Buckets

5. Navigate to the newly created bucket and click **Create Folder** and name it **functions**. Open this folder and select **Upload.** Drag the **fp-ngfw-aws-guardduty-lambda-v1.zip** file into the upload window and select **Upload.**
6. Return to the AWS Management Console and from there search for and open CloudFormation.
7. Click **Create Stack** and from the dropdown select **With new resources (standard)**.
8. Leave the **Prepare Template** field as default and under **Specify Template** select **Upload a Template File**. Upload the **fp-ngfw-aws-guardduty-cloudformation-v1** file extracted earlier. Click **Next.**
9. Set the stack name to **ForcepointNGFW-GuardDuty**. Under parameters update the SMC Endpoint to that of the connector. It should be in the following format

   http://<IP_OR_ADDRESS_OF_CONNECTOR>:5000/api/

   Leave all other parameters as default and click **Next**.

10. On the next page click **Next**.



▶ Quick-create link

Capabilities

The following resource(s) require capabilities: [AWS::IAM::Role]

This template contains Identity and Access Management (IAM) resources that might provide entities access to make changes to your AWS account. Check that you want to create each of these resources and that they have the minimum required permissions. Learn more

☐ **I acknowledge that AWS CloudFormation might create IAM resources.**

Cancel
Previous
Create change set
Create stack
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy

11. On the last page select '**I acknowledge that AWS CloudFormation might create IAM resources'** and click **Create Stack** to kick off the deployment of all the resources. This may take up to five minutes to deploy.

Once the deployment is completed, new findings generated by Amazon GuardDuty will be pushed to the integration component and subsequently into the SMC, which will then propagate the IP addresses into the blacklist of the NGFW engines.

Troubleshooting

Follow these steps to identify issues impacting the normal operation of the integration described in this document.

Docker Implementation

Validate the prerequisites

Make sure the prerequisites described in the **Summary** chapter are all satisfied:

→ Check the versions of Forcepoint NGFW and Forcepoint SMC in use are listed as compatible

Forcepoint NGFW 6.7.2

Forcepoint SMC 6.7.3

→ Docker images for this integration have been tested with

Docker 19.03.6

Docker-compose 1.25.5

→ The host machine should have at least 2 GB RAM and 20 GB disk and an existing docker engine and docker compose installed

→ Amazon GuardDuty disabled during installation of the CloudFormation stack.

→ Check the user can download the necessary files with the following commands:

wget --content-disposition https://frcpnt.com/fp-ngfw-aws-guardduty-docker-latest

wget --content-disposition https://frcpnt.com/fp-ngfw-aws-guardduty-lambda-latest

wget --content-disposition https://frcpnt.com/fp-ngfw-aws-guardduty-cloudwatch-latest

Check network connectivity

Make sure firewalls or other security appliances are not impacting the network connectivity necessary for the operation of all components involved into this integration:

→ Check the docker host machine has connectivity to SMC by executing the following command on docker host machine

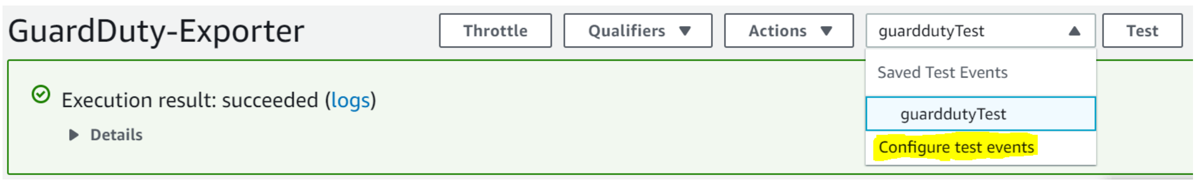
ping -c 2 SMC_HOST_IP_ADDRESS

replacing the SMC_HOST_IP_ADDRESS with your Forcepoint SMC host IP address. Once done check the result is similar to below:

```
PING SMC_HOST_IP_ADDRESS.url (10.10.120.12) 56(84) bytes of data.
64 bytes from 10.10.120.12 (10.10.120.12): icmp_seq=1 ttl=128 time=179 ms
64 bytes from 10.10.120.12 (10.10.120.12): icmp_seq=1 ttl=128 time=181 ms
```

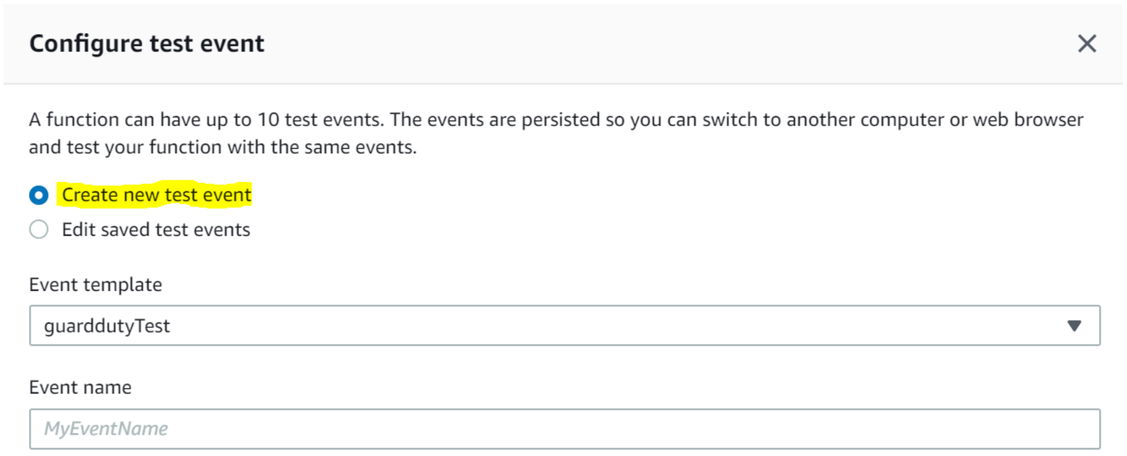
→ Check AWS Lambda can successfully post findings to the docker host machine by following the steps below:

1. Go to **AWS Lambda** service in the AWS Console.
2. Go to the **GuardDuty-Exporter** lambda function and click on **Configure test events**



Execution result: succeeded (logs)

3. Click on **Create new test event**



Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

[✓ Create new test event]
Edit saved test events

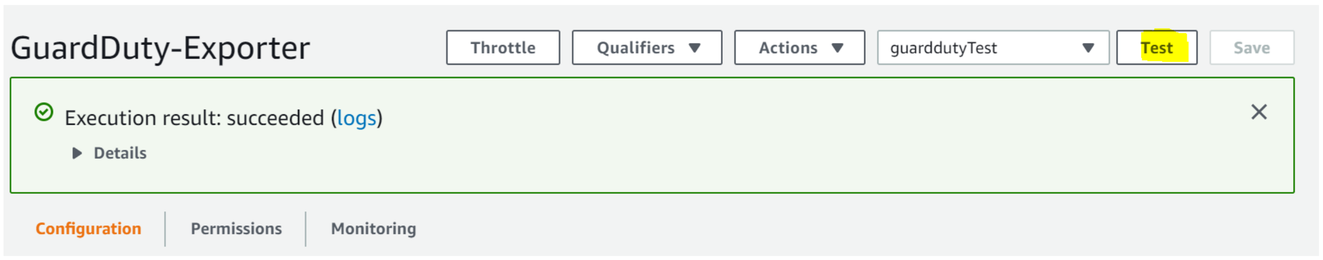
Event template: guarddutyTest

Event name: MyEventName

4. Enter the below contents into the empty json field and click **Create**

```
{
  "detail": {
    "type": "UnauthorizedAccess:IAMUser/MaliciousIPCaller.Custom",
    "service": {
      "action": {
        "awsApiCallAction": {
          "remoteIpDetails": {
            "ipAddressV4": "198.51.100.0"
          }
        }
      }
    }
  }
}
```

5. Click on **Test** button and verify you get **Execution result: succeeded**



Execution result: succeeded (logs)

Check dependencies are installed

Make sure the software dependencies needed by the components involved into this integration are installed:

→ Check all dependencies are installed: execute the following command on docker host machine to check docker-compose is installed:

docker-compose --version

→ Check the output presents a version of 1.25.4 or higher (example below):

```
docker-compose version 1.25.5, build 8d51620a
```

→ Check the host machine has docker installed: Execute the following command on the host machine:

docker --version

→ Check the first few lines of the output are similar to below:

Docker version 19.03.8, build afacb8b

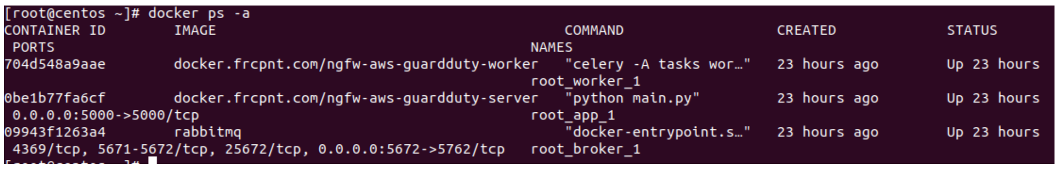
Check all components are configured and running properly

Make sure the products and services involved into this integration are configured as expected and they are running:

→ Check the docker containers are running with the below command:

docker ps -a

Verify the output is similar to below:



```
[root@centos ~]# docker ps -a
CONTAINER ID        IMAGE                                      COMMAND                  CREATED             STATUS
 PORTS                                                NAMES
704d548a9aae        docker.frcpnt.com/ngfw-aws-guardduty-worker   "celery -A tasks wor…"   23 hours ago        Up 23 hours
                                                      root_worker_1
0be1b77fa6cf        docker.frcpnt.com/ngfw-aws-guardduty-server   "python main.py"         23 hours ago        Up 23 hours
 0.0.0.0:5000->5000/tcp                               root_app_1
09943f1263a4        rabbitmq                                      "docker-entrypoint.s…"   23 hours ago        Up 23 hours
 4369/tcp, 5671-5672/tcp, 25672/tcp, 0.0.0.0:5672->5762/tcp   root_broker_1
```

→ Check the Connector Server is reachable and running by using a browser to open the webpage at the address:

http://<IP_OR_ADDRESS_OF_CONNECTOR>:5000/api/

Replace the part in red with the public IP address or FQDN resolving to the public IP assigned to the docker host. Check the page at the above URL reads:

Running...

→ Verify the below services are enabled and running on AWS:

- GuardDuty

  From the AWS Management Console navigate to AWS GuardDuty. If GuardDuty has been enabled, the findings field should be visible. If the deployment failed, the option to enable GuardDuty will be present.

- CloudWatch

To verify that the CloudWatch rule has been deployed, navigate to AWS CloudWatch and from there select Rules in the sidebar. If the deployment was successful a rule should be present with a name similar to: ForcepointNGFW-GuardDuty-GuardDutyWatcher-<AWS generated ID>

- Lambda

  If the Lambda function successfully deployed, you should be able to verify it by navigating to AWS Lambda and in the list of functions there should be one with a name similar to: ForcepointNGFW-GuardDuty-GuardDutyExporter- <AWS generated ID>

Traditional Implementation

Validate the prerequisites

Make sure the prerequisites described in the **Summary** chapter are all satisfied:

- → Check the versions of Forcepoint NGFW and Forcepoint SMC in use are listed as compatible

  Forcepoint NGFW 6.7.2

  Forcepoint SMC 6.7.3

- → Verify the integration is operating on a CentOS 7.6 machine with at least 2 GB RAM and 20 GB disk
- → User needs sudo permissions on the host machine
- → Amazon GuardDuty disabled during installation of the CloudFormation stack.
- → Check the user can download the necessary files with the following commands:

  wget --content-disposition https://frcpnt.com/fp-ngfw-aws-guardduty-latest

  wget --content-disposition https://frcpnt.com/fp-ngfw-aws-guardduty-lambda-latest

  wget --content-disposition https://frcpnt.com/fp-ngfw-aws-guardduty-cloudwatch-latest

Check network connectivity

Make sure firewalls or other security appliances are not impacting the network connectivity necessary for the operation of all components involved in this integration:

- → Check the host machine has connectivity to SMC: execute the following command on the host machine:

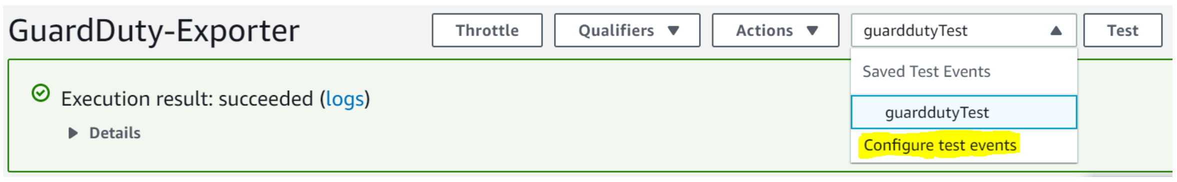
  ping -c 2 SMC_HOST_IP_ADDRESS

  Replacing the SMC_HOST_IP_ADDRESS with your Forcepoint SMC host IP address. Once done check the result is similar to below:

```
PING SMC_HOST_IP_ADDRESS.url (10.10.120.12) 56(84) bytes of data.
64 bytes from 10.10.120.12 (10.10.120.12): icmp_seq=1 ttl=128 time=179 ms
64 bytes from 10.10.120.12 (10.10.120.12): icmp_seq=1 ttl=128 time=181 ms
```

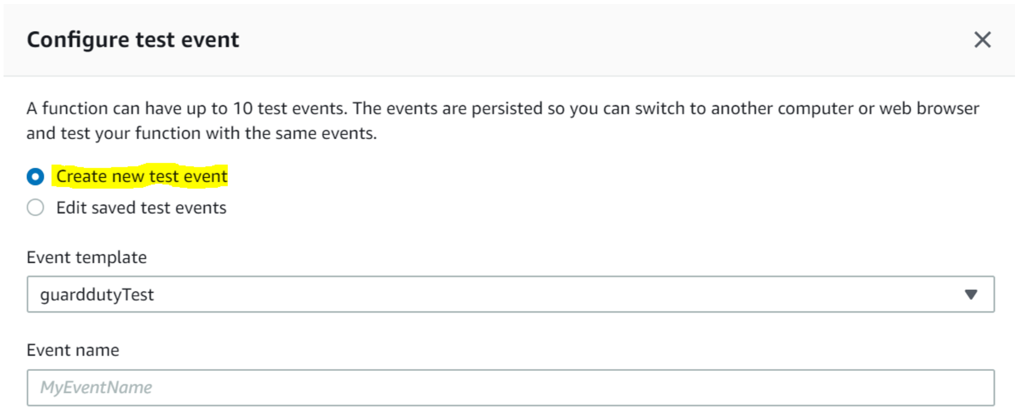
- → Check AWS lambda can successfully post findings to the connector host machine by following the steps below:

  1. Go to **AWS Lambda** service in the AWS Console
  2. Go to the **GuardDuty-Exporter** lambda function and click on **Configure test events**



Execution result: succeeded (logs)

  3. Click **Create new test event**



Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

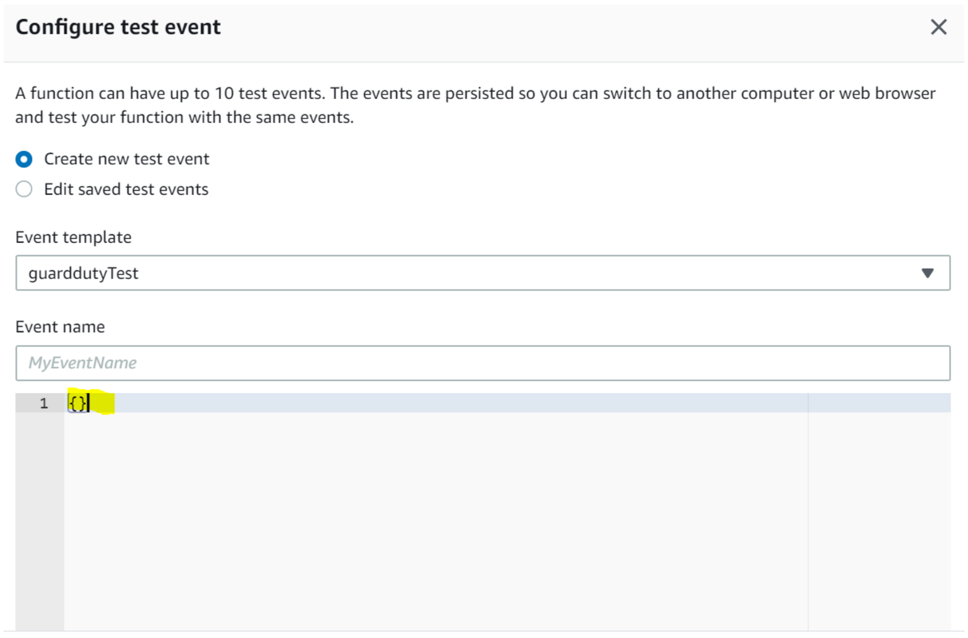
[✓ Create new test event]
Edit saved test events

Event template
guarddutyTest

Event name
MyEventName

4. Enter the below contents into the empty json field and click **Create**

```
{
  "detail": {
    "type": "UnauthorizedAccess:IAMUser/MaliciousIPCaller.Custom",
    "service": {
      "action": {
        "awsApiCallAction": {
          "remoteIpDetails": {
            "ipAddressV4": "198.51.100.0"
          }
        }
      }
    }
  }
}
```



Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

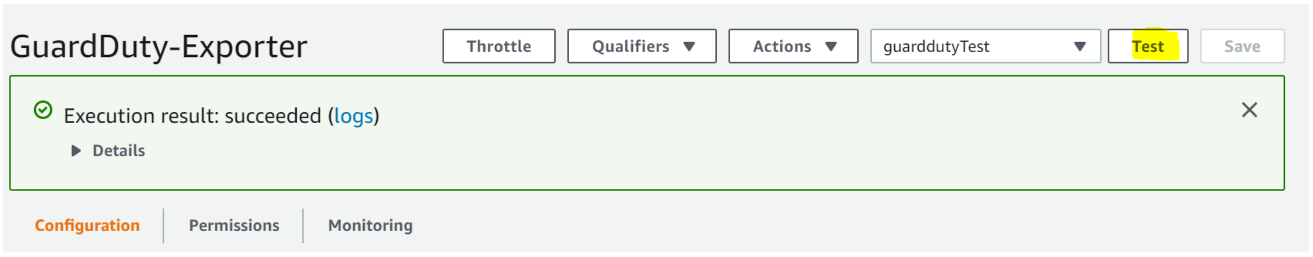
[✓ Create new test event]
Edit saved test events

Event template
guarddutyTest

Event name
MyEventName

```
{}
```

5. Click **Test** button and verify you get Execution result: succeeded



Execution result: succeeded (logs)

Check dependencies are installed

Make sure the software dependencies needed by the components involved into this integration are installed:

→ Check python3.6 is installed: Execute following command on host machine:

```
python3 --version
```

Check the output is similar to below:

```
Python 3.6.8
```

→ Check the necessary packages are installed: Execute the following command on the host machine:

```
pip3 list 2> /dev/null | grep -e requests -e Flask -e PyYaml -e celery -e fp-NGFW-SMC-python
```

Check the output is similar to below:

```
celery 4.4.2
Flask                1.1.1
fp-NGFW-SMC-python     0.7.0b20
requests              2.23.0
requests-unixsocket    0.2.0
```

Check all components are configured and running properly

Make sure the products and services involved into this integration are configured as expected and they are running:

→ Check the necessary services are running with the following commands:

```
systemctl status ngfw-aws-guardduty-worker.service

systemctl status ngfw-aws-guardduty-server.service
```

Check the output for both shows active status for both the services

→ Check the domain service is running at the below webpage:

http://<IP_OR_ADDRESS_OF_CONNECTOR>:5000/api/

Replace the part in red with the public IP address or FQDN resolving to the public IP assigned to the docker host. Check the page at the above URL reads:

```
Running...
```

→ Verify the below services are enabled and running on AWS:

- GuardDuty

From the AWS Management Console navigate to AWS GuardDuty. If GuardDuty has been enabled, the findings field should be visible. If the deployment failed, the option to enable GuardDuty will be present.

- CloudWatch

To verify that the CloudWatch rule has been deployed, navigate to AWS CloudWatch and from there select Rules in the sidebar. If the deployment was successful a rule should be present with a name similar to: ForcepointNGFW-GuardDuty-GuardDutyWatcher-<AWS generated ID>

- Lambda

If the Lambda function successfully deployed, you should be able to verify it by navigating to AWS Lambda and in the list of functions there should be one with a name similar to: ForcepointNGFW-GuardDuty-GuardDutyExporter- <AWS generated ID>

Appendix – Configuration File

This table provides a description for the values required in the configuration file utilized by the **Connector**

Field	Example	Notes	Requires to be changed
host	0.0.0.0	The IP address of the interface the Connector service is binded to on the hosting machine or the docker container. Default is 0.0.0.0 which will listen to any interface.	No
port	5000	The port on which the connector will listen for requests. Default is 5000.	No
log_level	info	Log level determines the level of logs to be created.	No
smc_endpoint	https://192.168.1.222	The FQDN or IP address of the SMC endpoint. Used for accessing the SMC API.	Yes
smc_port	8082	The port on which the SMC API is accessible. Default is 8082. This can be retrieved when enabling the SMC API.	Yes
smc_api_key	abcdefgh1234567	API key necessary to utilise the SMC API.	Yes
smc_api_version	6.7	Version of the SMC API to be used. Default is 6.7.	No
blacklist_duration	3600	Duration in seconds of how long IP addresses ingested from GuardDuty will stay in the user-defined list in the SMC.	No
exclude_engines	- NGFW NAME - NGFW 2	Engines whose blacklist will **not** receive the IP addresses exported from GuardDuty. Each engine is denoted by a dash followed by the name of the engine as visible in the list of engines in the SMC.	Yes

forcepoint.com/contact

About Forcepoint

Forcepoint is the global human-centric cybersecurity company transforming the digital enterprise by continuously adapting security response to the dynamic risk posed by individual users and machines. The Forcepoint human point system delivers risk-adaptive protection to continuously ensure trusted use of data and systems. Based in Austin, Texas, Forcepoint protects the human point for thousands of enterprise and government customers in more than 150 countries.

© 2020 Forcepoint. Forcepoint and the FORCEPOINT logo are trademarks of Forcepoint. All other trademarks used in this document are the property of their respective owners.