Forcepoint CASB and Azure Active Directory

Integration Guide

Integration Guide

Dlo Bagari
05 May 2020
Public

forcepoint.com

Table of Contents

Version	Date	Author	Notes
0.1	05 May 2020	Dlo Bagari	First draft
0.2	07 May 2020	Dlo Bagari	Updated
0.3	08 May 2020	Neelima Rai	Added troubleshooting chapter
0.4	08 May 2020	Mattia Maggioli	Review

Summary

This guide provides step by step instructions to set up an integration between **Forcepoint CASB** and **Azure Active Directory** (AD) to pass risk scores and to adjust authentication policies accordingly, while also providing a tool to export Azure AD events.

The code and instructions provided enable system administrators to automatically:

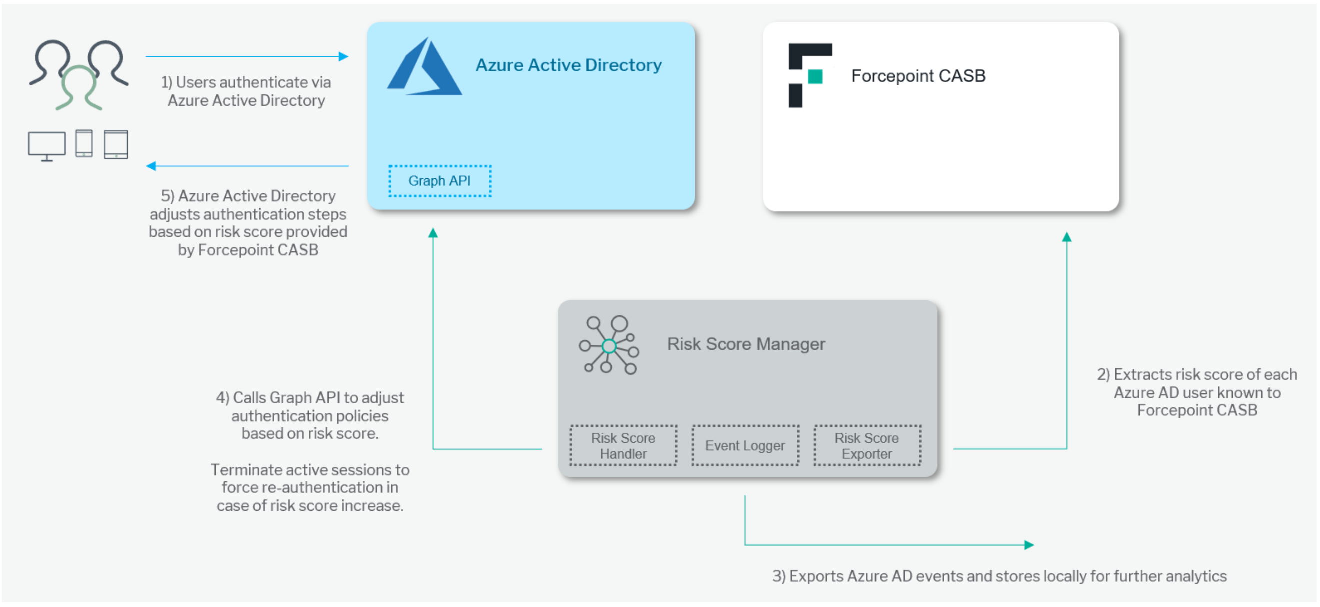
→ Provide the risk score calculated by Forcepoint CASB for each user managed by Azure AD.

→ Adjust authentication policies applied by Azure AD to users based on their risk score.

→ Terminate active sessions to force re-authentication upon the increase of risk score.

→ Export selected Azure AD events to a local machine for further analytics

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


1) Users authenticate via Azure Active Directory
Azure Active Directory
Graph API
Forcepoint CASB
5) Azure Active Directory adjusts authentication steps based on risk score provided by Forcepoint CASB
Risk Score Manager
Risk Score Handler
Event Logger
Risk Score Exporter
2) Extracts risk score of each Azure AD user known to Forcepoint CASB
4) Calls Graph API to adjust authentication policies based on risk score.
Terminate active sessions to force re-authentication in case of risk score increase.
3) Exports Azure AD events and stores locally for further analytics

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

→ Forcepoint CASB - 2019 R2

→ Azure Active Directory

This interoperability uses the **Risk Score** of Forcepoint CASB for Azure AD users to change the login policies for the Azure AD users.

The following activities are out of the scope of this document and therefore left to the system administrator, as part of ordinary maintenance procedures to be put in place within the existing infrastructure.

→ configuration of appropriate hygiene procedures to handle logs produced during any step of the solution workflow

→ monitoring of the scripts, services, and applications involved in the solution

Implementation options
Two implementation options are provided in this document

- → Docker – leverages docker images where the integration component is already installed with all necessary dependencies.
- → Traditional – requires the manual deployment of the integration component inside a clean Centos 7 host-machine.

The docker host machine must meet the minimum hardware requirements of 2GB of RAM and 20GB of storage. The docker images for this integration have been tested working with Docker 19.03.6

while the traditional version of this integration has been tested working with the following requirements

- → Centos 7.3 with at least 2 GB RAM and 20 GB of storage
- → Golang v1.14
- → Python3
- → Azure CLI v2.5.1

Groups into Azure Active Directory

Authentication steps applied to users authenticating through Azure Active Directory are defined as **Security policies** configured into **Groups**: a user authenticating through Azure Active Directory will be challenged according to the policies configured based on the user’s group membership.

Since risk score calculated by Forcepoint CASB have no upper limit (value can be anything from 0 to infinite depending on the user events identified by Forcepoint CASB), the **Risk Score Manager** maps ranges of risk scores into discrete risk levels, and each risk level is mapped to an Azure Active Directory group.

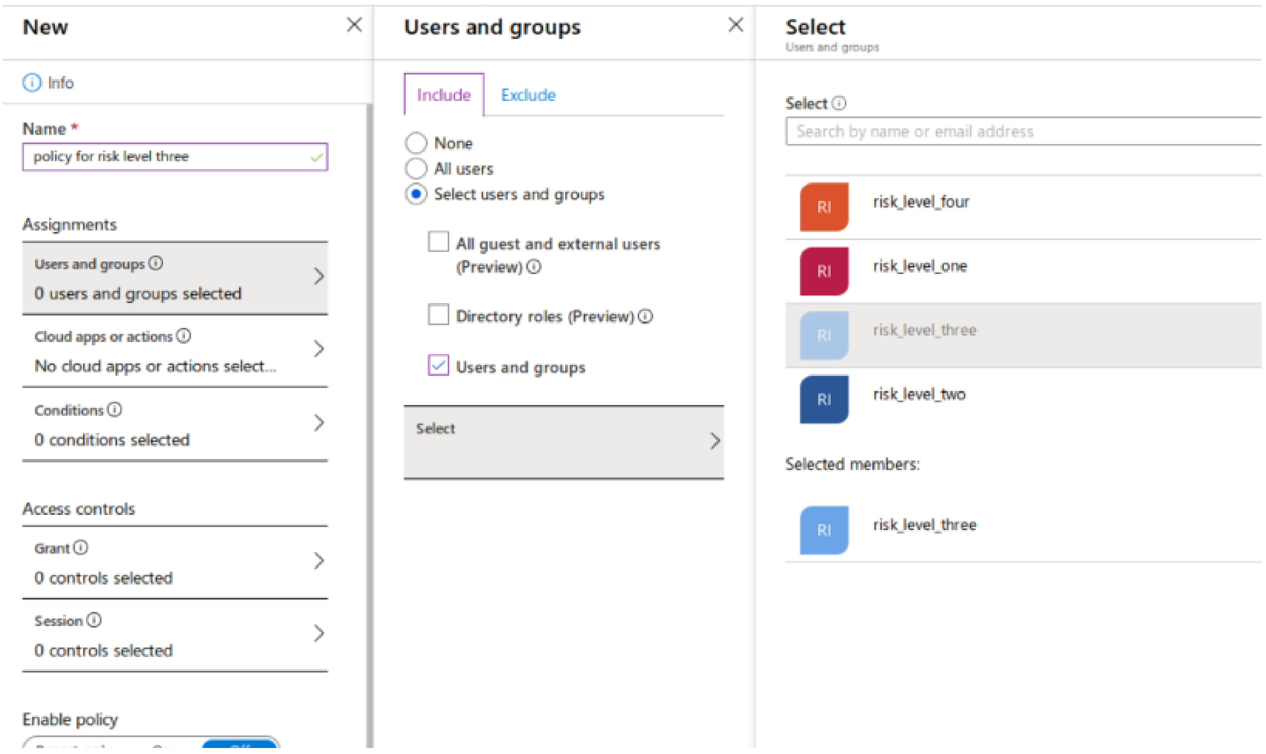
A typical mapping is as follows:

- → Users with risk score from 0 to 100 are mapped to risk_level 1, from 101 to 200 are mapped to risk level 2
  - risk level 1 and 2 (low risk) are assigned to groups with standard authentication policies
  - these groups are configured to authenticate users via username and password
- → Users with risk score from 201 to 300 are mapped to risk_level 3, from 301 to 400 are mapped to risk level 4
  - risk level 3 and 4 (medium risk) are assigned to groups with more complex authentication policies
  - these groups are configured to authenticate users via username and password first, then multi-factor authentication
- → Users with risk score from 401 upwards are mapped to risk_level 5 (high risk)
  - users assigned to this group will have their authentication requests denied.

If multiple user groups with existing policies are already configured into Azure Active Directory, then skip this part and move to the next chapter.

In the following example, we create a new user group for users with risk_level 3 and enable multi-factor authentication for all cloud apps:

1. Login to the Azure portal and go to **Azure Active Directory** > **Groups** > **New group**.
2. Select Security as Group type and enter risk_level_three as group name, leave Membership type to Assigned and click Create.
3. Go to Azure Active Directory > Security > Conditional Access > New policy.
4. Name your policy and assign it to the **risk_level_three** group, then click **Select** > **Done**.


New

Info

Name *
policy for risk level three

Assignments

Users and groups
0 users and groups selected

Cloud apps or actions
No cloud apps or actions select...

Conditions
0 conditions selected

Access controls

Grant
0 controls selected

Session
0 controls selected

Enable policy

Users and groups

Include | Exclude

None
All users
[✓ Select users and groups]

All guest and external users (Preview)
Directory roles (Preview)
[✓ Users and groups]

Select

Select
Users and groups

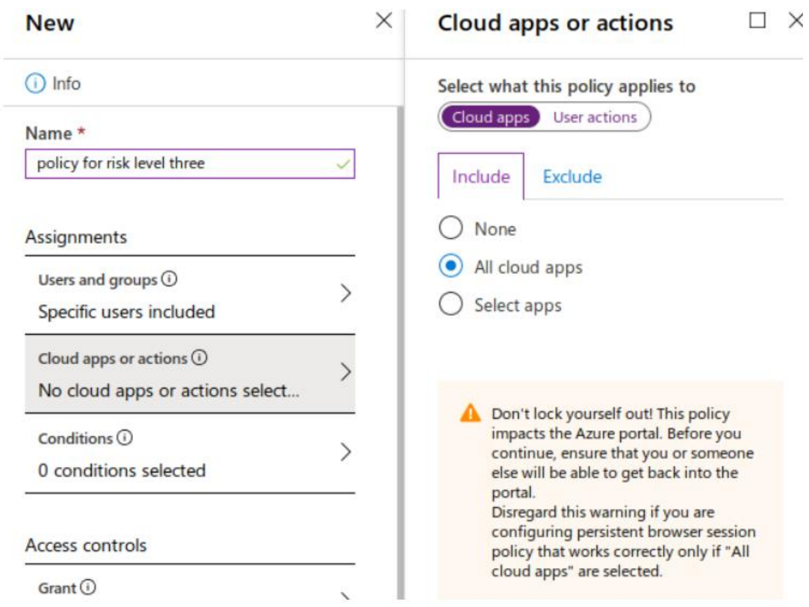
Select
Search by name or email address

risk_level_four
risk_level_one
risk_level_three
risk_level_two

Selected members:

risk_level_three

5. Click **Cloud apps** or **User actions** and select **All cloud apps** inside the **Include** tab, then click **Done**.


New

Info

Name *
policy for risk level three

Assignments

Users and groups
Specific users included

Cloud apps or actions
No cloud apps or actions select...

Conditions
0 conditions selected

Access controls

Grant

Cloud apps or actions

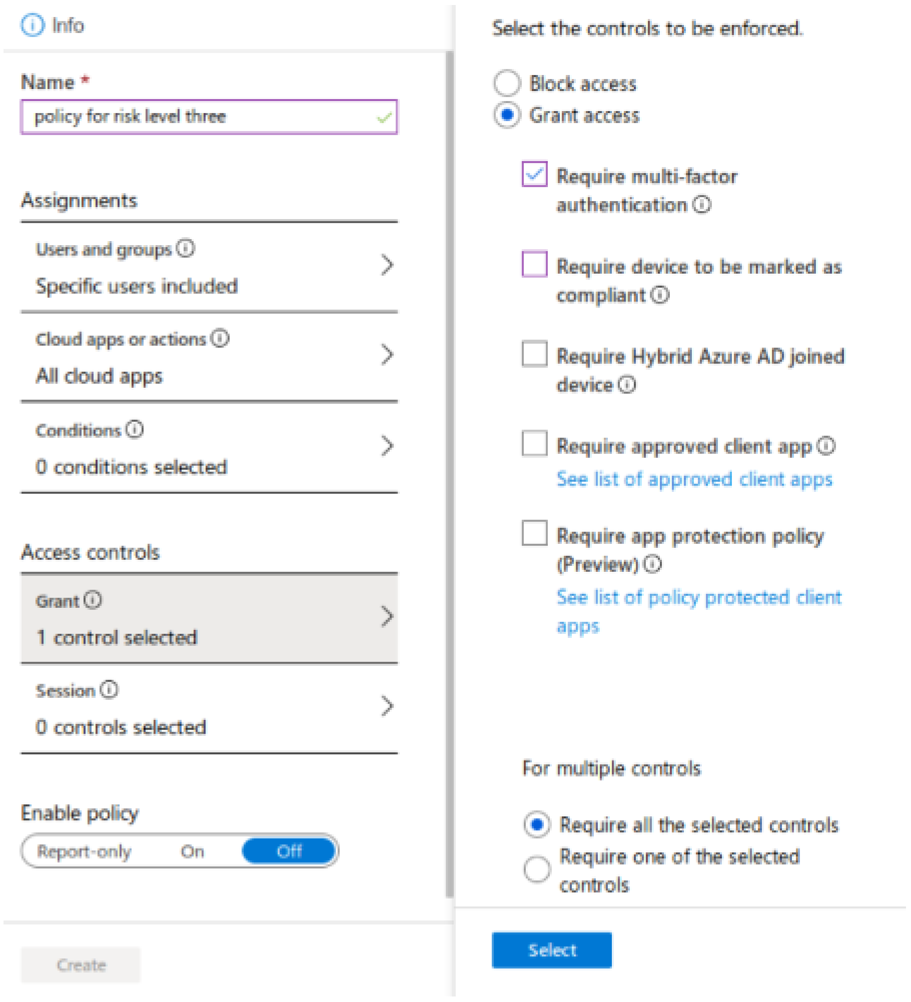
Select what this policy applies to
[✓ Cloud apps] User actions

Include | Exclude

None
[✓ All cloud apps]
Select apps

Don't lock yourself out! This policy impacts the Azure portal. Before you continue, ensure that you or someone else will be able to get back into the portal.
Disregard this warning if you are configuring persistent browser session policy that works correctly only if "All cloud apps" are selected.

6. Click Grant > Grant access and tick Require multi-factor authentication, then click Select.


Info

Name *
policy for risk level three

Assignments
Users and groups
Specific users included
Cloud apps or actions
All cloud apps
Conditions
0 conditions selected

Access controls
Grant
1 control selected
Session
0 controls selected

Enable policy
Report-only On [✓ Off]

Create

Select the controls to be enforced.

Block access
[✓ Grant access]

[✓ Require multi-factor authentication]
Require device to be marked as compliant
Require Hybrid Azure AD joined device
Require approved client app
See list of approved client apps
Require app protection policy (Preview)
See list of policy protected client apps

For multiple controls
[✓ Require all the selected controls]
Require one of the selected controls

Select

Under **Enable policy**, make sure to click **On** before clicking **Create**, otherwise the new policy will not be enforced.

Multi-factor authentication must be configured to be applied to any further authentication attempt. It is recommended for all users to be pre-enrolled for any authentication method that may be required. Multi-factor authentication can be configured by clicking **Azure Active Directory**> **Security**> **MFA**.

Repeat the steps above to create additional groups, each configured with specific authentication policies: each group will be mapped to Forcepoint CASB risk scores as explained in the rest of this document.

Risk Level Manager Configuration File.

All parameters required by the Risk Level Manager to operate its services are stored in a single file called **azure_casb.yml** file.

```
#Azure Risk Level Groups separated by a comma
AZURE_GROUPS_NAME: risk_level_1,risk_level_2,risk_level_3,risk_level_4,risk_level_5
#Map a risk score range to a Risk Level Group.
MAP_RISK_SCORE:
  - 100-200: risk_level_1
  - 201-400: risk_level_2
  - 401-500: risk_level_3
  - 501-1000: risk_level_4
  # equal and greater than 1001. Add plus sign to the end of the value
  - 1001+: risk_level_5
#risk Score URL. DO NOT CHANGE THIS
RISK_SCORE_URL:
https://my.skyfence.com/cm/rs/0/human_risk/accounts/reports/csv?search=%2BriskScore%3A(%22%5B1%20
TO%20*%5D%22)&sortBy=riskScore&sortDirection=DESC
#Your Forcepoint CASB instance username
CASB_USER_NAME: INSERT_YOUR_FORCEPOINT_CASB_USERNAME_HERE
#Your Forcepoint CASB instance password
CASB_PASSWORD: INSERT_YOUR_FORCEPOINT_CASB_PASSWORD_HERE
# logs output format JSON
LOGGER_JSON_FORMAT: false
#Terminate the user's active session if the user's risk level changes upon risk score increase
TERMINATE_USER_ACTIVE_SESSION: true
#Download the risk score from Forcepoint CASB every 'RISK_MANAGER_INTERVAL_TIME' minutes
RISK_MANAGER_INTERVAL_TIME: 10
```

The following table describes parameters in the **azure_casb.yml** file

Parameter	Description	Requires to be changed
AZURE_GROUPS_NAME	Names of the risk level groups of Azure AD separated by a comma	YES
MAP_RISK_SCORE	Mapping a range of risk scores to risk level groups	Yes
RISK_SCORE_URL	Forcepoint CASB risk score CSV file URL	NO
CASB_USER_NAME	Forcepoint CASB login username	YES
CASB_PASSWORD	Forcepoint CASB login password	YES
LOGGER_JSON_FORMAT	Change the logs output to JSON format	NO
TERMINATE_USER_ACTIVE_SESSION	Terminate user’s active session when user’s risk level group is changed	NO
RISK_MANAGER_INTERVAL_TIME	Download the risk score from Forcepoint CASB every 'RISK_MANAGER_INTERVAL_TIME' minutes.	NO

Implementation – Docker

The solution described in this chapter requires

→ A Linux machine (Centos 7.3 recommended with at least 2 GB RAM and 20 GB of storage). This machine will be referenced in the rest of this document as the **docker-host** machine.

→ Docker Engine must be installed on the **docker-host** machine, visit docker-installation-docs to install Docker Engine on **docker-host**

Step 1: Create Config File

1. Inside **docker-host** create a file named **azure_casb.yml** under **/root** directory

```
vi /root/azure_casb.yml
```

2. Add the following lines to **azure_casb.yml**.

```
#Azure Risk Level Groups separated by a comma
AZURE_GROUPS_NAME: risk_level_1,risk_level_2,risk_level_3,risk_level_4,risk_level_5
#Map a risk score range to a Risk Level Group
MAP_RISK_SCORE:
  - 100-200: risk_level_1
  - 201-400: risk_level_2
  - 401-500: risk_level_3
  - 501-1000: risk_level_4
  # equal and greater than 1001. Add plus sign to the end of the value
  - 1001+: risk_level_5
#risk Score URL. DO NOT CHANGE THIS
RISK_SCORE_URL:
https://my.skyfence.com/cm/rs/0/human_risk/accounts/reports/csv?search=%2BriskScore%3A(%22%5B1%20TO%2
0*%5D%22)&sortBy=riskScore&sortDirection=DESC
#Your Forcepoint CASB instance username
CASB_USER_NAME: INSERT_YOUR_FORCEPOINT_CASB_USERNAME_HERE
#Your Forcepoint CASB instance password
CASB_PASSWORD: INSERT_YOUR_FORCEPOINT_CASB_PASSWORD_HERE
# logs output format as json
LOGGER_JSON_FORMAT: false
#Terminate user's active session if the user's risk level group is been changed
TERMINATE_USER_ACTIVE_SESSION: true
#Download the risk score from Forcepoint CASB every 'RISK_MANAGER_INTERVAL_TIME' minutes
RISK_MANAGER_INTERVAL_TIME: 10
```

3. Replace the value of AZURE_GROUPS_NAME with your Azure risk level groups.
4. Modify the value for parameter MAP_RISK_SCORE to map risk level groups to a range of Forcepoint CASB risk score.
5. Insert your Forcepoint CASB username as a value for CASB_USER_NAME parameter.
6. Insert your Forcepoint CASB password as a value for CASB_PASSWORD parameter.
7. Save **azure_casb.yml** and move to the next step

Step 2: Download Docker Image and Run Docker container

1. Use the following command and credentials to login into the Docker registry hosting the containers needed for this integration

```
root@linux:~# docker login docker.frcpnt.com
Username: fp-integrations
Password: t1knmAkn19s
```

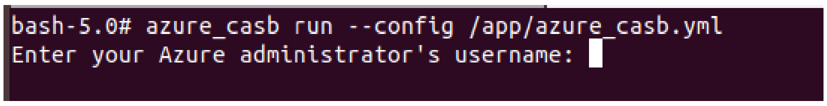
2. Run the following command which downloads the docker image and runs **docker.frcpnt.com/fp-casb-export-azure-ad** docker container in interactive mode:

```
docker run -v /root/azure_casb.yml:/app/azure_casb.yml -it docker.frcpnt.com/fp-
casb-export-azure-ad
```

3. Run the following command to start **Risk Level Manager**:

```
azure_casb run --config /app/azure_casb.yml
```

The output of the command will look like this:



```
bash-5.0# azure_casb run --config /app/azure_casb.yml
Enter your Azure administrator's username:
```

4. Enter your Azure administrator login username and password.

If you don't want to manually type your Azure credentials, add these parameters inside **azure_casb.yml** file replacing the red text with your Azure login information:

```
AZURE_ADMIN_LOGIN_NAME: INSERT_YOUR_AZURE_USERNAME_HERE

AZURE_ADMIN_LOGIN_PASSWORD: INSERT_YOUR_AZURE_PASSWORD_HERE
```

Once the steps above are completed, for each user in Forcepoint CASB with a risk level, user's risk level group membership in Azure will be changed within a few minutes and authentication policies will be enforced at next login.

Implementation - Traditional

The solution described in this chapter requires:

→ Centos 7.3 machine with at least 2 GB RAM and 20 GB of storage. This machine will be referenced in the rest of this document with the name **host-machine**.

→ The source files for this implementation, contained in the archive **fp-casb-export-azure-ad.tar.gz** available at this link: https://frcpnt.com/fp-casb-export-azure-ad-latest

The following dependencies installed inside the **host-machine**

→ Golang v1.14

→ Azure CLI 2.5.1

→ Python3

The archive **fp-casb-export -azure-ad.tar.gz** contains the following files and folders:

→ **azure_casb**: the Risk Level Manager application

→ **azure_casb.yml**: the config file for Risk Level Manager

→ **azure_casb.service**: Systemd service for **azure_casb**.service

→ **installation.sh**: installer to install all required dependences

Step 1: Install Dependencies

1. Inside the host-machine unpack the fp-casb-export-azure-ad.tar.gz archive and change your directory to **fp-casb-export-azure-ad**

```
tar -zxvf fp-casb-export-azure-ad.tar.gz
```

```
cd fp-casb-export-azure-ad
```

2. Execute the following command to make **installation.sh** executable

```
chmod +x installation.sh
```

   installation.sh will create systemd services for **azure_casb** service and will install the required dependencies.

3. Execute **installation.sh** script

```
sudo ./installation.sh
```

Step 2: Modify Configuration File.

Open **/var/azure_casb/azure_casb.yml** file and update the following parameters:

1. **AZURE_GROUPS_NAME**: insert here your Azure risk level groups separated by a comma
2. **MAP_RISK_SCORE**: map a risk score range to a risk level group
3. **CASB_USER_NAME**: Insert your Forcepoint CASB username here
4. **CASB_PASSWORD**: Insert your Forcepoint CASB password here
5. **AZURE_ADMIN_LOGIN_NAME**: Insert your Azure administrator username here
6. **AZURE_ADMIN_LOGIN_PASSWORD**: insert your Azure administrator password here.

close and save **azure_casb.yml** file.

Step 3: Reboot Host-Machine

Reboot the **host-machine** and verify systemd service **azure_casb.service** is running without any problem by executing this command:

```
systemctl list-units | grep azure_casb
```

The output of the command will be to the below:

```
azure_casb.service                                    loaded active running   Forcepoint CASB and Azure AD
```

Once the steps above are completed, for each user in Forcepoint CASB with a risk level, user's risk level group membership in Azure will be changed within a few minutes and authentication policies will be enforced at next login.

Exporting Azure activity events (with docker implementation only)

A CLI tool **azure-logs** is provided with this integration to interact with Azure Active directory in order to export Azure activity logs. Currently it supports exporting Azure sign-in logs only and is provided as a standalone docker container.

Azure-logs tool has the following features:

→ Filtering logs by log's creation date-time

→ Filtering logs by username for one user or multiple users

→ Supported output format are JSON and YAML

→ Saving logs to a local machine

Download and run azure-logs container

In your **docker-host** machine:

1. Use the following command and credentials to login into the Docker registry hosting the container

```
docker login docker.frcpnt.com
Username: fp-integrations
Password: t1knmAkn19s
```

2. Download and run the **azure-logs** container. The below command starts **azure-logs** applications and mount host-machine **/root/azure_logs** folder into the path **/app/azure_logs** within the container.

```
docker run -v /root/azure_logs:/app/azure_logs -it docker.frcpnt.com/fp-azure-logs-exporter
```

Azure-logs usage

The following command displays the help page for sign-ins sub-command:

```
#$ azure-logs sign-ins -h

Retrieve a specific Azure AD user sign-in event for your Azure tenant.

Usage:

  azure-logs sign-ins [flags]

Flags:

      --datetime-eq string     logs create datetime equal. allowed format is: YYYY-MM-DDThh:mm:ss or YYYY-MM-DD.

                               for example: 2020-04-23T14:50:32

                                            2020-03-26

      --datetime-ge string     logs create datetime greater than or equal. allowed format is: YYYY-MM-DDThh:mm:ss or YYYY-MM-
DD.

                               for example: 2020-04-23T14:50:32

                                            2020-03-26

      --datetime-gt string     logs create datetime greater than. allowed format is: YYYY-MM-DDThh:mm:ss or YYYY-MM-DD.

                               for example: 2020-04-23T14:50:32

                                            2020-03-26

      --datetime-le string     logs create datetime less than or equal. allowed format is: YYYY-MM-DDThh:mm:ss or YYYY-MM-DD.

                               for example: 2020-04-23T14:50:32

                                            2020-03-26

      --datetime-lt string     logs create datetime less than. allowed format is: YYYY-MM-DDThh:mm:ss or YYYY-MM-DD.
```

```
                               for example: 2020-04-23T14:50:32
                                            2020-03-26
      --datetime-ne string     logs create datetime not equal. allowed format is: YYYY-MM-DDThh:mm:ss or YYYY-MM-DD.
                               for example: 2020-04-23T14:50:32
                                            2020-03-26
  -h, --help                   help for sign-ins
  -o, --output-format string   the output format. possible formats: json, jsonc, yaml, yamlc (default "json")
  -s, --save string            Save the output to a specific given file
  -u, --username string        Users/User Principal Name. for multiple Users use comma separated values. for example:
                               user.one@yourdomain.com,user.two@yourdomain.com

Global Flags:
  -g, --graph-version string   Microsoft Graph API version. Default value is v1.0 (default "v1.0")
```

Sign-ins sub-command exports Azure Active Directory sign-ins activity logs.

When sign-ins sub-command is called for the first time, user will be asked to provide Azure credentials (username and password). Each following run sign-ins Azure credentials won't be necessary unless the active session between Azure CLI instance (on the user machine) and Azure is expired.

By default, sign-ins sub-command will use Microsoft Graph API version 1.0. the flag **--graph-version** can be used to specify a different version

```
#$ azure-logs sign-ins
Enter your Azure administrator's username: dlo.bagari@yourdomin.onmicrosoft.com
Enter password for 'dlo.bagari@yourdomin.onmicrosoft.com' and press Enter:
```

→ The output of the above command provides all sign-ins logs in your Azure instance for all users.

Since the unfiltered output can be extremely verbose, filtering capabilities are provided.

Filter logs by created date-time

Allowed date time format are: **YYYY-MM-DDThh:mm:ss** or **YYYY-MM-DD**.

```
azure-logs sign-ins --datetime-gt 2020-04-13T12:23:23
```

Filter logs by user

The flag **--username/-u** can be passed to sign-ins sub-command to filter logs by user/users. This flag expects the username or multiple username separated by a comma.

```
azure-logs sign-ins --username dlo.bagari@yourdomin.onmicrosoft.com
```

Output format

Azure-logs sign-ins supports two different output formats: JSON and YAML. The default output format is JSON. The flag **–output-format** can be used to change the output format.

```
azure-logs sign-ins --username dlo.bagari@yourdomin.onmicrosoft.com --output-format yaml
```

Output to file

The flag **--save** can be used to save the output of sign-ins sub-command into a file.

```
azure-logs sign-ins--datetime-eq 2020-04-13T12:23:23 –save /app/azure_logs/sign_in_logs.json
```

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 version of Forcepoint CASB in use is listed as compatible

Forcepoint CASB - 2019 R2

→ Docker images for this integration have been tested with

Docker 19.03.6

→ The docker implementation has been tested on a CentOS 7.3 machine with at least 2 GB RAM, 20 GB of storage and docker engine installed

→ User should have sudo permissions in the docker host machine

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 CASB: execute the following command on docker host machine:

ping -c 2 http://my.skyfence.com/

Once done check the result is similar to below:

```
PING http://my.skyfence.com/ (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 dependencies are installed

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

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

docker info

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

```
Client:
Debug Mode: false

Server:
Containers: 3
  Running: 2
  Paused: 0
  Stopped: 1
Images: 3
```

Server Version: 19.03.8

Traditional Implementation

Validate the prerequisites

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

→ Check the version of Forcepoint CASB in use is listed as compatible

Forcepoint CASB - 2019 R2

→ Verify the integration is correctly operating on a CentOS 7.3 machine with at least 2 GB RAM and 20 GB of storage

→ User needs sudo permissions for installing the dependencies and **azure_casb** service

→ Check the user can download the file with the below command:

wget --content-disposition https://frcpnt.com/fp-casb-export-azure-ad-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 CASB: execute the following command on docker host machine:

ping -c 2 http://my.skyfence.com/

Once done check the result is similar to below:

```
PING http://my.skyfence.com/ (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 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 host machine to check go is installed:

go version

Check the output is similar to below:

```
go version go1.14.1 linux/amd64
```

→ Check Azure CLI is installed: Execute following command on host machine:

az version

Check the output is similar to below:

```
{
  "Azure-cli": "2.3.1",
  "Azure-cli-command-modules-nspkg": "2.0.3",
  "Azure-cli-core": "2.3.1",
  "Azure-cli-nspkg": "3.0.4",
  "Azure-cli-telemetry": "1.0.4",
  "extensions": {}
```

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

python3 --version

Check the output is similar to below:

Python 3.6.x

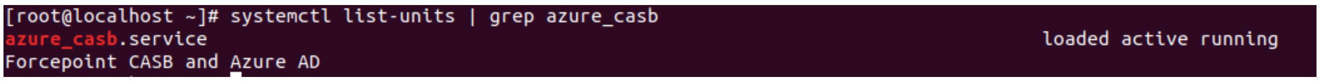
Check all components are configured and running properly

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

→ Check systemd service **azure_casb.service** is running without any problem by executing this command:

systemctl list-units | grep azure_casb

Verify the output is similar to below:



```
[root@localhost ~]# systemctl list-units | grep azure_casb
azure_casb.service                                                                            loaded active running
Forcepoint CASB and Azure AD
```