複数のエンドポイントでの開始/停止の分離を自動化

内容

はじめに

このドキュメントでは、Cisco Secure Endpoint用のAPIを使用して、複数のエンドポイントの停止/開始分離を自動化する方法について説明します。

前提条件

要件

次の項目に関する知識があることが推奨されます。

- Cisco Secure Endpoint
- Cisco Secure Endpointコンソール
- シスコセキュアエンドポイントAPI
- Python

使用するコンポーネント

このドキュメントの情報は、次のソフトウェアバージョンに基づくものです。

- Cisco Secureエンドポイント8.4.0.30201
- Python環境をホストするエンドポイント
- Python 3.11.7

このドキュメントの情報は、特定のラボ環境にあるデバイスに基づいて作成されました。このドキュメントで使用するすべてのデバイスは、クリアな（デフォルト）設定で作業を開始しています。本稼働中のネットワークでは、各コマンドによって起こる可能性がある影響を十分確認して

ください。

背景説明

- PUT要求を使用して分離を開始します。
- DELETE要求は、切り離しを停止するために使用されます。
- 詳細は、APIのドキュメントを参照してください。

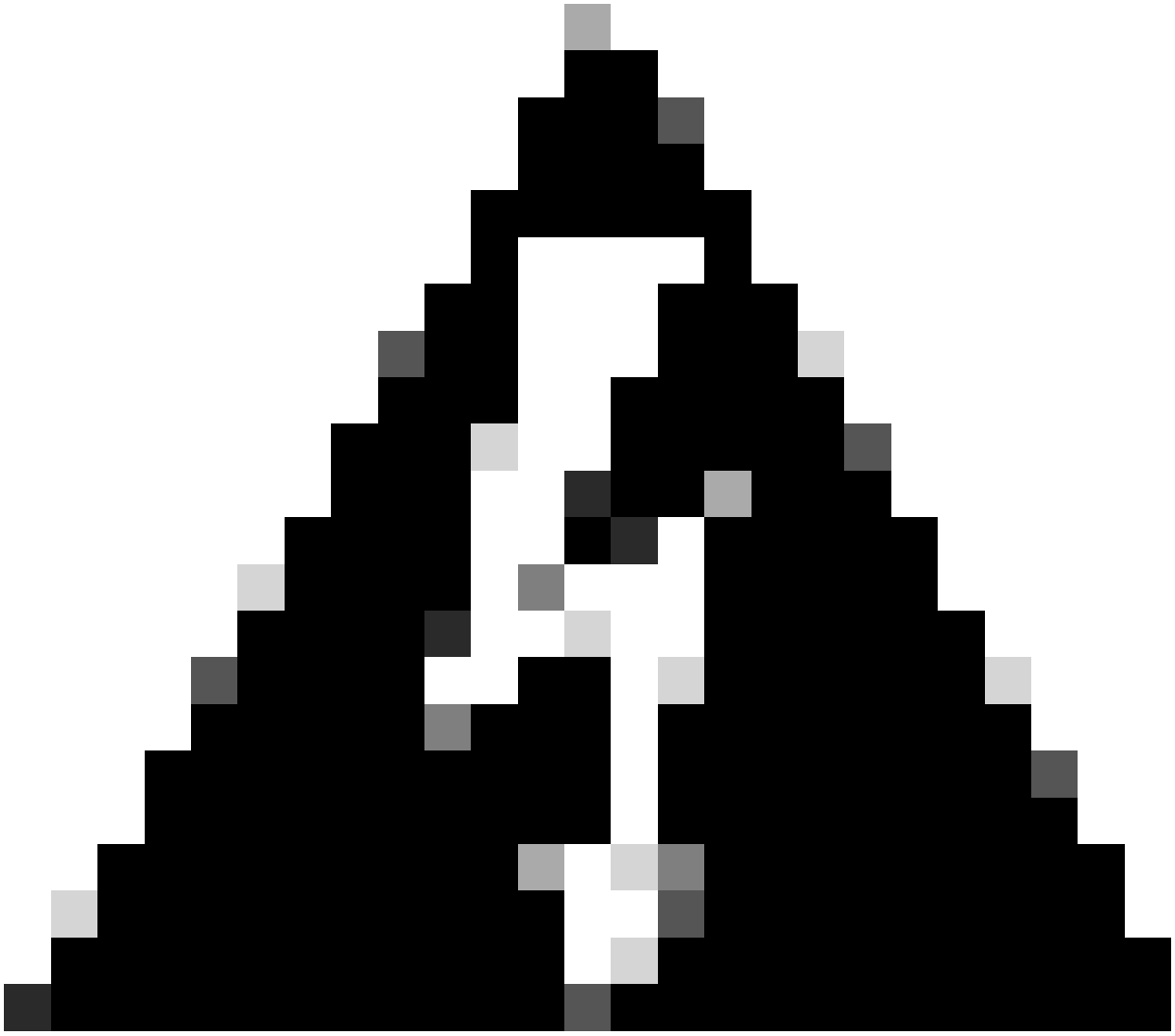
問題

Cisco Secure Endpointでは、一度に1台のマシン上で分離の開始/停止が可能です。ただし、セキュリティインシデントが発生した場合、潜在的な脅威を効果的に封じ込めるために、これらの操作を複数のエンドポイントで同時に実行する必要があります。APIを使用してバルクエンドポイントの開始/停止の分離プロセスを自動化すると、インシデント対応の効率が大幅に向上し、ネットワークに対する全体的なリスクが軽減されます。

解決方法

- この記事で提供されているPythonスクリプトを使用すると、セキュアエンドポイントAPIクレデンシャルを使用して、組織内の複数のエンドポイントで分離を開始/終了できます。
- AMP APIクレデンシャルを生成するには、「エンドポイント向けCisco AMP APIの概要」を参照してください。
- 提供されたスクリプトを使用するには、エンドポイントにpythonをインストールする必要があります。
- Pythonのインストール後、requestsモジュールをインストールしてください

```
pip install requests
```



警告：このスクリプトは説明のみを目的としており、APIを使用してエンドポイント分離機能を自動化する方法を示すことを目的としています。Cisco Technical Assistance Center(TAC)は、このスクリプトに関連する問題のトラブルシューティングには関与しません。スクリプトを実稼働環境に展開する前に、安全な環境で慎重にスクリプトをテストする必要があります。

スクリプト

提供されたスクリプトを使用して、ビジネス内の複数のエンドポイントで分離を開始できます。

```
import requests

def read_config(file_path):
    """
    Reads the configuration file to get the API base URL, client ID, and API key.
    """
    config = {}
    try:
```

```
        with open(file_path, 'r') as file:
            for line in file:
                # Split each line into key and value based on '='
                key, value = line.strip().split('=')
                config[key] = value
    except FileNotFoundError:
        print(f"Error: Configuration file '{file_path}' not found.")
        exit(1)  # Exit the script if the file is not found
    except ValueError:
        print(f"Error: Configuration file '{file_path}' is incorrectly formatted.")
        exit(1)  # Exit the script if the file format is invalid
    return config

def read_guids(file_path):
    """
    Reads the file containing GUIDs for endpoints to be isolated.
    """
    try:
        with open(file_path, 'r') as file:
            # Read each line, strip whitespace, and ignore empty lines
            return [line.strip() for line in file if line.strip()]
    except FileNotFoundError:
        print(f"Error: GUIDs file '{file_path}' not found.")
        exit(1)  # Exit the script if the file is not found
    except Exception as e:
        print(f"Error: An unexpected error occurred while reading the GUIDs file: {e}")
        exit(1)  # Exit the script if an unexpected error occurs

def isolate_endpoint(base_url, client_id, api_key, connector_guid):
    """
    Sends a PUT request to isolate an endpoint identified by the connector GUID.
    Args:
        base_url (str): The base URL for the API.
        client_id (str): The API client ID for authentication.
        api_key (str): The API key for authentication.
        connector_guid (str): The GUID of the connector to be isolated.
    """
    url = f"{base_url}/{connector_guid}/isolation"
    try:
        # Send PUT request with authentication
        response = requests.put(url, auth=(client_id, api_key))
        response.raise_for_status()  # Raise an HTTPError for bad responses (4xx and 5xx)

        if response.status_code == 200:
            print(f"Successfully isolated endpoint: {connector_guid}")
        else:
            print(f"Failed to isolate endpoint: {connector_guid}. Status Code: {response.status_code},
    except requests.RequestException as e:
        print(f"Error: An error occurred while isolating the endpoint '{connector_guid}': {e}")

if __name__ == "__main__":
    # Read configuration values from the config file
    config = read_config('config.txt')

    # Read list of GUIDs from the GUIDs file
    connector_guids = read_guids('guids.txt')

    # Extract configuration values
    base_url = config.get('BASE_URL')
    api_client_id = config.get('API_CLIENT_ID')
    api_key = config.get('API_KEY')
```

```
    # Check if all required configuration values are present
    if not base_url or not api_client_id or not api_key:
        print("Error: Missing required configuration values.")
        exit(1)  # Exit the script if any configuration values are missing

    # Process each GUID by isolating the endpoint
    for guid in connector_guids:
        isolate_endpoint(base_url, api_client_id, api_key, guid)
```

手順

- AMP APIクレデンシャルを生成するには、「エンドポイント向けCisco AMP APIの概要」を参照してください。
- 地域に関して記載されているBASE_URLを使用します。

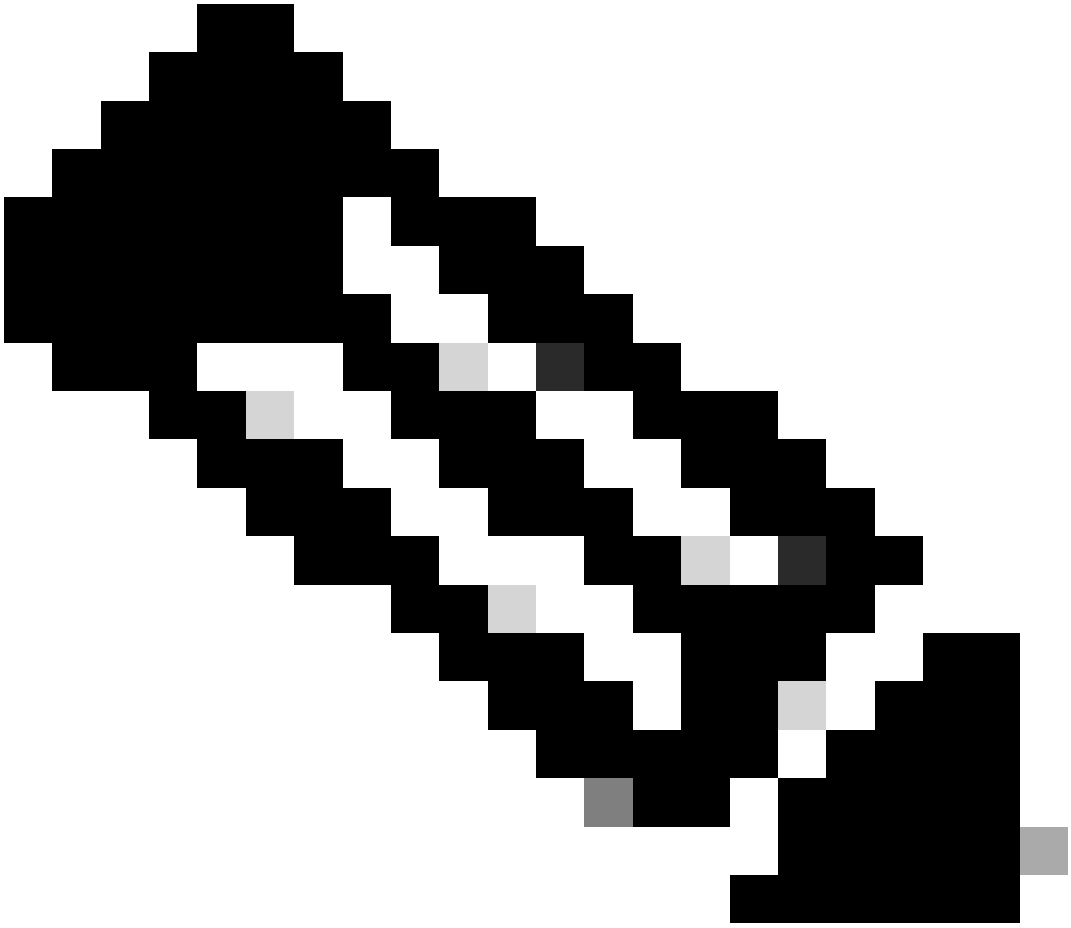
```
NAM - https://api.amp.cisco.com/v1/computers/
EU - https://api.eu.amp.cisco.com/v1/computers/
APJC - https://api.apjc.amp.cisco.com/v1/computers/
```

- 上記の内容のスクリプトと同じディレクトリにconfig.txtファイルを作成します。config.txtファイルの例：

```
BASE_URL=https://api.apjc.amp.cisco.com/v1/computers/
API_CLIENT_ID=xxxxxxxxxxxxxxxxxxxx
API_KEY=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

- スクリプトと同じディレクトリに、コネクタGUIDのリスト（1行に1つ）を持つguids.txtファイルを作成します。必要な数のGUIDを追加します。guids.txtファイルの例：

```
abXXXXXXXXXXXXcd-XefX-XghX-X12X-XXXXXX567XXXXXXX
yzXXXXXXXXXXXXlm-XprX-XmnX-X34X-XXXXXX618XXXXXXX
```



注：エンドポイントのGUIDは、API GET /v1/computersを使用するか、Cisco Secure Endpoint ConsoleからManagement > Computersに移動して特定のエンドポイントのエントリを展開し、コネクタGUIDをコピーすることで収集できます。

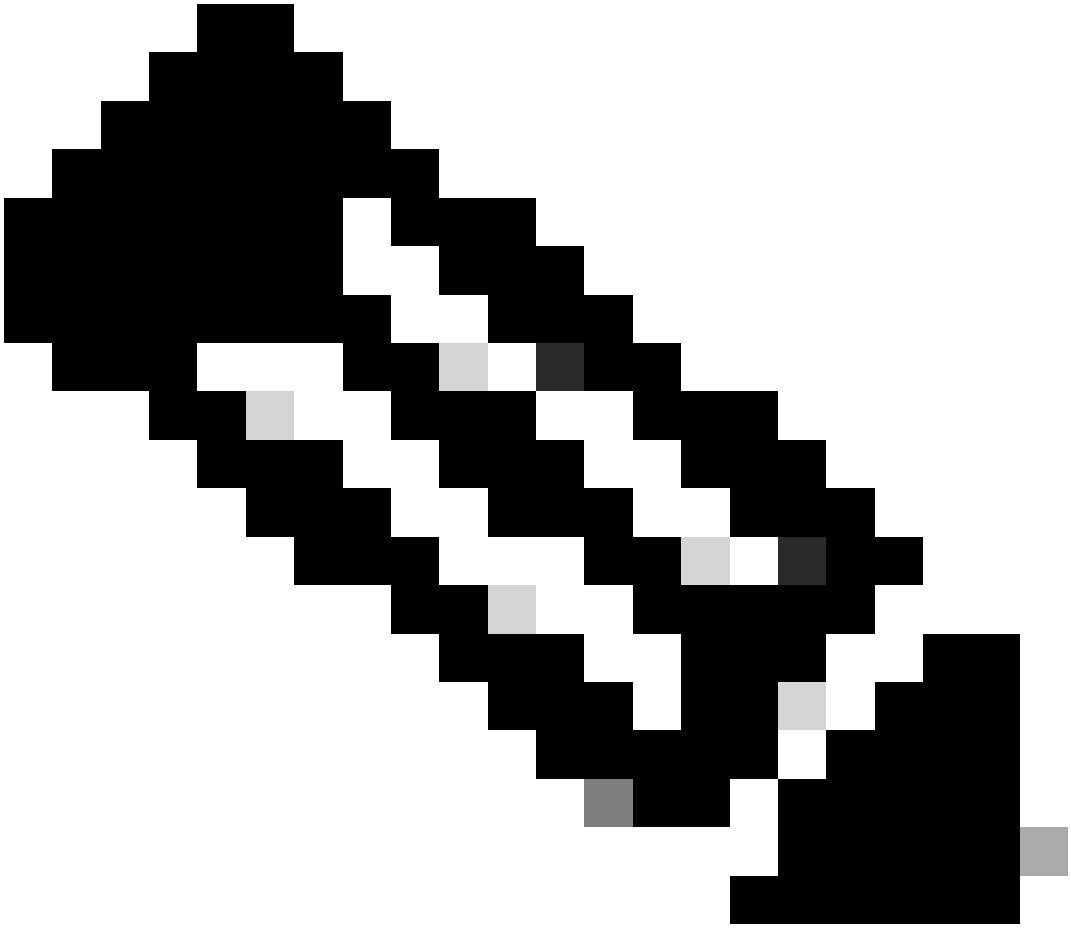
- ターミナルまたはコマンドプロンプトを開きます。start_isolation_script.pyがあるディレクトリに移動します。
- 次のコマンドを実行して、スクリプトを実行します。

```
python start_isolation_script.py
```

確認

- スクリプトは、guids.txtファイルで指定されている各エンドポイントを切り離そうとします。

- 各エンドポイントの端末またはコマンドプロンプトで、成功メッセージまたはエラーメッセージを確認します。



注：接続されたスクリプトstart_isolation.pyはエンドポイントで分離を開始するために使用できますが、stop_isolation.pyはエンドポイントで分離を停止するように設計されています。スクリプトを実行および実行する手順はすべて同じです。

翻訳について
シスコは世界中のユーザにそれぞれの言語でサポート コンテンツを提供するために、機械と人による翻訳を組み合わせて、本ドキュメントを翻訳しています。ただし、最高度の機械翻訳であっても、専門家による翻訳のような正確性は確保されません。シスコは、これら翻訳の正確性について法的責任を負いません。原典である英語版（リンクからアクセス可能）もあわせて参照することを推奨します。