MISP SOP

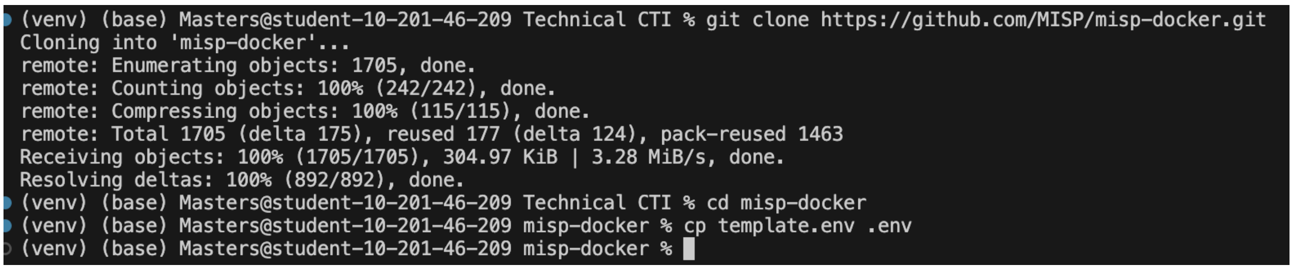
Table of Contents

1. Clone misp-docker github repository and initialise .env

Command:

```
git clone https://github.com/MISP/misp-docker.git
cd misp-docker
cp template.env .env
```

Screenshot:



```
(venv) (base) Masters@student-10-201-46-209 Technical CTI % git clone https://github.com/MISP/misp-docker.git
Cloning into 'misp-docker'...
remote: Enumerating objects: 1705, done.
remote: Counting objects: 100% (242/242), done.
remote: Compressing objects: 100% (115/115), done.
remote: Total 1705 (delta 175), reused 177 (delta 124), pack-reused 1463
Receiving objects: 100% (1705/1705), 304.97 KiB | 3.28 MiB/s, done.
Resolving deltas: 100% (892/892), done.
(venv) (base) Masters@student-10-201-46-209 Technical CTI % cd misp-docker
(venv) (base) Masters@student-10-201-46-209 misp-docker % cp template.env .env
(venv) (base) Masters@student-10-201-46-209 misp-docker %
```

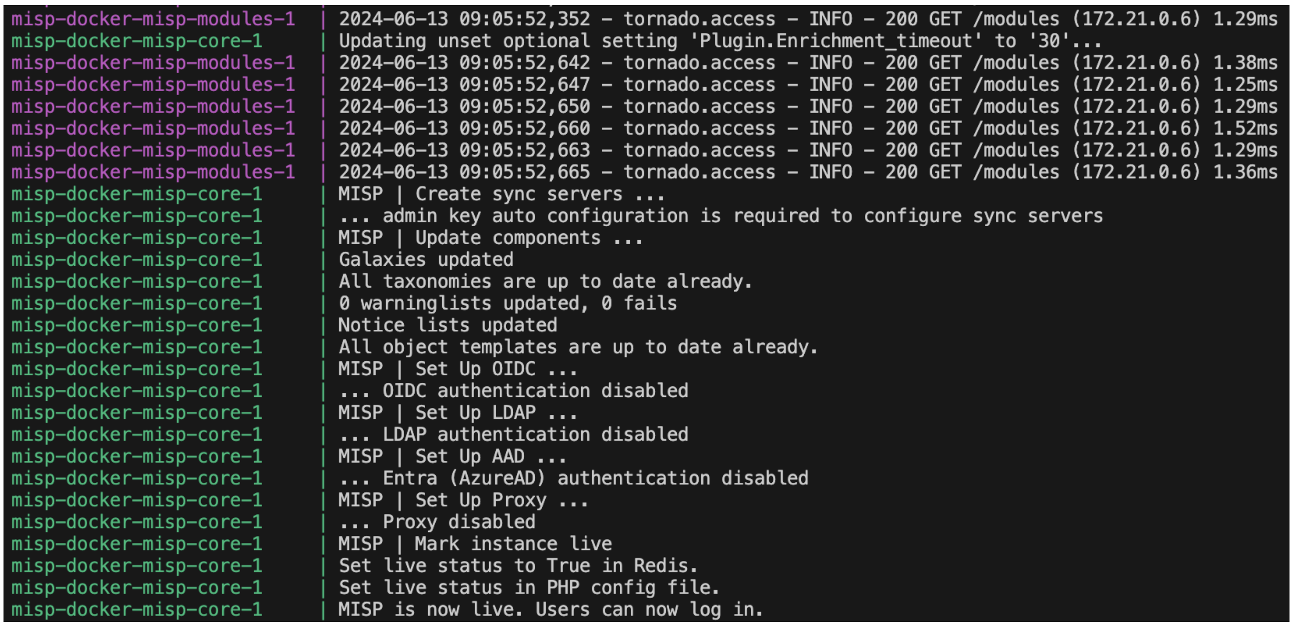
2. Setup docker container

Command:

```
docker-compose pull
docker-compose up
```

Note: It may take a several minutes to setup the docker container. Setup is over when it says "MISP is now live. Users can now log in."

Screenshot:



```
misp-docker-misp-modules-1  | 2024-06-13 09:05:52,352 - tornado.access - INFO - 200 GET /modules (172.21.0.6) 1.29ms
misp-docker-misp-core-1     | Updating unset optional setting 'Plugin.Enrichment_timeout' to '30'...
misp-docker-misp-modules-1  | 2024-06-13 09:05:52,642 - tornado.access - INFO - 200 GET /modules (172.21.0.6) 1.38ms
misp-docker-misp-modules-1  | 2024-06-13 09:05:52,647 - tornado.access - INFO - 200 GET /modules (172.21.0.6) 1.25ms
misp-docker-misp-modules-1  | 2024-06-13 09:05:52,650 - tornado.access - INFO - 200 GET /modules (172.21.0.6) 1.29ms
misp-docker-misp-modules-1  | 2024-06-13 09:05:52,660 - tornado.access - INFO - 200 GET /modules (172.21.0.6) 1.52ms
misp-docker-misp-modules-1  | 2024-06-13 09:05:52,663 - tornado.access - INFO - 200 GET /modules (172.21.0.6) 1.29ms
misp-docker-misp-modules-1  | 2024-06-13 09:05:52,665 - tornado.access - INFO - 200 GET /modules (172.21.0.6) 1.36ms
misp-docker-misp-core-1     | MISP | Create sync servers ...
misp-docker-misp-core-1     | ... admin key auto configuration is required to configure sync servers
misp-docker-misp-core-1     | MISP | Update components ...
misp-docker-misp-core-1     | Galaxies updated
misp-docker-misp-core-1     | All taxonomies are up to date already.
misp-docker-misp-core-1     | 0 warninglists updated, 0 fails
misp-docker-misp-core-1     | Notice lists updated
misp-docker-misp-core-1     | All object templates are up to date already.
misp-docker-misp-core-1     | MISP | Set Up OIDC ...
misp-docker-misp-core-1     | ... OIDC authentication disabled
misp-docker-misp-core-1     | MISP | Set Up LDAP ...
misp-docker-misp-core-1     | ... LDAP authentication disabled
misp-docker-misp-core-1     | MISP | Set Up AAD ...
misp-docker-misp-core-1     | ... Entra (AzureAD) authentication disabled
misp-docker-misp-core-1     | MISP | Set Up Proxy ...
misp-docker-misp-core-1     | ... Proxy disabled
misp-docker-misp-core-1     | MISP | Mark instance live
misp-docker-misp-core-1     | Set live status to True in Redis.
misp-docker-misp-core-1     | Set live status in PHP config file.
misp-docker-misp-core-1     | MISP is now live. Users can now log in.
```

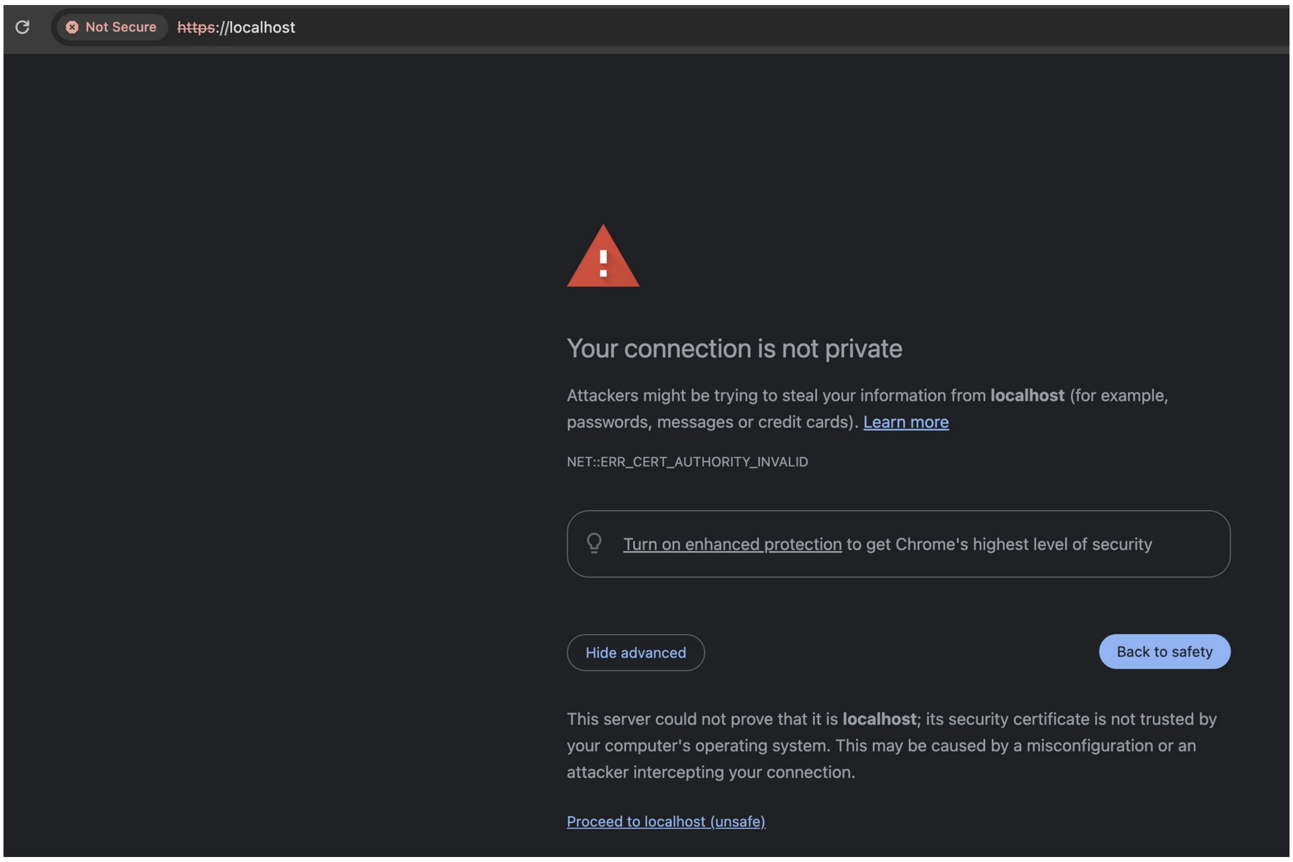
3. Login as admin@admin.test

Default URL: https://localhost

Default credential: admin@admin.test:admin

Note: Accept the self-signed certificate and “Proceed to localhost (unsafe)”

Screenshot:



Not Secure ~~https~~://localhost

Your connection is not private

Attackers might be trying to steal your information from **localhost** (for example, passwords, messages or credit cards). Learn more

NET::ERR_CERT_AUTHORITY_INVALID

Turn on enhanced protection to get Chrome's highest level of security

This server could not prove that it is **localhost**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

Proceed to localhost (unsafe)

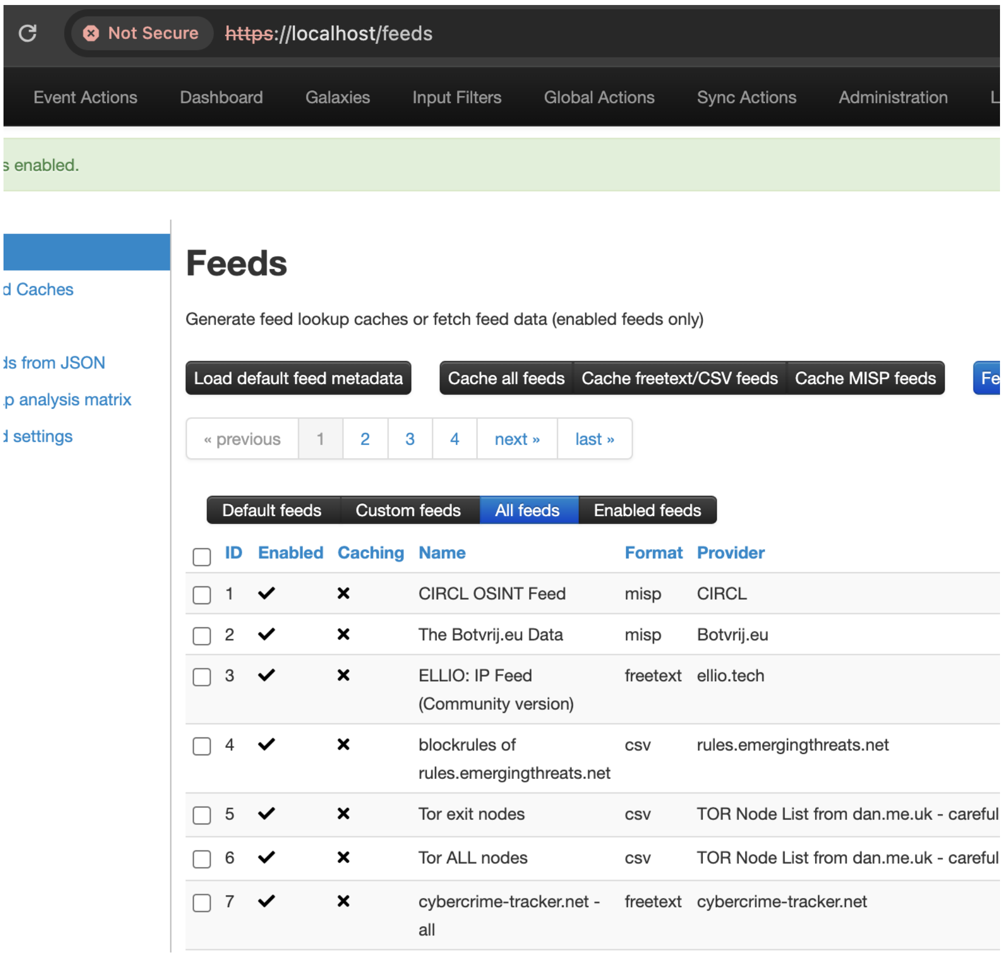
4. Load all default feeds

URL: https://localhost/Feeds

Steps:

1. Go to above provided URL
2. Select checkbox next to ID
3. Click on “Enable selected”
4. Click “Yes” to confirm to enable all selected feeds

Screenshot:



Feeds

Generate feed lookup caches or fetch feed data (enabled feeds only)

ID	Enabled	Caching	Name	Format	Provider
1	✓	✗	CIRCL OSINT Feed	misp	CIRCL
2	✓	✗	The Botvrij.eu Data	misp	Botvrij.eu
3	✓	✗	ELLIO: IP Feed (Community version)	freetext	ellio.tech
4	✓	✗	blockrules of rules.emergingthreats.net	csv	rules.emergingthreats.net
5	✓	✗	Tor exit nodes	csv	TOR Node List from dan.me.uk - careful
6	✓	✗	Tor ALL nodes	csv	TOR Node List from dan.me.uk - careful
7	✓	✗	cybercrime-tracker.net - all	freetext	cybercrime-tracker.net

5. Generate API keys to be used in script

URL: https://localhost/users/view/1

Steps:

1. Go to above provided URL
2. Click on “Auth keys”, then “Add authentication key”, then “Submit”
3. Copy the Auth keys and save it in a secure location to be used later in the script
4. Click on “I have noted down my key, take me back now”, leading to auth key view

6. Create .env and Run script

.env file:

```
MISP_URL=https://localhost
MISP_API_KEY=<auth_key>
```

Note: Copy the auth key from Step 5 to .env file, which is in the same directory as requirements.txt and main.py. The packages utilized in requirements.txt are provided in Appendix – A. The code for main.py is provided in Appendix – B.

Run the below commands:

Command:

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
sigma plugin install elasticsearch
python3 main.py &
```

Appendix A – requirements.txt

```
requests==2.32.3
python-dotenv==1.0.1
schedule==1.2.2
PyYAML==6.0.1
sigma-cli==1.0.2
```

Appendix B – main.py

```
import requests
import json
import yaml
from datetime import datetime, timedelta, timezone
from dotenv import load_dotenv
import os
import time
import uuid
import schedule
import subprocess

# Load environment variables from .env file
load_dotenv()

# Get MISP URL and API key from environment variables
misp_url = os.getenv('MISP_URL')
api_key = os.getenv('MISP_API_KEY')

headers = {
    'Authorization': api_key,
    'Accept': 'application/json',
    'Content-Type': 'application/json'
}

# Function to fetch all feeds
def fetch_all_feeds():
    response = requests.post(f'{misp_url}/feeds/fetchFromAllFeeds', headers=headers,
verify=False)
    if response.status_code == 200:
        return response.json()
    else:
        raise Exception(f'Error fetching feeds: {response.status_code} - {response.text}')

# Function to search attributes
def fetch_attributes():
    # Create the search payload
    payload = {
        'returnFormat': 'json',
        'last': '1d'
    }

    # Make the API request with SSL verification disabled
    response = requests.post(f'{misp_url}/attributes/restSearch', headers=headers, json=payload,
verify=False)
```

```
    # Check the response status and save the results to a file
    if response.status_code == 200:
        misp_response = response.json()
        with open(f'misp_response.json', 'w') as file:
            json.dump(misp_response, file, indent=2)
    else:
        with open(f'error.log', 'w+') as file:
            file.write(f'Error: {response.status_code} - {response.text}')

def extract_and_save(original_file_path='misp_response.json'):
    types_to_save = ['md5', 'sha1', 'sha256', 'filename', 'url', 'hostname', 'domain', 'uri', 'ip-src', 'ip-
dst']
    data_dict = {type_: [] for type_ in types_to_save}

    # Read the JSON file
    with open(original_file_path, 'r') as file:
        data = json.load(file)

    # Extract data if the response key exists
    if 'response' in data:
        IoCs = data['response']

        # Iterate over each attribute and store values in the dictionary
        for item in IoCs['Attribute']:
            type_ = item.get('type')
            value = item.get('value')
            if type_ in types_to_save:
                data_dict[type_].append(value)

        # Get current date in YYYY-MM-DD format
        current_date = datetime.now().strftime('%Y-%m-%d')

        for type_, values in data_dict.items():
            if values:
                # Create the directory structure if it doesn't exist
                dir_path = os.path.join('log', current_date)
                os.makedirs(dir_path, exist_ok=True)

                # Write the values to the appropriate file
                file_path = os.path.join(dir_path, f'{type_}.txt')
                with open(file_path, 'w') as file:
                    for value in values:
                        file.write(value + '\n')

        # Generate Sigma rules
        generate_sigma_rules(data_dict, current_date)

        # Cleanup files
        cleanup_files(original_file_path, current_date)
```

```
def generate_sigma_rules(data_dict, current_date):
    sigma_template = """title: Detection rule for {type} on {date}
id: {id}
description: Detection rule for detection of {type} values
author: Sarfaraz Ahamed
date: {date}
status: stable
logsource:
    category: {category}
detection:
    selection:
        {type}:
            {conditions}
    condition: selection
falsepositives:
    - Unknown
level: low
"""

    file_logsource_types = ['md5', 'sha1', 'sha256', 'filename']
    network_logsource_types = ['url', 'hostname', 'domain', 'uri', 'ip-src', 'ip-dst']

    for type_, values in data_dict.items():
        if values:
            # Determine the logsource category based on the type
            category = 'file' if type_ in file_logsource_types else 'network_connection'

            rule_id = str(uuid.uuid4())
            title = f'Detection rule for {type_} on {current_date}'
            description = f'detection of {type_} values'

            conditions = "\n            ".join([f"- '{value}'" for value in values])

            sigma_rule = sigma_template.format(
                type=type_,
                id=rule_id,
                description=description,
                date=current_date,
                category=category,
                conditions=conditions
            )

            # Save the Sigma rule with the same name as the text file but with .yml extension
            filename = f"{category}_{type_}_{rule_id}.yml"
            file_path_yml = os.path.join('log', current_date, filename)
            with open(file_path_yml, 'w') as file:
                file.write(sigma_rule)
```

```
def cleanup_files(original_file_path, current_date):
    # Delete all non .yml files in the log directory
    log_dir = os.path.join('log', current_date)
    for filename in os.listdir(log_dir):
        if not filename.endswith('.yml'):
            os.remove(os.path.join(log_dir, filename))

    # Delete all folders within log folder that are older than 30 days
    log_root_dir = 'log'
    thirty_days_ago = datetime.now() - timedelta(days=30)

    for foldername in os.listdir(log_root_dir):
        folder_path = os.path.join(log_root_dir, foldername)
        if os.path.isdir(folder_path):
            folder_mod_time = datetime.fromtimestamp(os.path.getmtime(folder_path))
            if folder_mod_time < thirty_days_ago:
                # Recursively delete the folder
                for root, dirs, files in os.walk(folder_path, topdown=False):
                    for name in files:
                        os.remove(os.path.join(root, name))
                    for name in dirs:
                        os.rmdir(os.path.join(root, name))
                os.rmdir(folder_path)

def generate_elk_query_from_sigma(file_path, query_dir):
    try:
        # Use Sigma CLI to convert Sigma rule to Elasticsearch query
        result = subprocess.run(
            ['sigma', 'convert', '-t', 'eql', file_path, '-p', 'ecs_windows'], #'--without-pipeline'],
            capture_output=True,
            text=True,
            check=True
        )

        elk_query = result.stdout

        # Generate a unique filename for the query
        query_filename = f"{os.path.splitext(os.path.basename(file_path))[0]}_query.txt"
        query_file_path = os.path.join(query_dir, query_filename)

        # Save the query to a separate .txt file
        with open(query_file_path, 'w') as query_file:
            query_file.write(elk_query)
    except Exception as e:
        print(f"Failed to generate ELK query from {file_path}: {e}")

# Function to be scheduled
def scheduled_task():
    try:
        print("Fetching all feeds...")
        feeds_response = fetch_all_feeds()
        print("Feeds fetched successfully.")

        # Sleep for 3 minutes
        print("Sleeping for 3 minutes...")
        time.sleep(180)

        # Search attributes only if fetch is successful
        print("Fetching attributes...")
        fetch_attributes()
        print("Attributes searched successfully.")

        # Specify the path to your JSON file
        file_path = 'misp_response.json'

        extract_and_save(file_path)
        print("IoCs extracted successfully.")

        # Generate ELK queries from Sigma rules
        current_date = datetime.now().strftime('%Y-%m-%d')
        log_dir = os.path.join('log', current_date)

        for filename in os.listdir(log_dir):
            if filename.endswith('.yml'):
                file_path_yml = os.path.join(log_dir, filename)
                generate_elk_query_from_sigma(file_path_yml, log_dir)

    except Exception as e:
        print(f"Failed to fetch feeds or attributes: {e}")

# Schedule the task to run every day at 8:00 AM
schedule.every().day.at("08:00").do(scheduled_task)

# Keep the script running
while True:
    schedule.run_pending()
    time.sleep(1)
```