SOC Tools Integration Guide

SOC Tools to be Integrated

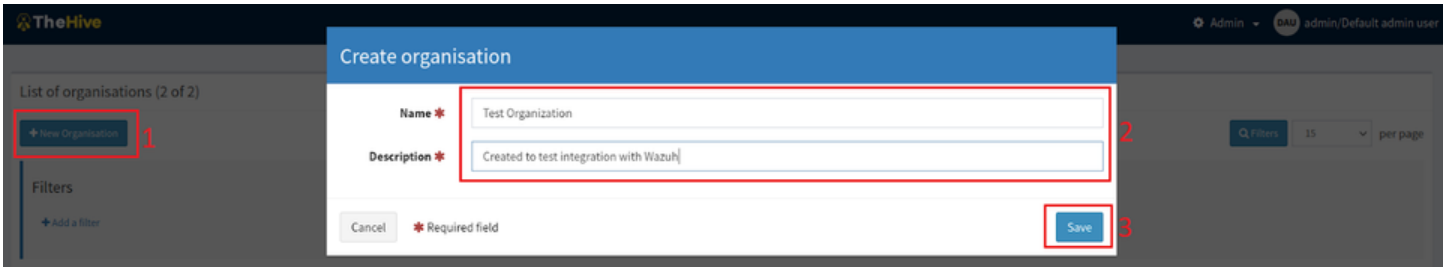
- Wazuh and TheHive
- TheHive and Cortex
- TheHive and MISP
- Wazuh and OpenVAS

Tools Integration

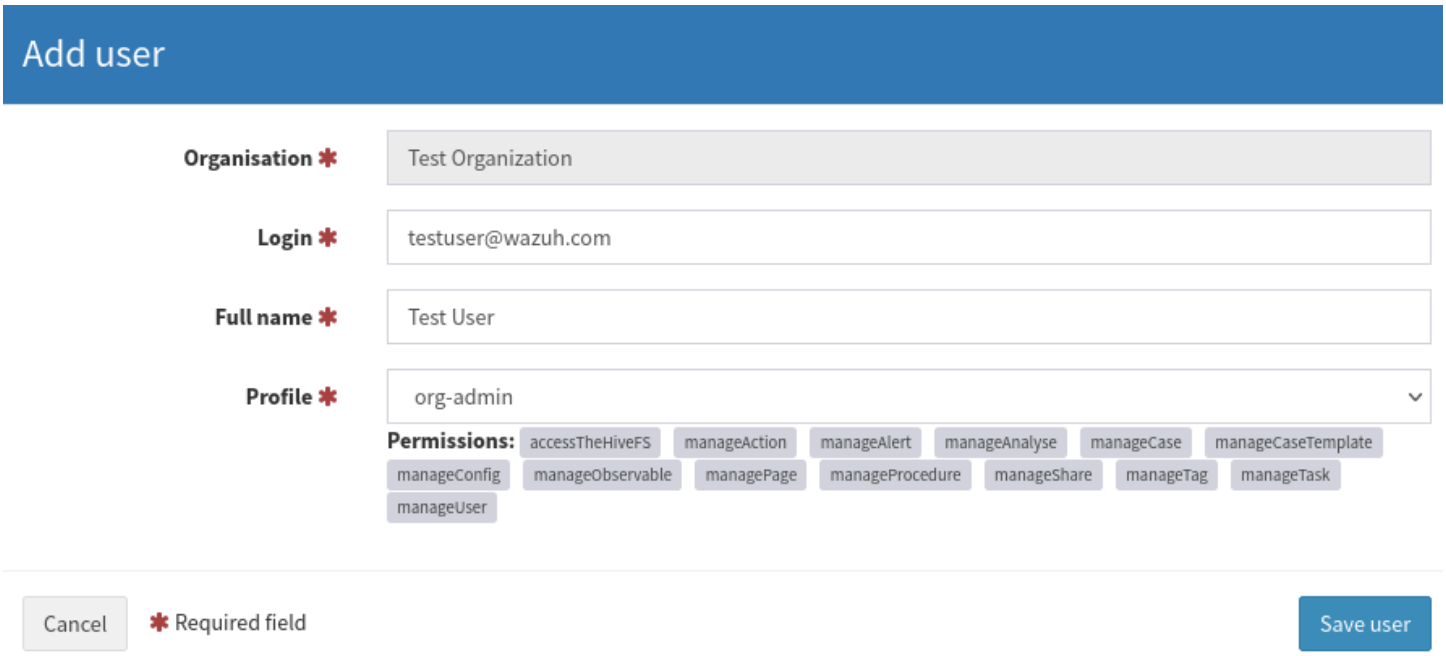
Wazuh and TheHive

Prepare TheHive:

1.**Create** a **new organization** on **TheHive web interface** and with an **administrator account.**



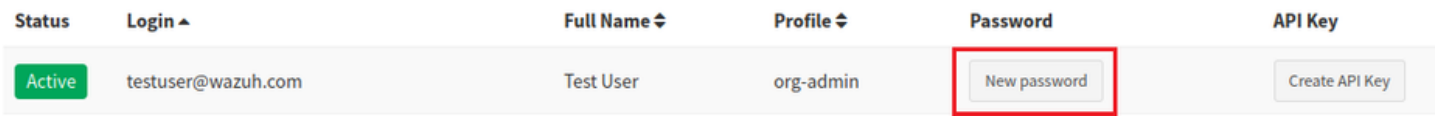
2.In **Test Organization**, we **create a new user** with organization administrator privileges.



3.This user has permissions to manage the organization, including **creating new users, managing cases, and alerts, amongst others.**

4.We also **create a password** for this user so that we can **log in** to view the dashboard and manage cases.

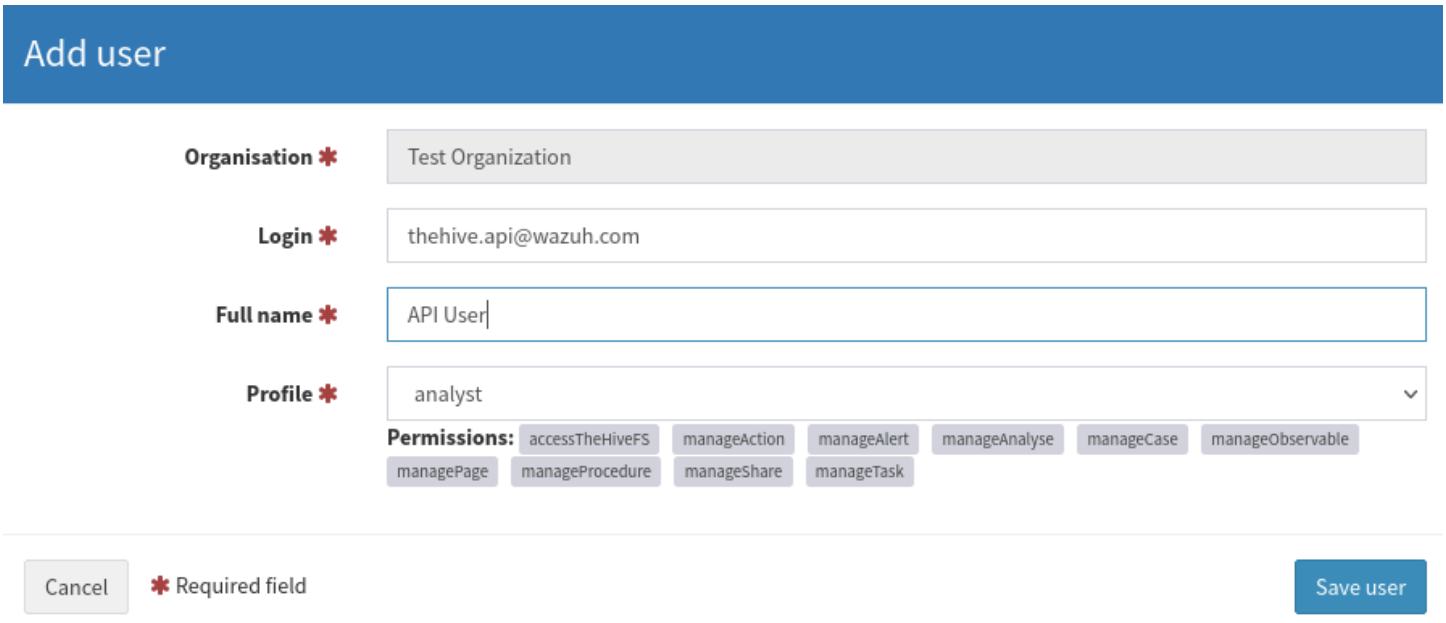
5.This is done by clicking on **“New password”** beside the user account and entering the desired password.



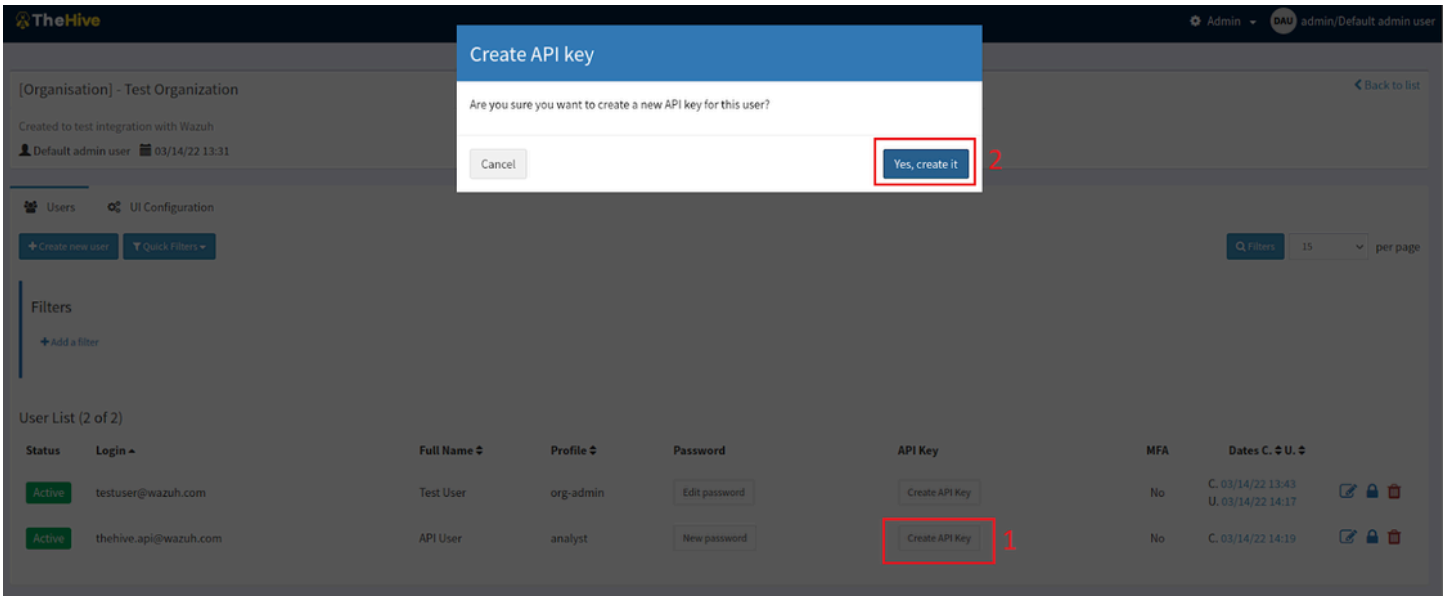
6.The **integration** with **Wazuh** is possible with the **aid of TheHive REST API.**

7.Therefore, we need a **user on TheHive** that can **create alerts via the API.**

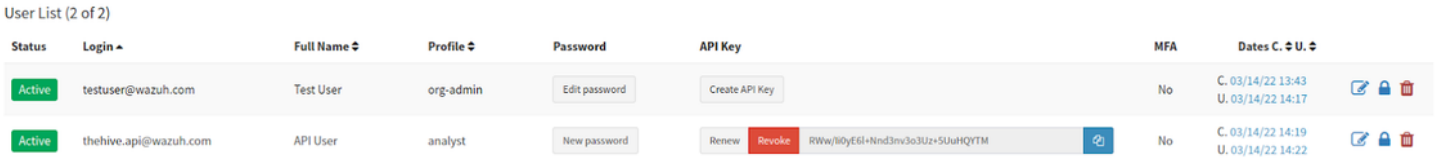
8.We **create** an account with an **“analyst”** privilege for this purpose.



9.we **generate the API key** for the **user:**



10.In order to **extract the API key**, we reveal the **key to view** and **copy it out** for future use:



Configure Wazuh manager:

1.**Install** TheHive Python module:

sudo /var/ossec/framework/python/bin/pip3 install thehive4py==1.8.1

2.**Create** the custom **integration script** by **pasting** the following python code in **/var/ossec/integrations/custom-w2thive.py.**

3.The **lvl_threshold** variable in the script indicates the **minimum alert level** that will be forwarded to TheHive.

4.The **variable** can be **customized** so that only **relevant alerts** are forwarded to TheHive:

5.**Custom-w2thive.py:**

- Create a **bash script** as **/var/ossec/integrations/custom-w2thive.** This will properly execute the .py script created in the previous step:

#!/bin/sh

 # Copyright (C) 2015-2020, Wazuh Inc.

 # Created by Wazuh, Inc. <info@wazuh.com>.

 # This program is free software; you can redistribute it and/or modify it under the terms of GP>

```
WPYTHON_BIN="framework/python/bin/python3"
SCRIPT_PATH_NAME="$0"
DIR_NAME="$(cd $(dirname ${SCRIPT_PATH_NAME}); pwd -P)"
SCRIPT_NAME="$(basename ${SCRIPT_PATH_NAME})"
case ${DIR_NAME} in
  */active-response/bin | */wodles*)
    if [ -z "${WAZUH_PATH}" ]; then
      WAZUH_PATH="$(cd ${DIR_NAME}/../..; pwd)"
    fi
  PYTHON_SCRIPT="${DIR_NAME}/${SCRIPT_NAME}.py"
  ;;
  */bin)
  if [ -z "${WAZUH_PATH}" ]; then
    WAZUH_PATH="$(cd ${DIR_NAME}/..; pwd)"
  fi
  PYTHON_SCRIPT="${WAZUH_PATH}/framework/scripts/${SCRIPT_NAME}.py"
  ;;
   */integrations)
    if [ -z "${WAZUH_PATH}" ]; then
      WAZUH_PATH="$(cd ${DIR_NAME}/..; pwd)"
    fi
  PYTHON_SCRIPT="${DIR_NAME}/${SCRIPT_NAME}.py"
  ;;
esac
${WAZUH_PATH}/${WPYTHON_BIN} ${PYTHON_SCRIPT} $@
```

6.**Change** the **files' permission** and the **ownership** to ensure that Wazuh has **adequate permissions** to **access and run** them:

```
sudo chmod 755 /var/ossec/integrations/custom-w2thive.py
sudo chmod 755 /var/ossec/integrations/custom-w2thive
```

sudo chown root:ossec /var/ossec/integrations/custom-w2thive.py

sudo chown root:ossec /var/ossec/integrations/custom-w2thive

NOTE:

- ☐ The correct ownership for **Wazuh 4.3.0** is **root:wazuh.**

7.**Allow Wazuh** to **run the integration script**, Add the following lines to the manager configuration file located at **/var/ossec/etc/ossec.conf.**

8.**Insert** the **IP address** for TheHive server along with the **API key** that was generated earlier:

<ossec_config>

...

<integration>

<name>custom-w2thive</name>

<hook_url>http://TheHive_Server_IP:9000</hook_url>

<api_key>RWw/Ii0yE6l+Nnd3nv3o3Uz+5UuHQYTM</api_key>

<alert_format>json</alert_format>

</integration>

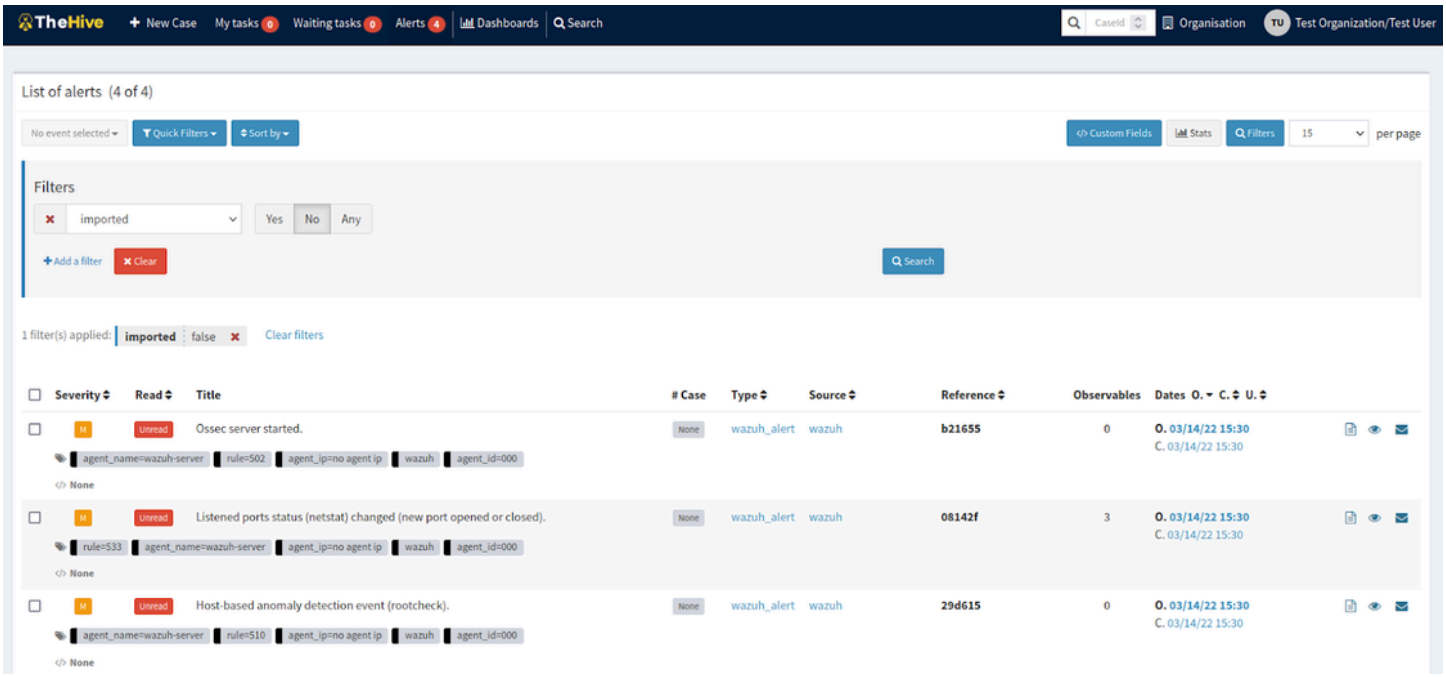
...

</ossec_config>

9.**Restart** the **manager** to apply the changes:

sudo systemctl restart wazuh-manager

10.**Log into TheHive** with our **test user account,** and we can see **Wazuh generated alerts** under the **“Alerts”** tab:



11.At this point, we can proceed to **perform other standard TheHive actions** on the alerts, such as **creating cases** on them or **adding them** to other **existing cases.**

Conclusion:

- Wazuh is a flexible security solution that integrates well with other solutions. It is open source and gives users the freedom to create and use custom integration scripts. This blog post shows that Wazuh integrates well with TheHive with the aid of custom scripts.

TheHive and Cortex

Prerequisites:

1. **TheHive**: Installed and running (v4 or later recommended).
2. **Cortex**: Installed and running.
3. **Connectivity**: Ensure TheHive can connect to Cortex (same network or proper firewall rules in place).

Configure Analyzers in Cortex:

1. Ensure Cortex is accessible from TheHive.
2. **Configure Analyzers** in Cortex:
    - Go to Cortex's web UI and enable the analyzers you need.
    - Configure the required settings for each analyzer, such as API keys for third-party services (e.g., VirusTotal, Shodan).
3. **Test Analyzers**:
    - Test each enabled analyzer in Cortex to confirm it works correctly.

Configure Cortex in TheHive:

1. **Generate API Key in Cortex**:
    - Go to Cortex's web UI → **API Keys** → Generate a new API key.
    - Copy the **API key**.

2. **Add Cortex to TheHive**:
    - Log in to TheHive as an admin.
    - Navigate to **Administration** → **Connectors** → **Cortex**.
    - Add a new Cortex instance:
        - **Name**: Provide a name for the Cortex instance.
        - **URL**: Enter the Cortex server URL (e.g., http://<cortex-ip>:<port>).
        - **API Key**: Paste the API key generated earlier.
        - **Active**: Enable the Cortex connector.
3. **Test the connection.**

Link Analyzers to TheHive:

1. **Map Cortex Analyzers in TheHive**:
    - Go to **Administration** → **Connectors** → **Cortex**.
    - For each enabled analyzer in Cortex, set permissions and link them to TheHive.
2. **Set Default Analyzers (Optional)**:
    - Define default analyzers for specific types of observables (e.g., URLs, IPs, hashes).

Test the Integration:

1. Create a **case** or **alert** in TheHive.
2. Add observables (e.g., file hash, URL, IP address).
3. Use the Cortex analyzer by:
    - Clicking **Analyze** for an observable.
    - Selecting one or more analyzers to run.
4. Verify that the results from Cortex analyzers appear in TheHive.

Automate with Responders (Optional):

1. Cortex responders can automate actions like blocking an IP, isolating a machine, or notifying teams.
2. Configure responders in Cortex, then link and trigger them from TheHive.

Troubleshooting:

1. Check logs for errors:
    - TheHive: /var/log/thehive/application.log
    - Cortex: /var/log/cortex/application.log
2. Verify network connectivity and API key validity.

TheHive and MISP

Prerequisites:

1. **TheHive:** Installed and running.
2. **MISP:** Installed and configured with threat intelligence data.
3. **Connectivity:** Ensure TheHive can access the MISP instance.

Configure MISP:

1. Generate an API Key in MISP:

- Log in to MISP.
- Go to **Administration** → **List Users** → **Select your user**.
- Copy the **Auth Key**.

2. Verify MISP Settings:

- Note the MISP URL (e.g., https://<misp-ip>) and API key.
- Ensure MISP has SSL certificates (or disable SSL verification in TheHive if not using HTTPS).

3. Add MISP Connector in TheHive:

- Log in to TheHive as an **administrator.**
- Navigate to **Administration** → **Connectors** → **MISP**.
- Add a **new MISP instance:**
  - **Name**: Choose a descriptive name (e.g., MISP Server).
  - **URL**: Enter the MISP server URL.
  - **API Key**: Paste the API key copied earlier.
  - **Trust SSL**: Enable this if MISP uses HTTPS with a valid certificate. Disable if it's self-signed.
  - **Active**: Enable the connector.
- **Save** and **Test** the connection.

Sync MISP Events with TheHive:

1. **Fetch Events:**
   - Configure TheHive to periodically pull events from MISP:
     - Go to **Administration** → **Connectors** → **MISP**.
     - Set up **synchronization settings**, such as:
       a. **Frequency** of pulling events.
       b. **Tags or filters** for specific events.
2. **View MISP Data in TheHive:**
   - Synced MISP events appear as cases or observables in TheHive.
   - TheHive can analyze these observables or enrich them with Cortex.

Use MISP Data in Incident Response:

1. Enrich Cases:

- Use **MISP** as an enrichment source for **observables** in **TheHive.**
- **TheHive queries MISP** for related **threat intelligence**.

2. Share Back to MISP (Optional):

- Configure **TheHive** to push **observables or cases** back to **MISP** for sharing with other organizations

Automate with Cortex (Optional):

1. **Link** Cortex analyzers to TheHive and use them alongside MISP data for detailed analysis.

Troubleshooting:

1. **Check logs** for errors:
   - TheHive: /var/log/thehive/application.log
   - MISP: /var/www/MISP/app/tmp/logs/error.log
2. **Verify** API key permissions in **MISP**.
3. **Test MISP connectivity** manually using curl or a browser.

Wazuh and OpenVAS

Prerequisites:

Before starting, ensure the following:

1. Tools Installed:

- OpenVAS (Greenbone Vulnerability Manager)
- Wazuh Manager and Agent
- Elasticsearch and Kibana (integrated with Wazuh)

2. Environment Setup:

- The **Wazuh agent** is installed on the **same system** as **OpenVAS** or has access to OpenVAS reports.
- Python is installed with **`xmltodict`** and **`requests`** libraries.

3. Network Configuration:

- Wazuh Manager and Kibana are accessible from the system running OpenVAS.

Export Reports from OpenVAS:

1. Log in to the **Greenbone Security Assistant (GSA).**

2. Navigate to **Scans > Reports.**

3. Export the desired report in **XML** format.

- Ensure reports are saved to a dedicated directory, e.g., **`/var/lib/openvas/reports`.**

Convert OpenVAS XML Reports to JSON:

- Wazuh requires JSON-formatted alerts for ingestion. Use a Python script to convert OpenVAS XML reports into JSON and upload them to Wazuh.

Install Required Python Libraries:

pip install xmltodict requests

Python Script for Conversion and Upload:

Save the following script as **`upload_openvas_to_wazuh.py`:**

```python

import os

import xmltodict

import json

import requests

 # Paths and URLs

openvas_reports_dir = "/var/lib/openvas/reports" **# Directory for OpenVAS reports**

wazuh_api_url = "http://<Wazuh_IP>:55000" **# Replace with Wazuh Manager IP**

wazuh_api_user = "wazuh" **# Replace with Wazuh username**

wazuh_api_password = "your_password" **# Replace with Wazuh password**

def convert_xml_to_json(xml_file):

  """Convert OpenVAS XML report to JSON format."""

  with open(xml_file, 'r') as file:

    xml_content = file.read()

  json_data = json.loads(json.dumps(xmltodict.parse(xml_content)))

  return json_data

def upload_to_wazuh(json_data):

  """Upload JSON alerts to Wazuh."""

  endpoint = f"{wazuh_api_url}/agents/<Agent_ID>/logs" # Replace <Agent_ID> with Wazuh agent ID

  headers = {"Content-Type": "application/json"}

```
    response = requests.post(endpoint, auth=(wazuh_api_user, wazuh_api_password), headers=headers, json=json_data)

    return response.status_code, response.text


def process_reports():

    """Process and upload all XML reports in the folder."""

    for file_name in os.listdir(openvas_reports_dir):

        if file_name.endswith(".xml"):

            xml_path = os.path.join(openvas_reports_dir, file_name)

            json_data = convert_xml_to_json(xml_path)

            status, response = upload_to_wazuh(json_data)

            print(f"Uploaded {file_name}: Status {status}, Response: {response}")


if __name__ == "__main__":

    process_reports()
```

Script Details

- Replace placeholders (`<Wazuh_IP>`, `<Agent_ID>`, etc.) with actual environment details.
- Ensure the directory `/var/lib/openvas/reports` matches your OpenVAS report location.

Configure Wazuh to Monitor OpenVAS Alerts:

1. **Edit** the Wazuh agent configuration:

sudo nano /var/ossec/etc/ossec.conf

2. Add a **`<localfile>`** entry to monitor the directory with JSON reports:

```
<localfile>
  <log_format>json</log_format>
  <location>/var/lib/openvas/reports/*.json</location>
</localfile>
```

3. Add a **'<decoder>'** in **sudo nano /var/ossec/etc/decoders/local_decoder.xml:**

```
<decoder name="openvas_json">
```

<program_name>openvas_json</program_name>

<type>json</type>

<regex>.</regex>*

<description>OpenVAS JSON Report Decoder</description>

<field name="vulnerability_id">.</field> <!-- Adjust field names based on your JSON structure -->*

<field name="severity">.</field> <!-- Adjust field names based on your JSON structure -->*

<field name="description">.</field> <!-- Adjust field names based on your JSON structure -->*

<group>openvas</group>

</decoder>

4. **Restart the Wazuh agent** to apply changes:

sudo systemctl restart wazuh-agent

5. **Verify Alerts** in Wazuh:

- Log in to the Wazuh Dashboard.
- Navigate to Alerts and confirm that OpenVAS alerts are being ingested.
- Check Wazuh logs if alerts are not appearing:

sudo tail -f /var/ossec/logs/ossec.log

Visualize OpenVAS Alerts in Kibana:

1. **Open** Kibana and navigate to the Wazuh app.

2. **Verify** that OpenVAS alerts are displayed.

3. To create **custom visualizations:**

    - Go to Dashboard > Create Dashboard.
    - Add visualizations for OpenVAS alerts, such as:
        - Vulnerability severity breakdown.
        - Timeline of detected vulnerabilities.
        - Assets/IPs with the highest vulnerabilities.

Automate the Workflow:

1. Schedule **periodic scans** in OpenVAS.

2. Automate the **XML-to-JSON conversion script** using a cron job:

crontab -e

3. Add the following entry to **run the script every hour:**

*0 * * * * python3 /path/to/upload_openvas_to_wazuh.py*

Troubleshooting:

1.Wazuh Alerts Not Appearing:

- Check if JSON reports are correctly formatted.
- Verify Wazuh agent configuration and logs.

2.Kibana Visualizations Missing Data:

- Ensure Elasticsearch is indexing Wazuh alerts properly.
- Check Kibana's index patterns and visualization settings.
