LAB4 SIEM TERUNGWA AKOR JACOB

Created	@February 27, 2025 3:42 PM
Attendance Required	☐

Part A

Task 1 - Introduction

a. **Brief Explanation of the Architecture of my SIEM Solution**

- **Architecture of Wazuh SIEM**
  Wazuh is an open-source security platform that provides threat detection, visibility, and compliance monitoring

The architecture of Wazuh consists of the following components:

1. **Wazuh Manager**: The central component that collects, analyzes, and stores data from agents. It also performs correlation and rule-based alerting and integrates with external tools like VirusTotal, YARA, and osquery.
2. **Wazuh Agents**: Lightweight software installed on endpoints (e.g., Windows, Linux, network devices) that collect and forward logs, file integrity data, and system inventory to the Wazuh Manager.
3. **Elastic Stack (Elasticsearch, Logstash, Kibana)**: Wazuh integrates with the Elastic Stack for data storage, visualization, and advanced analytics. Elasticsearch stores the data, Logstash processes it, and Kibana provides a user-friendly dashboard for visualization.
4. **External Integrations**: Wazuh can integrate with tools like VirusTotal for malware analysis, YARA for threat detection, and osquery for endpoint monitoring.

b. **Advantages of Open Source Solutions and How Vendors Make Money**

- **Cost-Effective:** No licensing fees, reducing total cost of ownership.
- **Transparency & Security:** Open-source code allows for auditing and verifying security features.
- **Community Support & Customizability:** Large developer communities provide updates, plugins, and integrations.

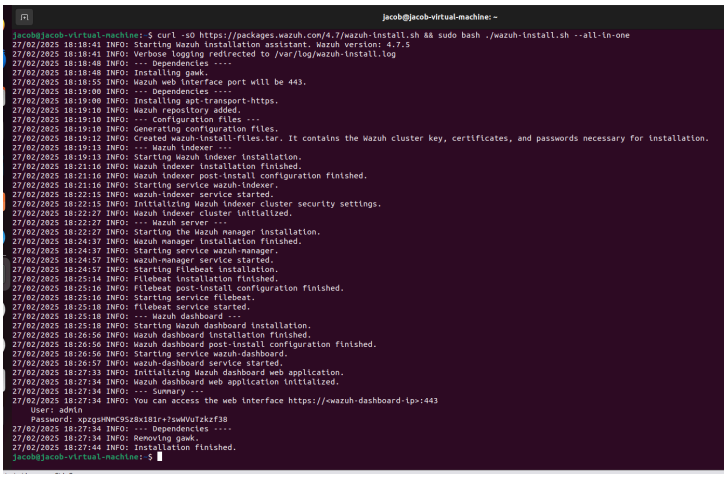
How Vendors Make Money:

- **Support and Consulting**: Vendors offer paid support, consulting, and training services.
- **Enterprise Features**: Some open-source solutions offer premium features or enterprise versions with additional capabilities.
- **Managed Services**: Vendors may offer managed SIEM services where they handle the deployment, monitoring, and maintenance for clients.

Task 2 - Setup infrastructure

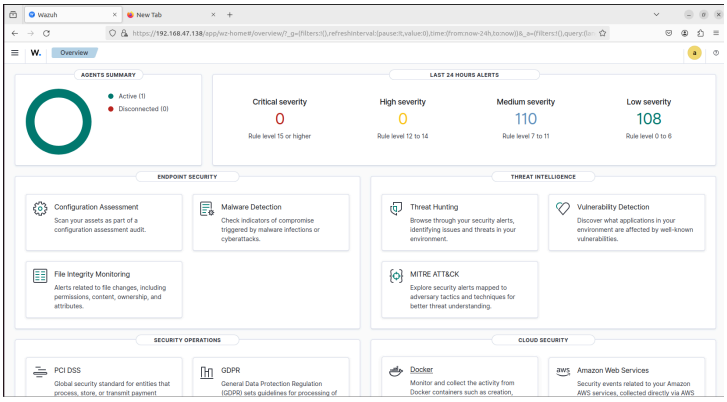
a. Configure a SIEM solution with 3(or more) unique devices. e.g Windows, Linux and a Network device. Can you view the log data from each connected device? If yes show this.

- For a Linux Machine, I installed the Wazuh Manager, Wazuh Indexer (Elasticsearch), and Wazuh Dashboard (Kibana).
  - curl -sO https://packages.wazuh.com/4.7/wazuh-install.sh && sudo bash ./wazuh-install.sh --all-in-one



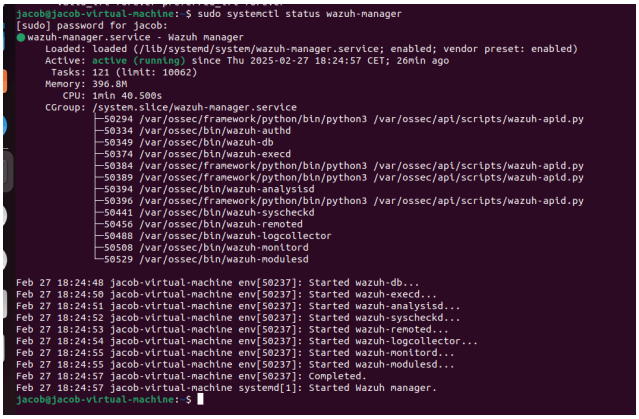
I accessed the Wazuh Dashboard by navigating to a browser using wazuh-dasboard ip address and the port number 443

- https://192.168.47.138:443



I verified if the Wazuh Manager is running by checking the status:

- sudo systemctl status wazuh-manager



- **Set Up the Wazuh Agent on a Windows Machine**

I downloaded and installed the Wazuh Agent by running the installer.

On the Wazuh Manager (Linux) - I generated the Agent Key:

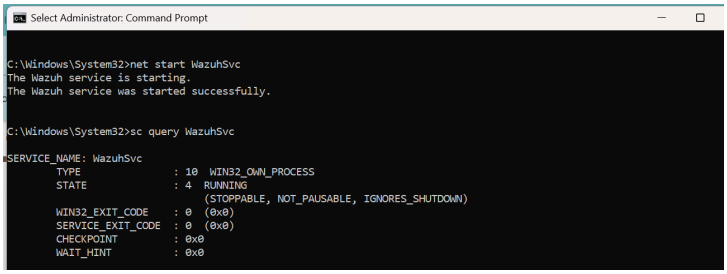
- sudo /var/ossec/bin/manage_agents

On the Wazuh Agent (Windows) - I registered the Agent with the Wazuh Manager using a command prompt as administrator.

- "C:\Program Files (x86)\ossec-agent\agent-auth.exe" -m 192.168.47.138 -p 1515 -A WinServer01 -k "MDAxIHdpbnNlcnZlcjAxIDE5Mi4xNjguNDcuMTM4IDk4N2YwZjAzZTBkNzNiZjFhY2QyYjdkM2I4ZTU5NzFjMzhkMWI3Nz

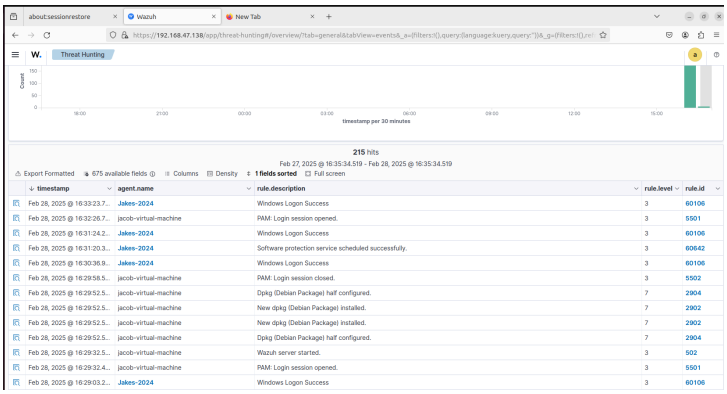
Then I restarted the agent service:

- net start WazuhSvc
- sc query WazuhSvc



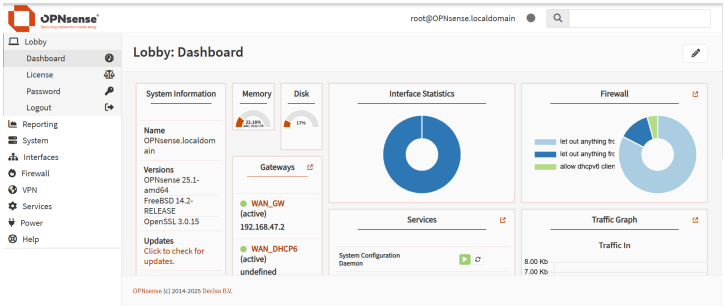
Verification of Log Collection:

- On the Wazuh Dashboard, I navigated to the **Agents** section and verified that the Windows agent is connected Then i checked for Windows Event Logs

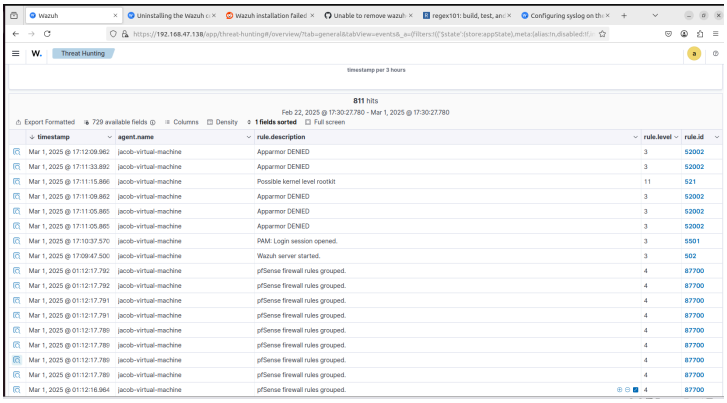


Configuration of Network Device

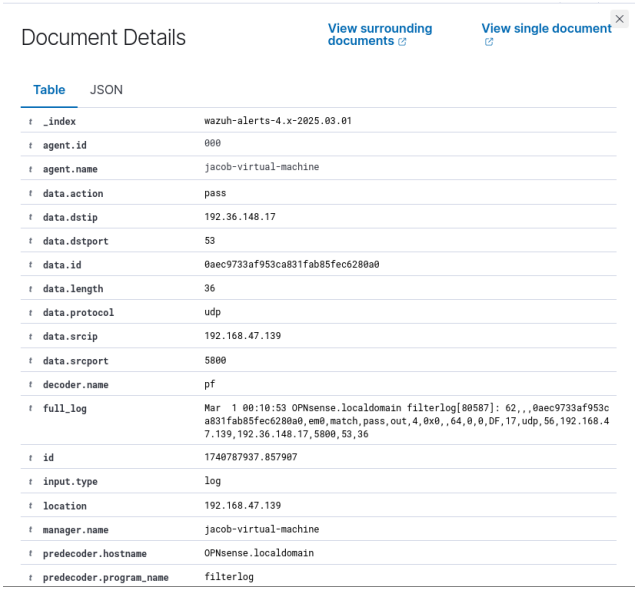
- I configured OPNsense to forward its logs to the Wazuh manager by using the Wazuh agent. Below is the OPNsense Dashboard:



- After the configuration, Logs from OPNsense appeared in the Wazuh dashboard under the **Events** section, as seen below:



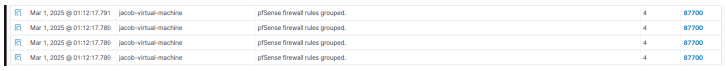
I was able to view one of the pfSense firewall logs:



2b. Why specifically are you able to view these logs i.e select two visible logs, explain these logs, and explain why and how you can view them on the SIEM.

Using 2 examples of the visible logs on my wazuh-dashboard

Example 1:



Explanation of pfSense Firewall Rules Grouped Logs

- **PfSense**: pfSense is an open-source firewall and router software. It logs events related to firewall rules, such as allowed or blocked traffic.
- **Firewall Rules Grouped Logs**: These logs represent events where traffic matches a specific firewall rule or group of rules. For example:
  - Traffic allowed by a rule.
  - Traffic is blocked by a rule.
  - Logs grouped by rule ID or rule description.

Why You Can View These Logs:

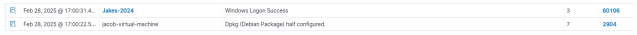
You can view the logs by the following reasons:

- **Centralized Log Collection**: A SIEM like Wazuh is designed to collect logs from multiple sources, including firewalls like pfSense. This centralized approach allows you to view and analyze logs from all connected devices in one place.
- **Real-Time Monitoring**: The SIEM continuously monitors logs in real time, enabling you to detect and respond to security events as they happen.
- **Normalization and Correlation**: The SIEM normalizes logs (converts them into a standard format) and correlates events across devices, making it easier to identify patterns and anomalies.

How You Can View These Logs

- **Syslog**: pfSense sends logs to the Wazuh Manager via Syslog.
- **Log Forwarding**: Logs are forwarded to the Wazuh Manager, where they are processed, analyzed, and stored.
- **Visualization**: The Wazuh Dashboard (Kibana) provides a user-friendly interface to view and analyze these logs.

Example 2: Dpkg (Debian Package) Half-Configured



- **What It Is**: A Dpkg half-configured log indicates that a Debian package installation or update was interrupted or failed, leaving the package in a partially configured state. This log helps identify issues with software installations, which could lead to system instability or security vulnerabilities.
- **How It's Collected**:
  - The Wazuh agent on the Linux machine monitors the system logs (e.g., `/var/log/dpkg.log`).

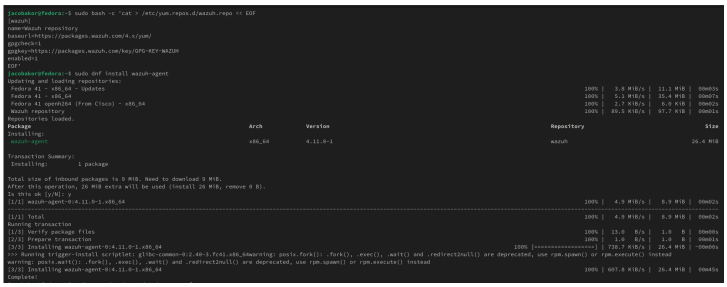
Configuration of Linux Device

On your Fedora device, i added the Wazuh repository and installed the wazuh-agent:

- sudo bash -c 'cat > /etc/yum.repos.d/wazuh.repo << EOF
  [wazuh]

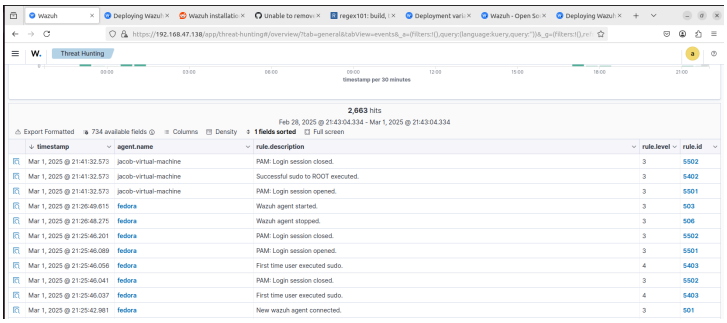
name=Wazuh repository
baseurl=https://packages.wazuh.com/4.x/yum/
gpgcheck=1
gpgkey=https://packages.wazuh.com/key/GPG-KEY-WAZUH
enabled=1
EOF'

- sudo dnf install wazuh-agent

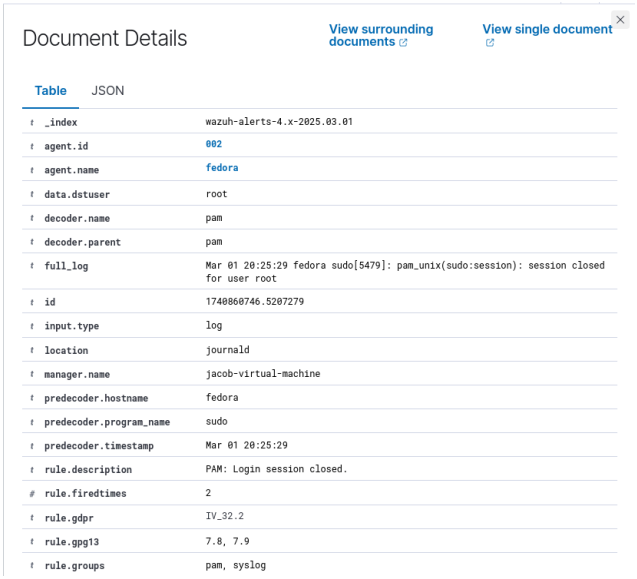


I configured the Wazuh Agent by **e**diting the Wazuh Agent configuration file (`/var/ossec/etc/ossec.conf`) to point to the Wazuh Manager.

After the configuration, Logs from the fedora were forwarded to the Wazuh dashboard and I viewed them under the **Events** section as seen below:



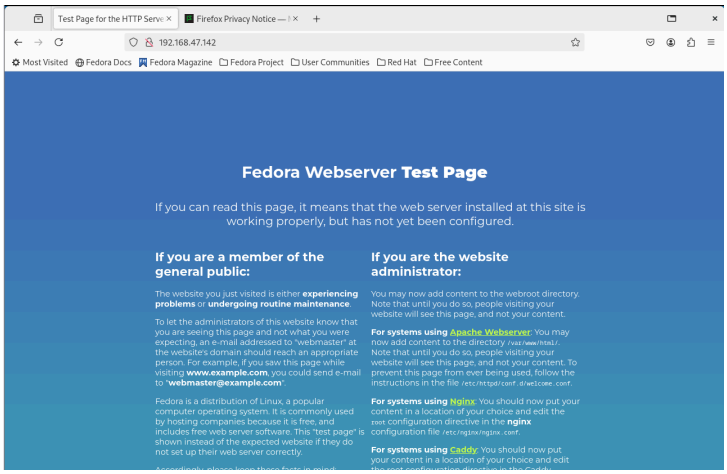
I view one of the logs from fedora as shown:



Task 3 - Use cases <select any 2>

a. Demonstrate how to block malicious IP addresses from accessing web resources on a web server. To do this , you will set up your web servers on select endpoints within your infrastructure and try to access them from an external endpoint.**LAB4 SIEM**

- I installed and configured an Apache web server on fedora using the command:
  - sudo dnf install httpd -y
  - sudo systemctl enable --now httpd
- In the browser, I viewed the Apache landing page and verified the installation: http://192.168.47.142



I added the following to /var/ossec/etc/ossec.conf file to configure the Wazuh agent and monitor the Apache access logs:

I restarted the Wazuh agent to apply the changes:

- sudo systemctl restart wazuh-agent

On the Wazuh server, I added the IP address of the RHEL endpoint to a CDB list and then configured rules and Active Response using the command:

I downloaded the Alienvault IP reputation database:

- sudo wget https://raw.githubusercontent.com/firehol/blocklist-ipsets/master/alienvault_reputation.ipset -O /var/ossec/etc/lists/alienvault_reputation.ipset

I appended the IP address of the attacker endpoint to the IP reputation database using the command below:

- sudo echo "192.168.47.139" >> /var/ossec/etc/lists/alienvault_reputation.ipset

Downloaded a script to convert from the .ipset format to the .cdb list format:

- sudo wget https://wazuh.com/resources/iplist-to-cdblist.py -O /tmp/iplist-to-cdblist.py

I converted the alienvault_reputation.ipset file to a .cdb format using the previously downloaded script:

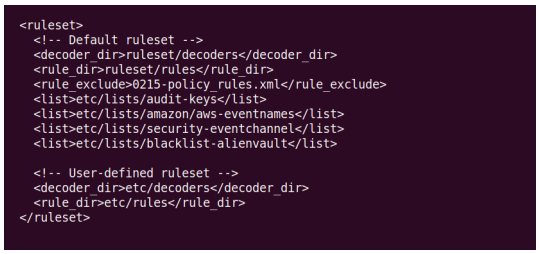
- sudo /var/ossec/framework/python/bin/python3 /tmp/iplist-to-cdblist.py /var/ossec/etc/lists/alienvault_reputation.ipset /var/ossec/etc/lists/blacklist-alienvault

I assigned the right permissions and ownership to the generated file:

- sudo chown wazuh:wazuh /var/ossec/etc/lists/blacklist-alienvault

Configured the Active Response module to block the malicious IP address and added a custom rule to trigger a Wazuh active response script in /var/ossec/etc/rules/local_rules.xml custom ruleset file:

Edited the Wazuh server /var/ossec/etc/ossec.conf configuration file and added the etc/lists/blacklist-alienvault list to the <ruleset> section:



Added the Active Response block to the Wazuh server /var/ossec/etc/ossec.conf file:

For the fedora enpoint: The firewall-drop command integrates with the fedora local iptables firewall and drops incoming network connection from the attacker endpoint for 60 seconds:

I restarted the Wazuh manager to apply the changes:

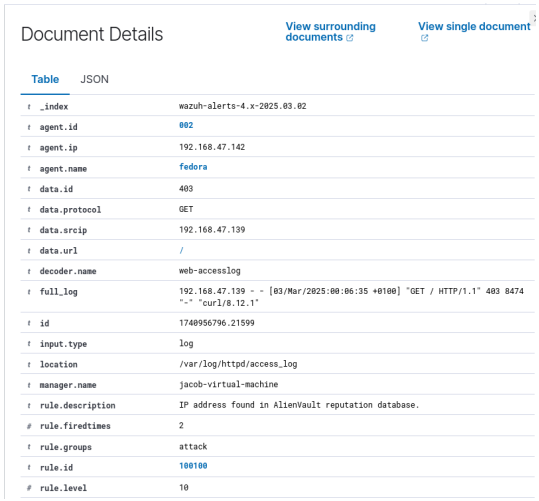
- sudo systemctl restart wazuh-manager

I accessed the web server from the RHEL endpoint using the IP address from the attacker endpoint:

- curl http://192.168.47.139

Here, the attacker endpoint connects to the victim's web servers for the first time. After the first connection, the Wazuh Active Response module temporarily blocks any successive connection to the web servers for 60 seconds.

I visualized the alert data in the Wazuh dashboard



b. Simulate a brute force attack against your infrastructure and demonstrate how you would detect the attack on each of the devices within your infrastructure. Are you able to detect the attack? If not, ensure you are able to.

I performed the following steps to configure the fedora endpoint. This allows performing authentication failure attempts on the monitored RHEL and Windows endpoints.

- On the attacker endpoint, I installed Hydra and used it to execute the brute-force attack:
  - sudo apt update
  - sudo apt install -y hydra

1. I created a text file with 10 random passwords

```
 GNU nano 6.2                                                pass.text
123
456
789
123
456
789
123
456
789
123
```

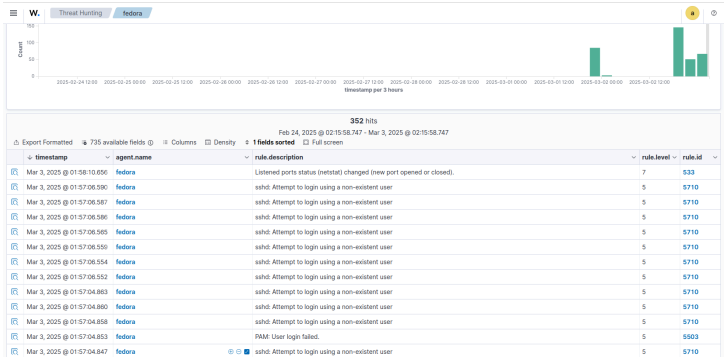
2. I run Hydra from the attacker endpoint to execute brute-force attacks against the 3 endpoints separately. example:

- sudo hydra -l badguy -P pass.text 192.168.47.139 ssh

```
^Cjacob@jacob-virtual-machine:~$ sudo hydra -l badguy -P pass.text 192.168.47.142 ssh
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-03-03 01:57:02
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 11 tasks per 1 server, overall 11 tasks, 11 login tries (l:1/p:11), ~1 try per task
[DATA] attacking ssh://192.168.47.142:22/
1 of 1 target completed, 0 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-03-03 01:57:07
```

I visualized the alerts:



Task 5 - SOC integrations <select any 1>

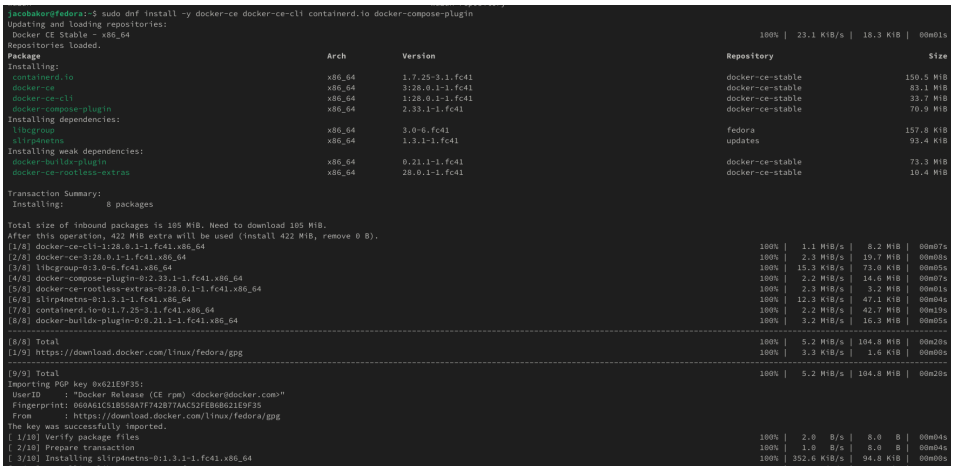
a. Integrate the SIEM with a case management system of your choice? e.g theHive.
Show that you are able to automatically open cases from SIEM alerts.

I created the Docker repository file and u**pdated the package lists, and verified that the repo exists**:

```
jacobakor@fedora:~$ sudo tee /etc/yum.repos.d/docker-ce.repo <<EOF
[docker-ce-stable]
name=Docker CE Stable - \$basearch
baseurl=https://download.docker.com/linux/fedora/\$releasever/\$basearch/stable
enabled=1
gpgcheck=1
gpgkey=https://download.docker.com/linux/fedora/gpg
EOF
[docker-ce-stable]
name=Docker CE Stable - $basearch
baseurl=https://download.docker.com/linux/fedora/$releasever/$basearch/stable
enabled=1
gpgcheck=1
gpgkey=https://download.docker.com/linux/fedora/gpg
jacobakor@fedora:~$ sudo dnf repolist
repo id                                                                repo name
docker-ce-stable                                                       Docker CE Stable - x86_64
fedora                                                                 Fedora 41 - x86_64
fedora-cisco-openh264                                                  Fedora 41 openh264 (From Cisco) - x86_64
updates                                                                Fedora 41 - x86_64 - Updates
wazuh                                                                  Wazuh repository
```

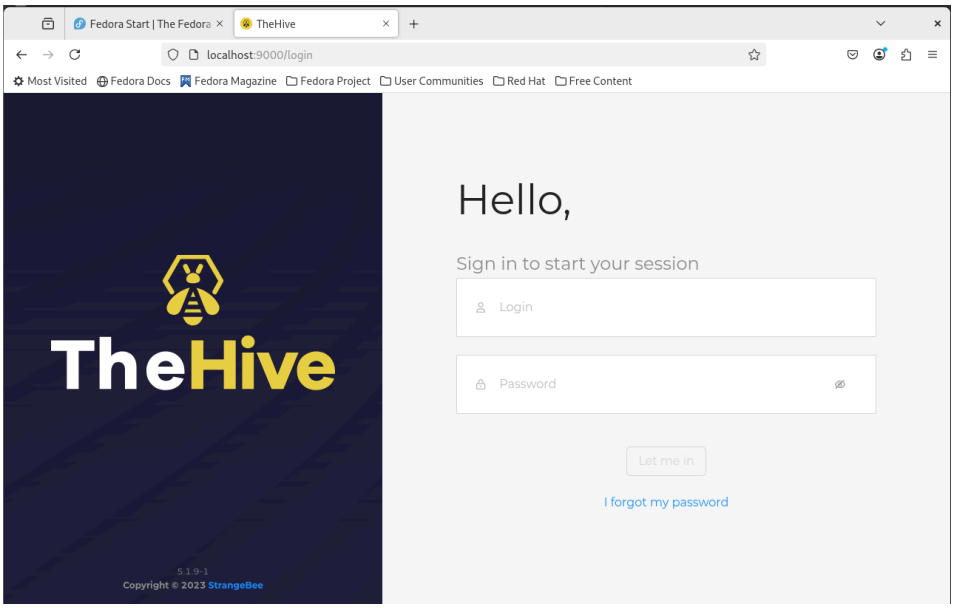
Then, I installed Docker

- sudo dnf install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

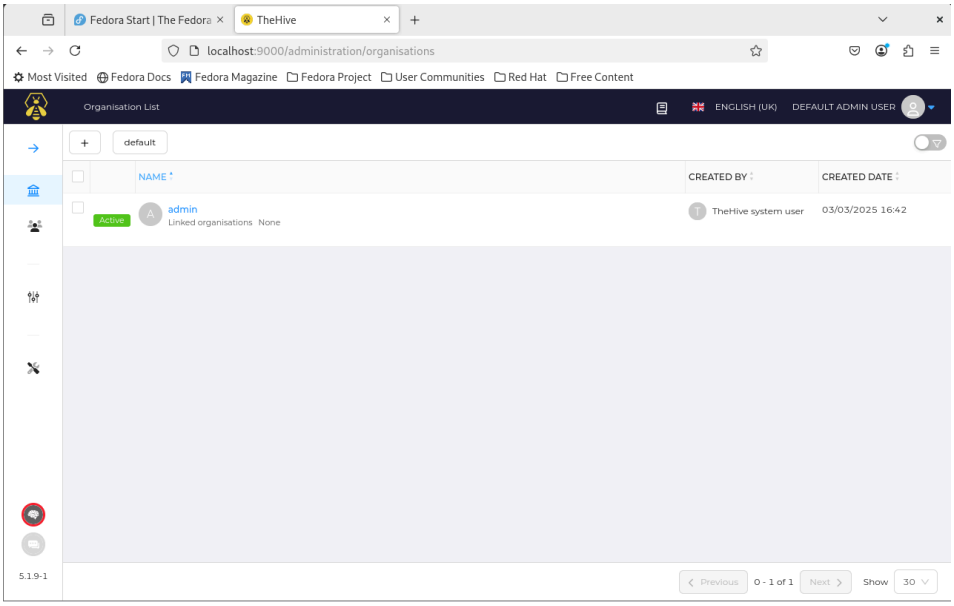


sudo docker compose up -d

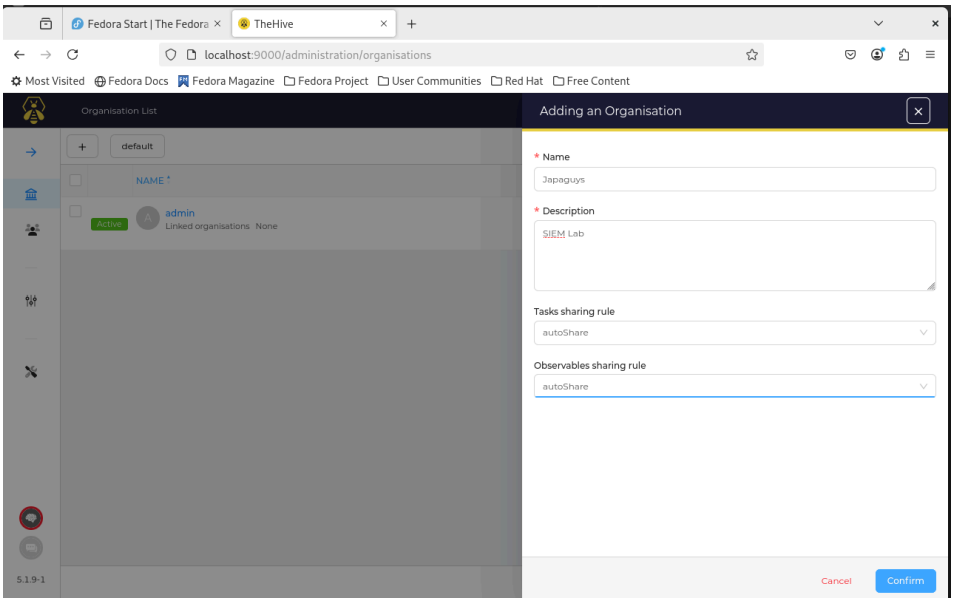
It was possible for me to access TheHive website



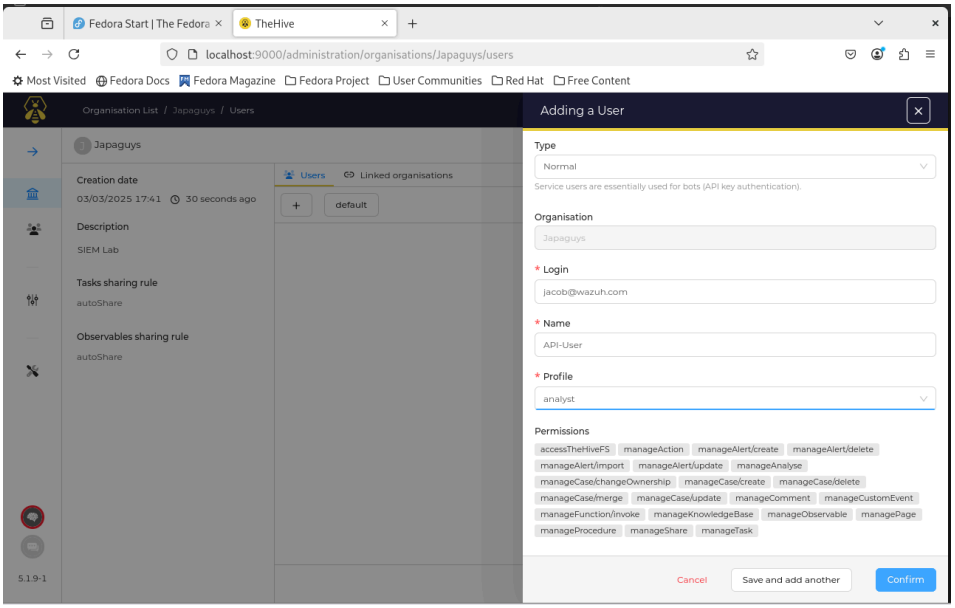
Then, I logged into the TheHive website the default credentials



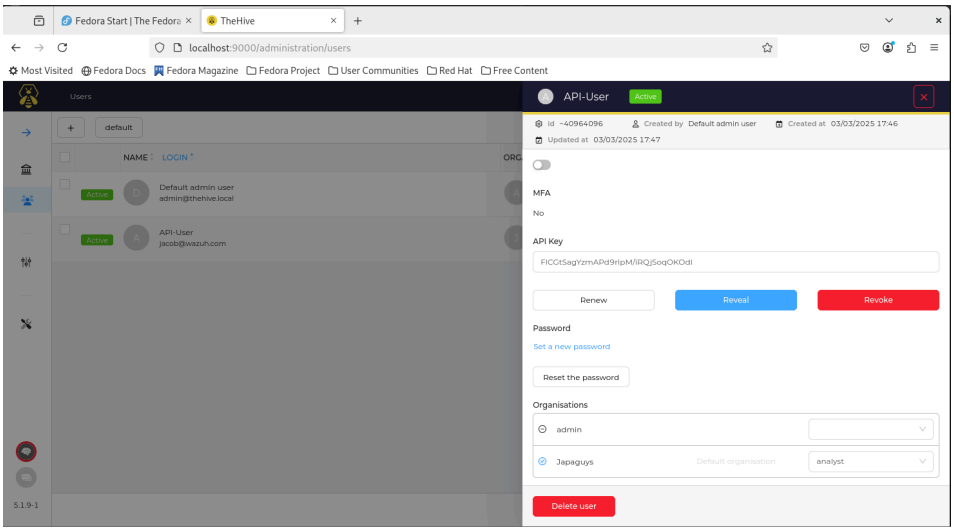
I proceeded by adding an organization:



I added a user:

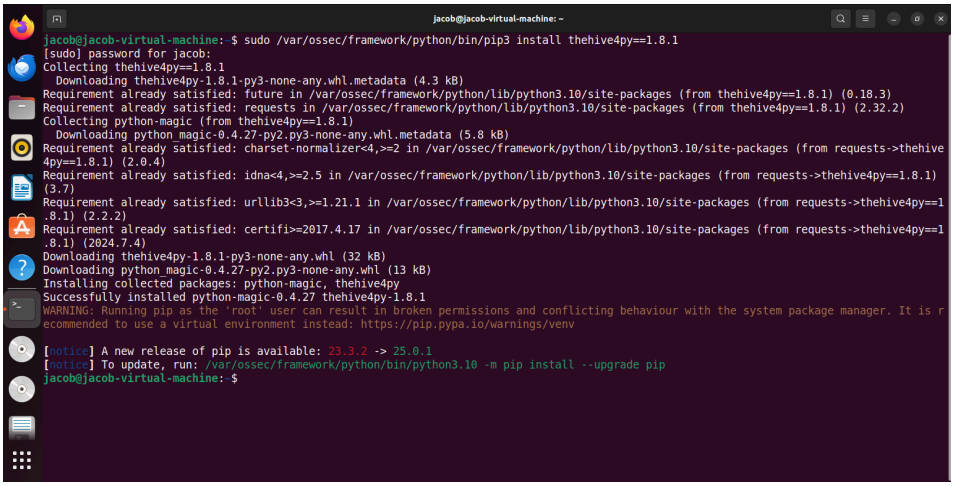


Created the API Key



I installed first, TheHive Python Module using the command:

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



Created the custom integration script by pasting the following python code in `/var/ossec/integrations/custom-w2thive.py`.
The `lvl_threshold` variable in the script indicates the minimum alert level that will be forwarded to TheHive. The variable can be customized so that only relevant alerts are forwarded to TheHive:

```
#!/var/ossec/framework/python/bin/python3
import json
import sys
import os
import re
import logging
import uuid
from thehive4py.api import TheHiveApi
from thehive4py.models import Alert, AlertArtifact
```

```
#start user config

# Global vars

#threshold for wazuh rules level
lvl_threshold=0
#threshold for suricata rules level
suricata_lvl_threshold=3

debug_enabled = False
#info about created alert
info_enabled = True

#end user config

# Set paths
pwd = os.path.dirname(os.path.dirname(os.path.realpath(__file__)))
log_file = '{0}/logs/integrations.log'.format(pwd)
logger = logging.getLogger(__name__)
#set logging level
logger.setLevel(logging.WARNING)
if info_enabled:
    logger.setLevel(logging.INFO)
if debug_enabled:
    logger.setLevel(logging.DEBUG)
# create the logging file handler
fh = logging.FileHandler(log_file)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
fh.setFormatter(formatter)
logger.addHandler(fh)



def main(args):
    logger.debug('#start main')
    logger.debug('#get alert file location')
    alert_file_location = args[1]
    logger.debug('#get TheHive url')
    thive = args[3]
    logger.debug('#get TheHive api key')
    thive_api_key = args[2]
    thive_api = TheHiveApi(thive, thive_api_key )
    logger.debug('#open alert file')
    w_alert = json.load(open(alert_file_location))
    logger.debug('#alert data')
    logger.debug(str(w_alert))
    logger.debug('#gen json to dot-key-text')
    alt = pr(w_alert,'',[])
    logger.debug('#formatting description')
    format_alt = md_format(alt)
    logger.debug('#search artifacts')
    artifacts_dict = artifact_detect(format_alt)
    alert = generate_alert(format_alt, artifacts_dict, w_alert)
    logger.debug('#threshold filtering')
```

```
    if w_alert['rule']['groups']==['ids','suricata']:
        #checking the existence of the data.alert.severity field
        if 'data' in w_alert.keys():
            if 'alert' in w_alert['data']:
                #checking the level of the source event
                if int(w_alert['data']['alert']['severity'])<=suricata_lvl_threshold:
                    send_alert(alert, thive_api)
    elif int(w_alert['rule']['level'])>=lvl_threshold:
        #if the event is different from suricata AND suricata-event-type: alert check lvl_threshold
        send_alert(alert, thive_api)


def pr(data,prefix, alt):
    for key,value in data.items():
        if hasattr(value,'keys'):
            pr(value,prefix+'.'+str(key),alt=alt)
        else:
            alt.append((prefix+'.'+str(key)+'|||'+str(value)))
    return alt



def md_format(alt,format_alt=''):
    md_title_dict = {}
    #sorted with first key
    for now in alt:
        now = now[1:]
        #fix first key last symbol
        dot = now.split('|||')[0].find('.')
        if dot==-1:
            md_title_dict[now.split('|||')[0]] =[now]
        else:
            if now[0:dot] in md_title_dict.keys():
                (md_title_dict[now[0:dot]]).append(now)
            else:
                md_title_dict[now[0:dot]]=[now]
    for now in md_title_dict.keys():
        format_alt+='### '+now.capitalize()+'\n'+'| key | val |\n| ------ | ------ |\n'
        for let in md_title_dict[now]:
            key,val = let.split('|||')[0],let.split('|||')[1]
            format_alt+='| **' + key + '** | ' + val + ' |\n'
    return format_alt


def artifact_detect(format_alt):
    artifacts_dict = {}
    artifacts_dict['ip'] = re.findall(r'\d+\.\d+\.\d+\.\d+',format_alt)
    artifacts_dict['url'] =  re.findall(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+',format
_alt)
    artifacts_dict['domain'] = []
    for now in artifacts_dict['url']: artifacts_dict['domain'].append(now.split('//')[1].split('/')[0])
    return artifacts_dict
```

```
def generate_alert(format_alt, artifacts_dict,w_alert):
    #generate alert sourceRef
    sourceRef = str(uuid.uuid4())[0:6]
    artifacts = []
    if 'agent' in w_alert.keys():
        if 'ip' not in w_alert['agent'].keys():
            w_alert['agent']['ip']='no agent ip'
    else:
        w_alert['agent'] = {'id':'no agent id', 'name':'no agent name'}

    for key,value in artifacts_dict.items():
        for val in value:
            artifacts.append(AlertArtifact(dataType=key, data=val))
    alert = Alert(title=w_alert['rule']['description'],
            tlp=2,
            tags=['wazuh',
            'rule='+w_alert['rule']['id'],
            'agent_name='+w_alert['agent']['name'],
            'agent_id='+w_alert['agent']['id'],
            'agent_ip='+w_alert['agent']['ip'],],
            description=format_alt ,
            type='wazuh_alert',
            source='wazuh',
            sourceRef=sourceRef,
            artifacts=artifacts,)
    return alert




def send_alert(alert, thive_api):
    response = thive_api.create_alert(alert)
    if response.status_code == 201:
        logger.info('Create TheHive alert: '+ str(response.json()['id']))
    else:
        logger.error('Error create TheHive alert: {}/{}'.format(response.status_code, response.text))



if __name__ == "__main__":

    try:
      logger.debug('debug mode') # if debug enabled
      # Main function
      main(sys.argv)

    except Exception:
      logger.exception('EGOR')
```

Created 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} $@
```

To allow Wazuh to run the integration script, I added the following lines to the manager configuration file at `/var/ossec/etc/ossec.conf`. Inserted the IP address for the TheHive server along with the API key that was generated earlier:

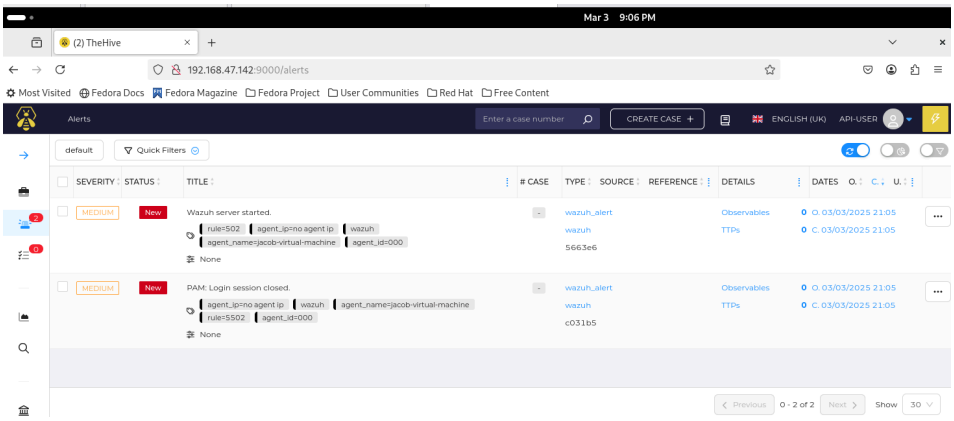
```
<ossec_config>

  <integration>
    <name>custom-w2thive</name>
    <hook_url>http://192.168.47.142:9000</hook_url>
    <api_key>FlCGtSagYzmAPd9rlpM/iRQjSoqOKOdI</api_key>
    <alert_format>json</alert_format>
  </integration>
</ossec_config>
```

Then, restart the manager to apply the changes:

- `sudo systemctl restart wazuh-manager`

I logged into TheHive with my test user account, and i can see Wazuh generated alerts under the “Alerts” tab



References:

https://wazuh.com/blog/using-wazuh-and-thehive-for-threat-protection-and-incident-response/

https://documentation.wazuh.com/current/index.html

https://documentation.wazuh.com/current/proof-of-concept-guide/detect-brute-force-attack.html