Wazuh SIEM Integration with MISP Threat Intelligence & Cowrie Honeypot

Security Operations Center (SOC) Lab Project

Prepared By:

Ishtiaq Rashid

ishtiaqrashid299@gmail.com

https://www.linkedin.com/in/shtiaq-rashid/

https://github.com/iamishtiaq

Cybersecurity | SOC Operations | Threat Intelligence & Honeypot Analysis

MISP & Honeypot Integration with Wazuh

Objective

The objective of this task is to integrate **MISP (Malware Information Sharing Platform)** and a **Honeypot** with **Wazuh** SIEM to enhance threat intelligence ingestion, detect real-world attacks, and improve SOC visibility. This task demonstrates how external threat intelligence and deception technologies are used together in enterprise SOC environments to strengthen detection and response capabilities.

Architecture Overview

The purpose of this architecture is to demonstrate how threat intelligence **(MISP)** and deception technology **(Honeypot)** can be integrated with **Wazuh** SIEM to detect, correlate, and analyze real-world cyber attacks in a SOC environment.

This setup simulates an enterprise **SOC workflow**, where external intelligence and live attack data are combined to improve detection accuracy and incident response.

Components Used

Wazuh SIEM

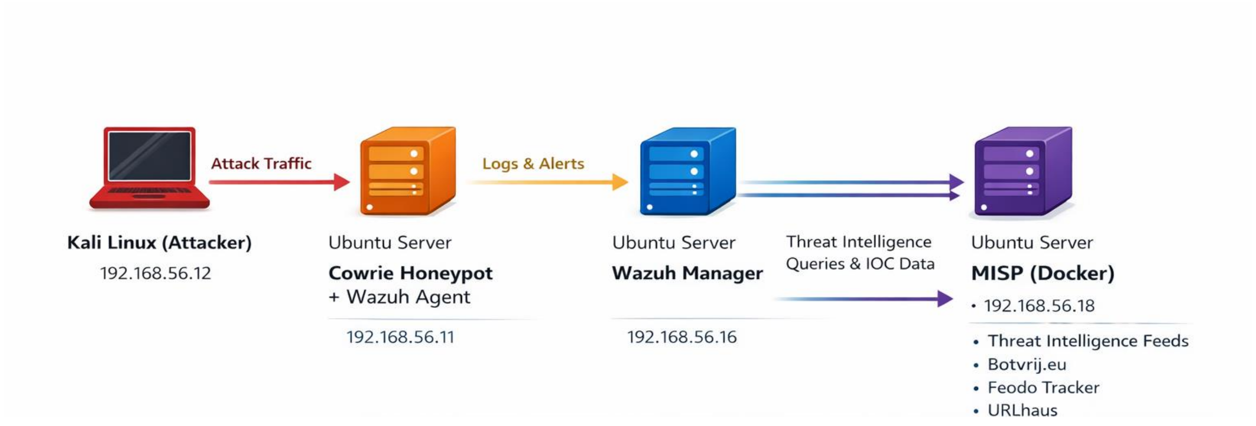
- Acts as the central log collection, correlation, and alerting platform.
- Receives logs from the honeypot and threat indicators from MISP.
- Displays alerts and analysis in the Wazuh Dashboard.

MISP (Malware Information Sharing Platform)

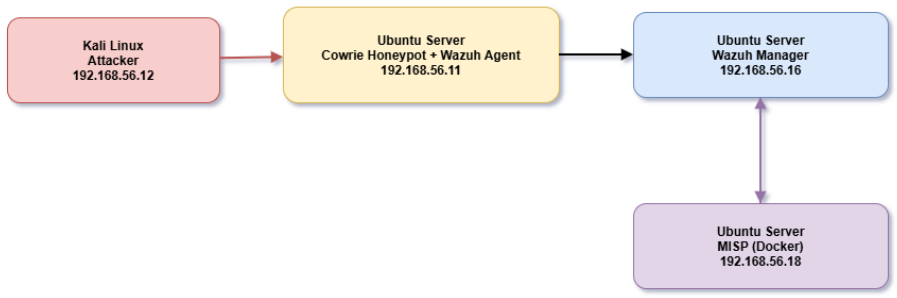
- Provides external threat intelligence such as malicious IP addresses, domains, and hashes.
- Supplies Indicators of Compromise (IOCs) to Wazuh for correlation.

Honeypot (Cowrie SSH Honeypot)

- Simulates a vulnerable SSH service to attract attackers.
- Captures attacker behavior such as brute-force attempts and login activity.



This diagram illustrates the complete Security Operations Center (SOC) architecture implemented during the internship. A Kali Linux machine initiates simulated attacks such as SSH brute-force attempts against the Cowrie honeypot deployed on an Ubuntu server. Cowrie captures attacker interactions and forwards the generated logs to the Wazuh Agent installed on the same system. The Wazuh Agent securely transmits these logs to the Wazuh Manager, where they are analyzed, decoded, and evaluated against custom detection rules. The Wazuh Manager integrates with MISP (running in Docker) to perform threat intelligence lookups, enabling IOC matching and alert enrichment using feeds such as Botvrij.eu, Feodo Tracker, and URLhaus. This architecture demonstrates end-to-end attack detection, log analysis, and threat intelligence correlation in a SOC environment.



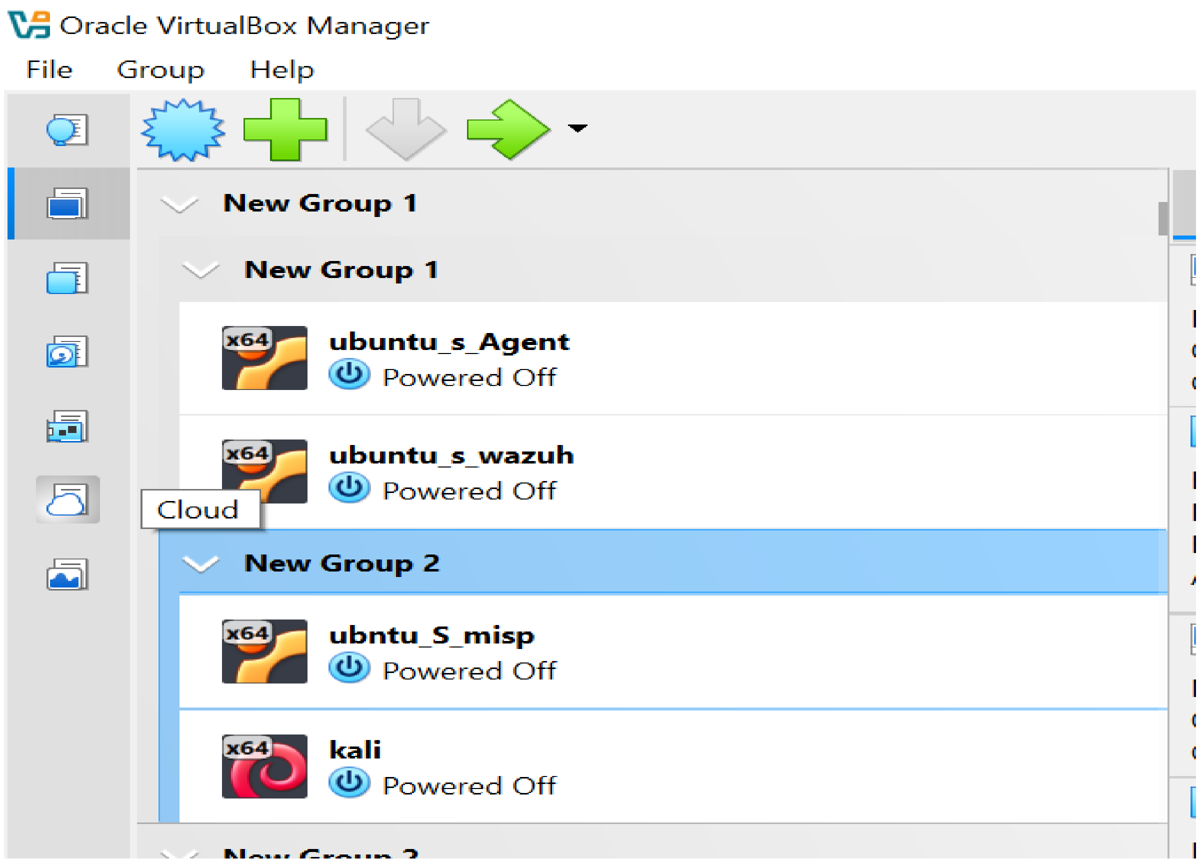
This simplified diagram presents a high-level logical view of the SOC workflow. Attack traffic originates from the Kali Linux attacker machine and targets the Cowrie honeypot. The Cowrie honeypot records all malicious interactions and forwards logs via the Wazuh Agent to the Wazuh Manager for centralized analysis. The Wazuh Manager communicates bidirectionally with the MISP server to enrich alerts with external threat intelligence and IOC data. This logical representation highlights how deception technologies and threat intelligence platforms enhance detection and decision-making in modern SOC operations.

Environment Setup

To perform this task, a virtualized lab environment was created using **Oracle VirtualBox**. Multiple virtual machines were deployed to simulate a real SOC setup with separate systems for attack simulation, honeypot deployment, SIEM management, and threat intelligence.

The following virtual machines were used:

- **Kali Linux**
  Used as the attacker machine to simulate real-world attacks against the honeypot.
- **Ubuntu Server – Cowrie Honeypot + Wazuh Agent**
  Hosts the Cowrie SSH honeypot and forwards attack logs to Wazuh
- **Ubuntu Server – Wazuh Manager**
  Acts as the central SIEM, receiving logs, correlating events, and generating alerts.
- **Ubuntu Server – MISP (Docker-based)**
  Hosts MISP using Docker to provide threat intelligence indicators.



ubuntu_s_Agent
Powered Off

ubuntu_s_wazuh
Powered Off

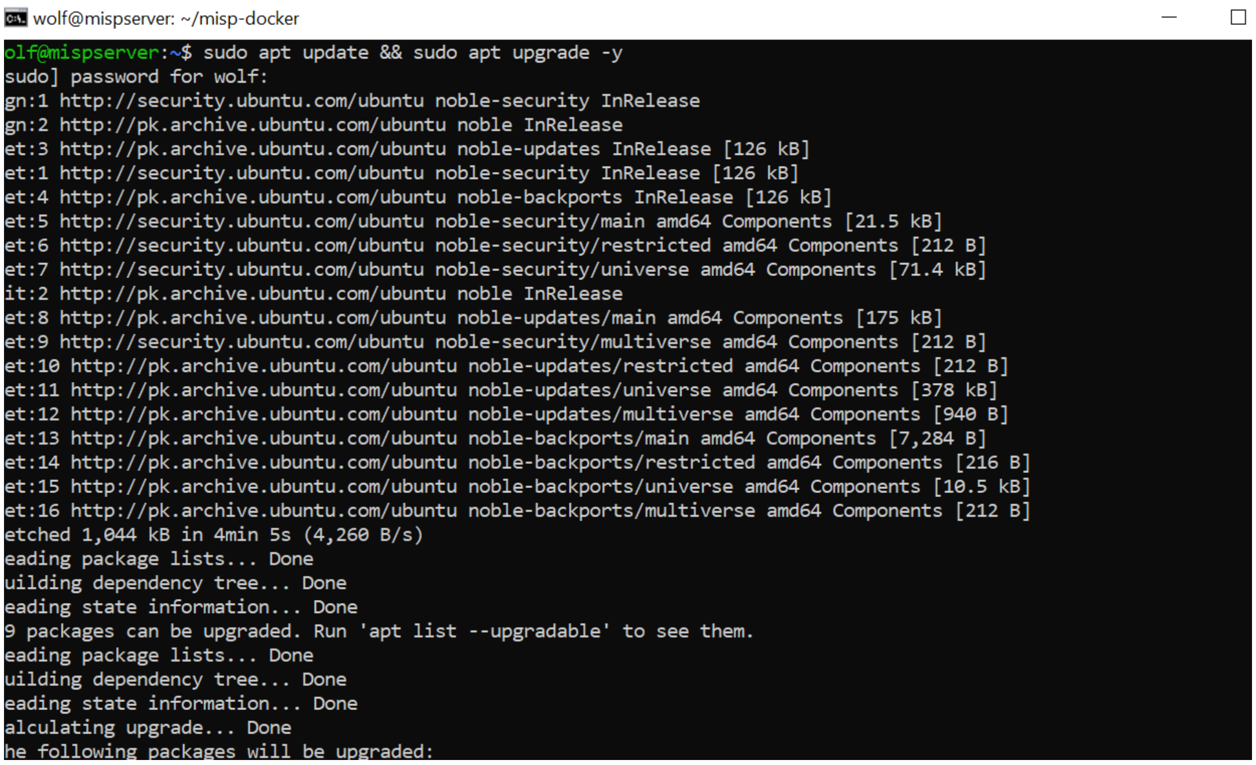
ubntu_S_misp
Powered Off

kali
Powered Off

This screenshot shows all virtual machines running simultaneously in Oracle VirtualBox. It confirms the separation of roles between the attacker system, honeypot, SIEM manager, and threat intelligence platform, reflecting a real-world SOC lab environment.

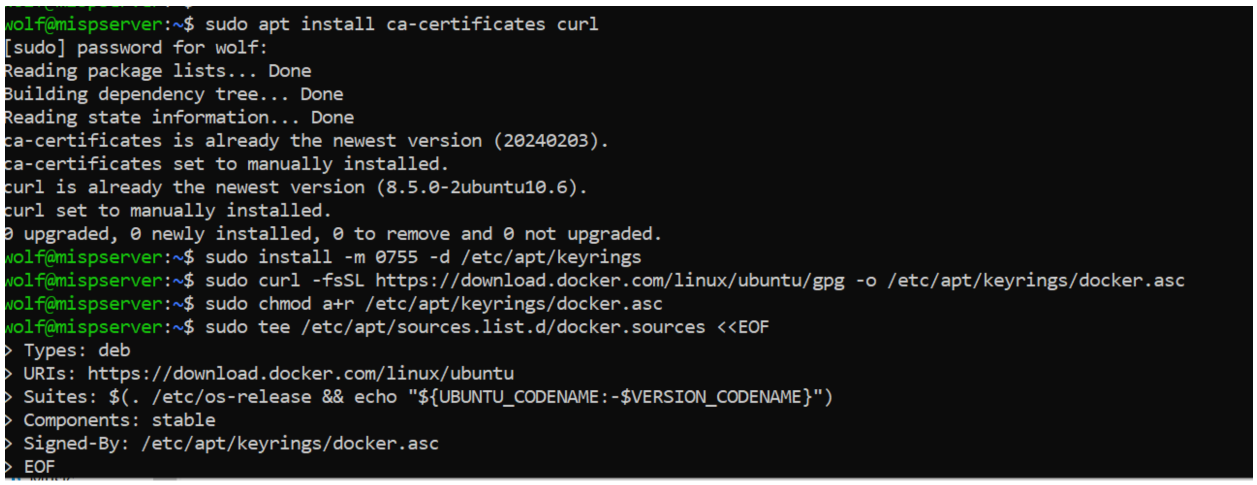
MISP Setup on docker in ubuntu

The MISP (Malware Information Sharing Platform) instance was deployed on a dedicated Ubuntu server using Docker containers. Docker was chosen to ensure a stable, isolated, and easily maintainable environment for running MISP, avoiding dependency conflicts and simplifying future updates.



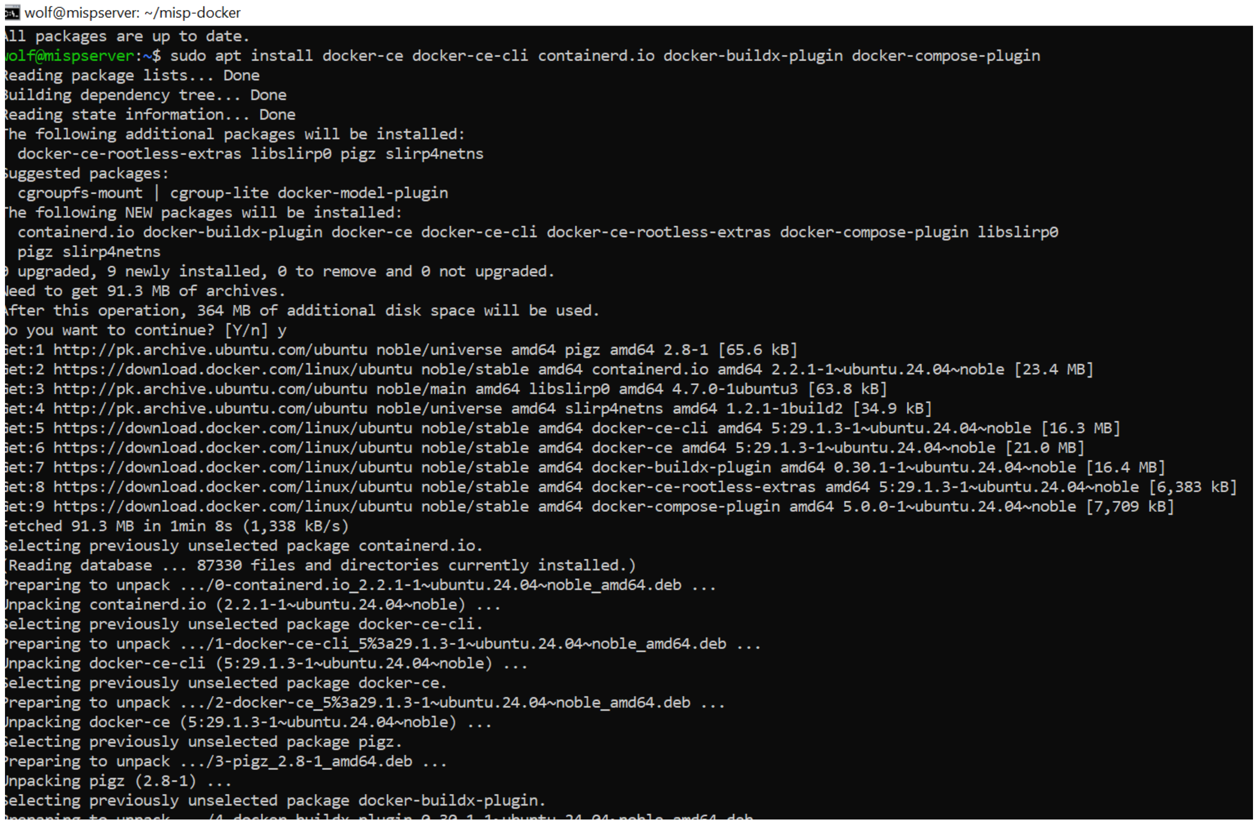
```
olf@mispserver:~$ sudo apt update && sudo apt upgrade -y
sudo] password for wolf:
gn:1 http://security.ubuntu.com/ubuntu noble-security InRelease
gn:2 http://pk.archive.ubuntu.com/ubuntu noble InRelease
et:3 http://pk.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
et:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
et:4 http://pk.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
et:5 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.5 kB]
et:6 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
et:7 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [71.4 kB]
it:2 http://pk.archive.ubuntu.com/ubuntu noble InRelease
et:8 http://pk.archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [175 kB]
et:9 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [212 B]
et:10 http://pk.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Components [212 B]
et:11 http://pk.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [378 kB]
et:12 http://pk.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Components [940 B]
et:13 http://pk.archive.ubuntu.com/ubuntu noble-backports/main amd64 Components [7,284 B]
et:14 http://pk.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [216 B]
et:15 http://pk.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [10.5 kB]
et:16 http://pk.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
etched 1,044 kB in 4min 5s (4,260 B/s)
eading package lists... Done
uilding dependency tree... Done
eading state information... Done
9 packages can be upgraded. Run 'apt list --upgradable' to see them.
eading package lists... Done
uilding dependency tree... Done
eading state information... Done
alculating upgrade... Done
he following packages will be upgraded:
```

This screenshot shows the Ubuntu system being updated and upgraded using package management commands.



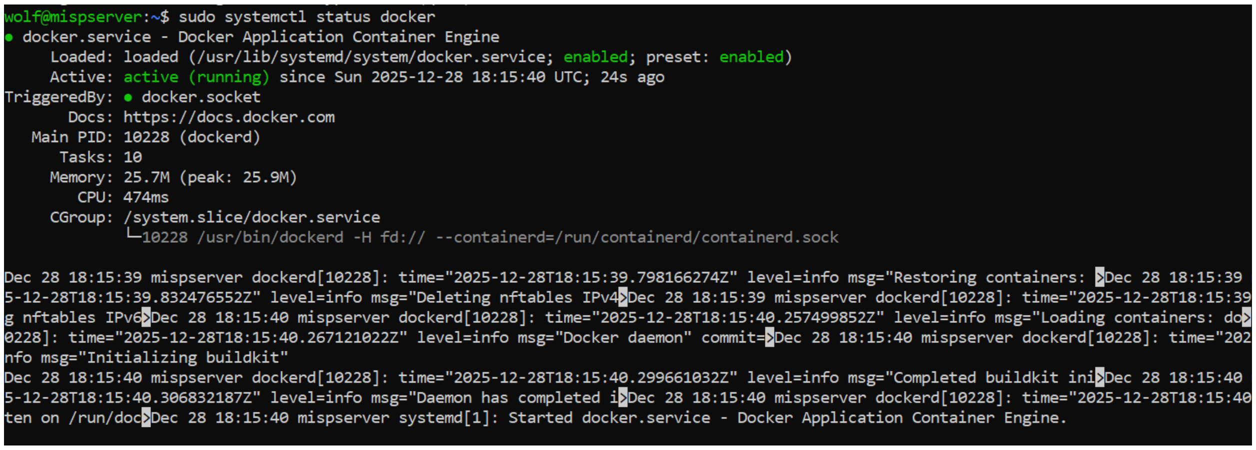
```
wolf@mispserver:~$ sudo apt install ca-certificates curl
[sudo] password for wolf:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20240203).
ca-certificates set to manually installed.
curl is already the newest version (8.5.0-2ubuntu10.6).
curl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
wolf@mispserver:~$ sudo install -m 0755 -d /etc/apt/keyrings
wolf@mispserver:~$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
wolf@mispserver:~$ sudo chmod a+r /etc/apt/keyrings/docker.asc
wolf@mispserver:~$ sudo tee /etc/apt/sources.list.d/docker.sources <<EOF
> Types: deb
> URIs: https://download.docker.com/linux/ubuntu
> Suites: $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}")
> Components: stable
> Signed-By: /etc/apt/keyrings/docker.asc
> EOF
```

This screenshot shows the execution of commands to install required dependencies and add Docker’s official GPG key and repository to the Ubuntu system. This step enables secure installation of Docker from the official source, which is required for deploying MISP using Docker containers. Proper repository configuration ensures package authenticity, system stability, and reliable container-based deployment in a SOC lab environment.



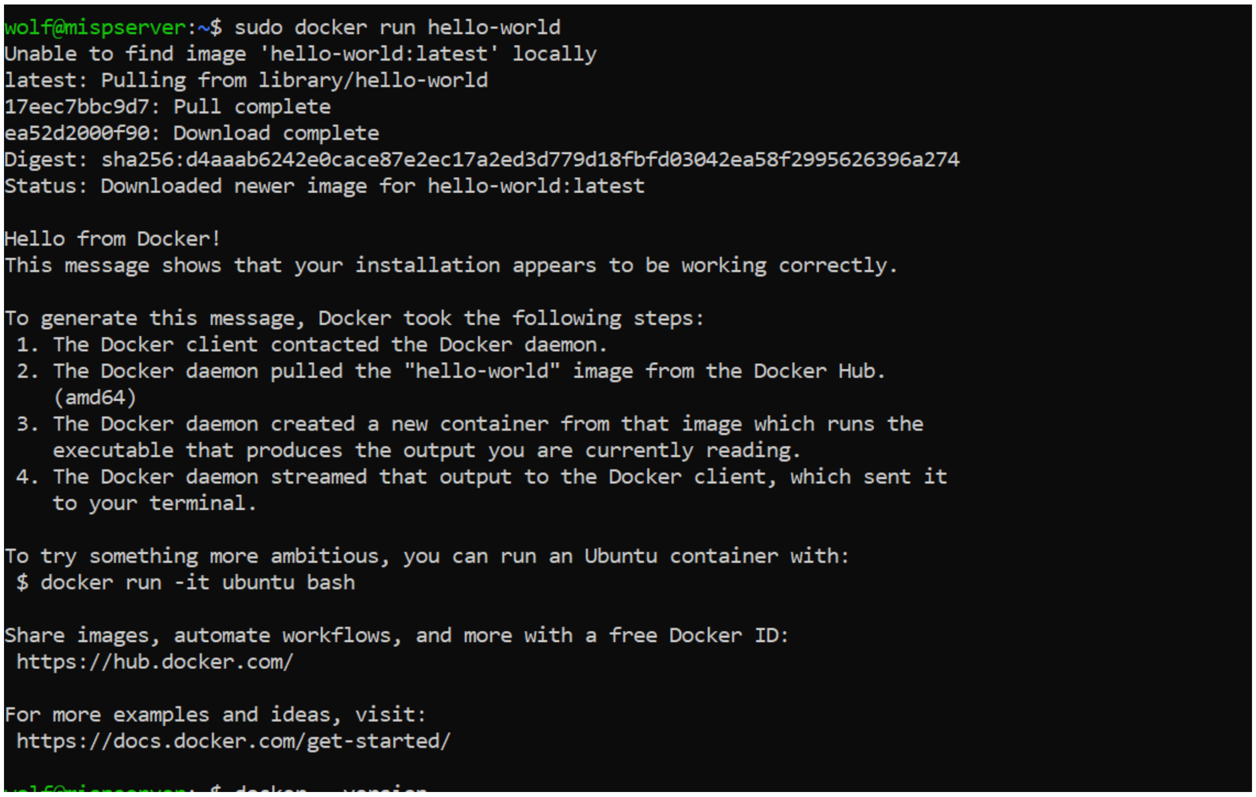
```
wolf@mispserver: ~/misp-docker
ll packages are up to date.
olf@mispserver:~$ sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
eading package lists... Done
uilding dependency tree... Done
eading state information... Done
he following additional packages will be installed:
 docker-ce-rootless-extras libslirp0 pigz slirp4netns
uggested packages:
 cgroupfs-mount | cgroup-lite docker-model-plugin
he following NEW packages will be installed:
 containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libslirp0
 pigz slirp4netns
 upgraded, 9 newly installed, 0 to remove and 0 not upgraded.
eed to get 91.3 MB of archives.
fter this operation, 364 MB of additional disk space will be used.
o you want to continue? [Y/n] y
et:1 http://pk.archive.ubuntu.com/ubuntu noble/universe amd64 pigz amd64 2.8-1 [65.6 kB]
et:2 https://download.docker.com/linux/ubuntu noble/stable amd64 containerd.io amd64 2.2.1-1~ubuntu.24.04~noble [23.4 MB]
et:3 http://pk.archive.ubuntu.com/ubuntu noble/main amd64 libslirp0 amd64 4.7.0-1ubuntu3 [63.8 kB]
et:4 http://pk.archive.ubuntu.com/ubuntu noble/universe amd64 slirp4netns amd64 1.2.1-1build2 [34.9 kB]
et:5 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce-cli amd64 5:29.1.3-1~ubuntu.24.04~noble [16.3 MB]
et:6 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce amd64 5:29.1.3-1~ubuntu.24.04~noble [21.0 MB]
et:7 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-buildx-plugin amd64 0.30.1-1~ubuntu.24.04~noble [16.4 MB]
et:8 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce-rootless-extras amd64 5:29.1.3-1~ubuntu.24.04~noble [6,383 kB]
et:9 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-compose-plugin amd64 5.0.0-1~ubuntu.24.04~noble [7,709 kB]
etched 91.3 MB in 1min 8s (1,338 kB/s)
electing previously unselected package containerd.io.
(Reading database ... 87330 files and directories currently installed.)
reparing to unpack .../0-containerd.io_2.2.1-1~ubuntu.24.04~noble_amd64.deb ...
npacking containerd.io (2.2.1-1~ubuntu.24.04~noble) ...
electing previously unselected package docker-ce-cli.
reparing to unpack .../1-docker-ce-cli_5%3a29.1.3-1~ubuntu.24.04~noble_amd64.deb ...
npacking docker-ce-cli (5:29.1.3-1~ubuntu.24.04~noble) ...
electing previously unselected package docker-ce.
reparing to unpack .../2-docker-ce_5%3a29.1.3-1~ubuntu.24.04~noble_amd64.deb ...
npacking docker-ce (5:29.1.3-1~ubuntu.24.04~noble) ...
electing previously unselected package pigz.
reparing to unpack .../3-pigz_2.8-1_amd64.deb ...
npacking pigz (2.8-1) ...
electing previously unselected package docker-buildx-plugin.
```

This screenshot shows the installation of Docker Engine, Docker CLI, container runtime, Buildx plugin, and Docker Compose plugin on the Ubuntu server. Installing these components is crucial to deploy containerized applications, such as MISP, in a stable and isolated environment. Using Docker ensures that MISP runs consistently across the lab, simplifying maintenance and reducing dependency conflicts



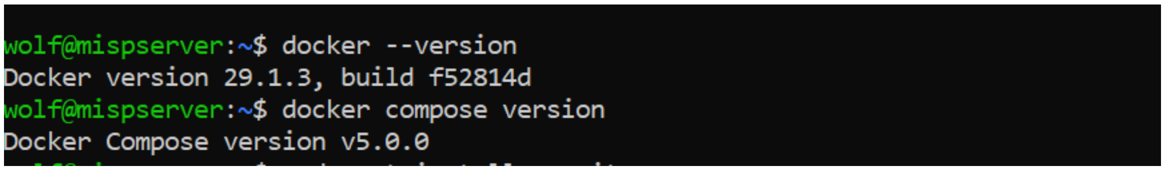
```
wolf@mispserver:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
     Active: active (running) since Sun 2025-12-28 18:15:40 UTC; 24s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 10228 (dockerd)
      Tasks: 10
     Memory: 25.7M (peak: 25.9M)
        CPU: 474ms
     CGroup: /system.slice/docker.service
             └─10228 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Dec 28 18:15:39 mispserver dockerd[10228]: time="2025-12-28T18:15:39.798166274Z" level=info msg="Restoring containers: >Dec 28 18:15:39
5-12-28T18:15:39.832476552Z" level=info msg="Deleting nftables IPv4>Dec 28 18:15:39 mispserver dockerd[10228]: time="2025-12-28T18:15:39
g nftables IPv6>Dec 28 18:15:40 mispserver dockerd[10228]: time="2025-12-28T18:15:40.257499852Z" level=info msg="Loading containers: do>
0228]: time="2025-12-28T18:15:40.267121022Z" level=info msg="Docker daemon" commit=>Dec 28 18:15:40 mispserver dockerd[10228]: time="202
nfo msg="Initializing buildkit"
Dec 28 18:15:40 mispserver dockerd[10228]: time="2025-12-28T18:15:40.299661032Z" level=info msg="Completed buildkit ini>Dec 28 18:15:40
5-12-28T18:15:40.306832187Z" level=info msg="Daemon has completed i>Dec 28 18:15:40 mispserver dockerd[10228]: time="2025-12-28T18:15:40
ten on /run/doc>Dec 28 18:15:40 mispserver systemd[1]: Started docker.service - Docker Application Container Engine.
```

This screenshot shows the Docker service status on the Ubuntu server, confirming that Docker is active and running. Verifying that Docker is operational ensures that containerized applications, such as MISP, can be deployed successfully.



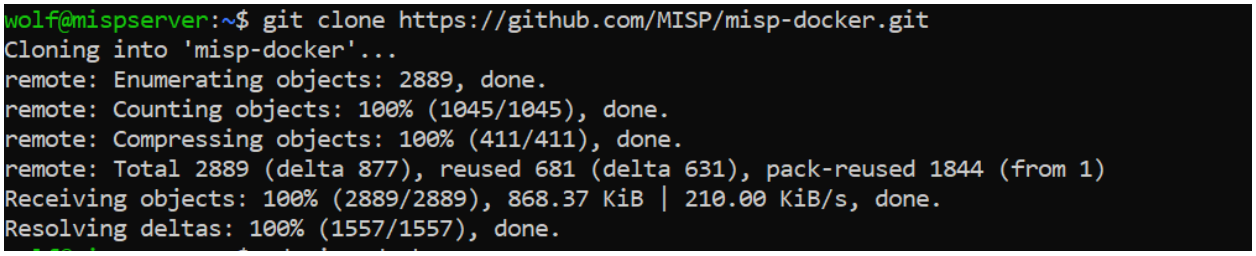
```
wolf@mispserver:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
17eec7bbc9d7: Pull complete
ea52d2000f90: Download complete
Digest: sha256:d4aaab6242e0cace87e2ec17a2ed3d779d18fbfd03042ea58f2995626396a274
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

This screenshot shows the successful execution of the `sudo docker run hello-world` command. It verifies that Docker is installed correctly and can run containers on the system. This test ensures that the environment is ready for deploying containerized applications



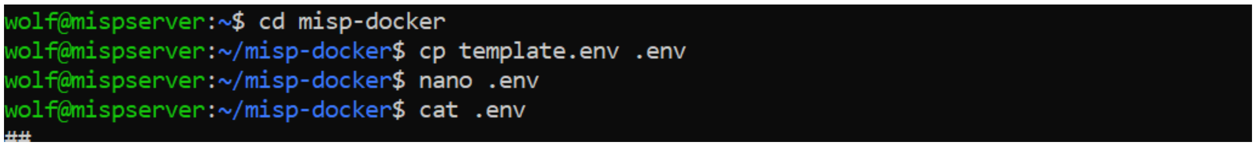
```
wolf@mispserver:~$ docker --version
Docker version 29.1.3, build f52814d
wolf@mispserver:~$ docker compose version
Docker Compose version v5.0.0
```

This screenshot shows the versions of Docker Engine and Docker Compose installed on the Ubuntu server.



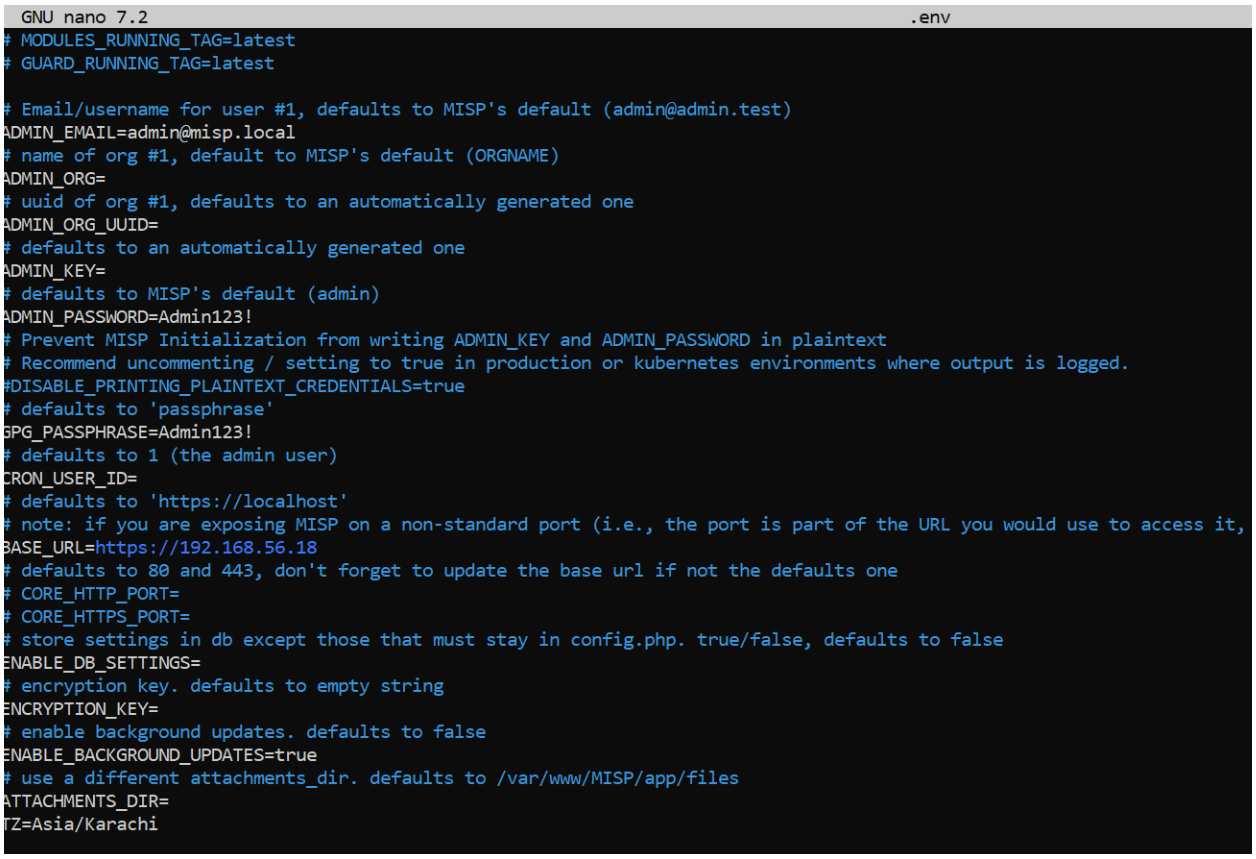
```
wolf@mispserver:~$ git clone https://github.com/MISP/misp-docker.git
Cloning into 'misp-docker'...
remote: Enumerating objects: 2889, done.
remote: Counting objects: 100% (1045/1045), done.
remote: Compressing objects: 100% (411/411), done.
remote: Total 2889 (delta 877), reused 681 (delta 631), pack-reused 1844 (from 1)
Receiving objects: 100% (2889/2889), 868.37 KiB | 210.00 KiB/s, done.
Resolving deltas: 100% (1557/1557), done.
```

This screenshot shows the command used to clone the MISP repository from GitHub onto the Ubuntu server. Cloning the official MISP repository is the first step in setting up the platform for deployment.



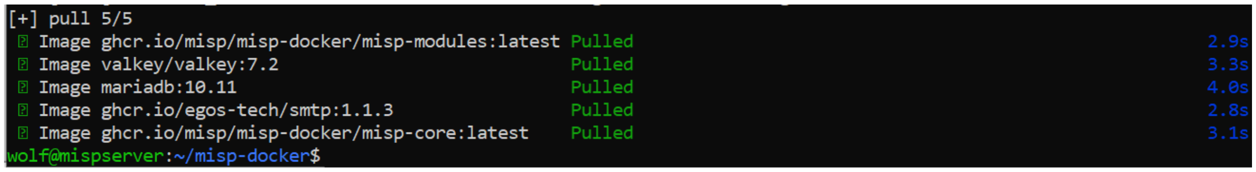
```
wolf@mispserver:~$ cd misp-docker
wolf@mispserver:~/misp-docker$ cp template.env .env
wolf@mispserver:~/misp-docker$ nano .env
wolf@mispserver:~/misp-docker$ cat .env
##
```

This screenshot shows navigating into the cloned `misp-docker` directory, copying the template environment file to `.env`, and opening it for editing using `nano`. Configuring the `.env` file is essential to set environment-specific variables



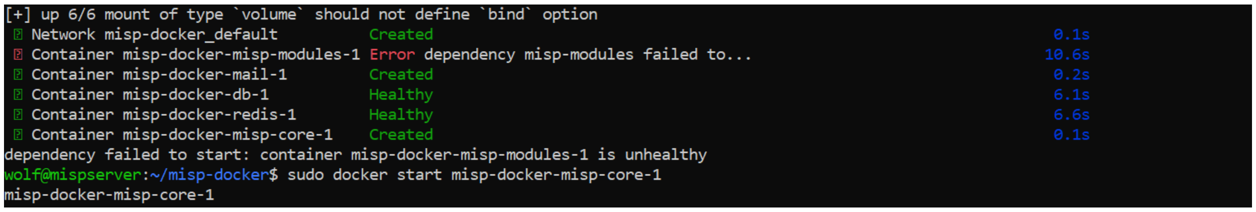
```
  GNU nano 7.2                                                               .env
# MODULES_RUNNING_TAG=latest
# GUARD_RUNNING_TAG=latest

# Email/username for user #1, defaults to MISP's default (admin@admin.test)
ADMIN_EMAIL=admin@misp.local
# name of org #1, default to MISP's default (ORGNAME)
ADMIN_ORG=
# uuid of org #1, defaults to an automatically generated one
ADMIN_ORG_UUID=
# defaults to an automatically generated one
ADMIN_KEY=
# defaults to MISP's default (admin)
ADMIN_PASSWORD=Admin123!
# Prevent MISP Initialization from writing ADMIN_KEY and ADMIN_PASSWORD in plaintext
# Recommend uncommenting / setting to true in production or kubernetes environments where output is logged.
#DISABLE_PRINTING_PLAINTEXT_CREDENTIALS=true
# defaults to 'passphrase'
GPG_PASSPHRASE=Admin123!
# defaults to 1 (the admin user)
CRON_USER_ID=
# defaults to 'https://localhost'
# note: if you are exposing MISP on a non-standard port (i.e., the port is part of the URL you would use to access it,
BASE_URL=https://192.168.56.18
# defaults to 80 and 443, don't forget to update the base url if not the defaults one
# CORE_HTTP_PORT=
# CORE_HTTPS_PORT=
# store settings in db except those that must stay in config.php. true/false, defaults to false
ENABLE_DB_SETTINGS=
# encryption key. defaults to empty string
ENCRYPTION_KEY=
# enable background updates. defaults to false
ENABLE_BACKGROUND_UPDATES=true
# use a different attachments_dir. defaults to /var/www/MISP/app/files
ATTACHMENTS_DIR=
TZ=Asia/Karachi
```

This screenshot shows the `.env` file being edited to configure key environment variables for the MISP Docker deployment. The admin email and password were set to secure access, the GPG passphrase was configured for signing events, the `BASEURL` was set to the server's local IP (`https://192.168.56.18`) for proper web access, and the timezone was set to `Asia/Karachi`. Correctly configuring these variables ensures the MISP instance runs securely, is accessible within the lab network, and is synchronized with the local timezone, which is critical for accurate logging and SOC operations.

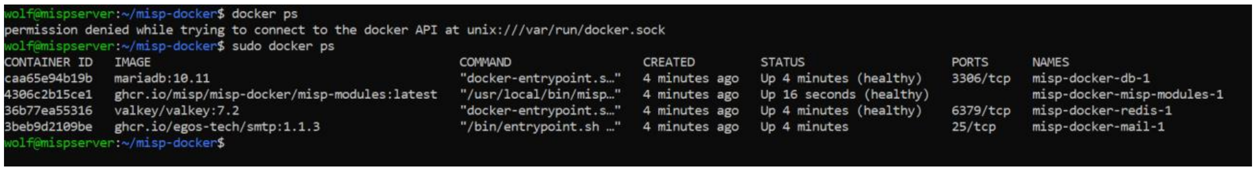


```
[+] pull 5/5
 ✔ Image ghcr.io/misp/misp-docker/misp-modules:latest Pulled                                                 2.9s
 ✔ Image valkey/valkey:7.2                            Pulled                                                 3.3s
 ✔ Image mariadb:10.11                                Pulled                                                 4.0s
 ✔ Image ghcr.io/egos-tech/smtp:1.1.3                 Pulled                                                 2.8s
 ✔ Image ghcr.io/misp/misp-docker/misp-core:latest    Pulled                                                 3.1s
wolf@mispserver:~/misp-docker$
```

This screenshot shows the execution of `sudo docker compose pull`, which downloads all required Docker images for the MISP deployment. Pulling the latest container images ensures that the environment uses up-to-date and stable versions of MISP services.

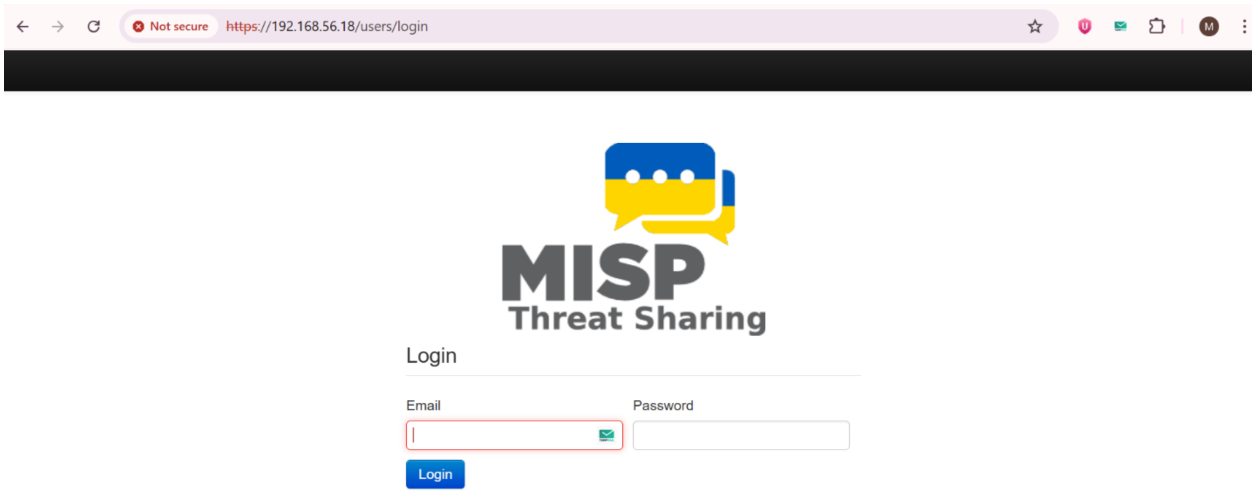


```
[+] up 6/6 mount of type `volume` should not define `bind` option
 ▯ Network misp-docker_default          Created                                                 0.1s
 ▯ Container misp-docker-misp-modules-1 Error dependency misp-modules failed to...              10.6s
 ▯ Container misp-docker-mail-1         Created                                                 0.2s
 ▯ Container misp-docker-db-1           Healthy                                                 6.1s
 ▯ Container misp-docker-redis-1        Healthy                                                 6.6s
 ▯ Container misp-docker-misp-core-1    Created                                                 0.1s
dependency failed to start: container misp-docker-misp-modules-1 is unhealthy
wolf@mispserver:~/misp-docker$ sudo docker start misp-docker-misp-core-1
misp-docker-misp-core-1
```

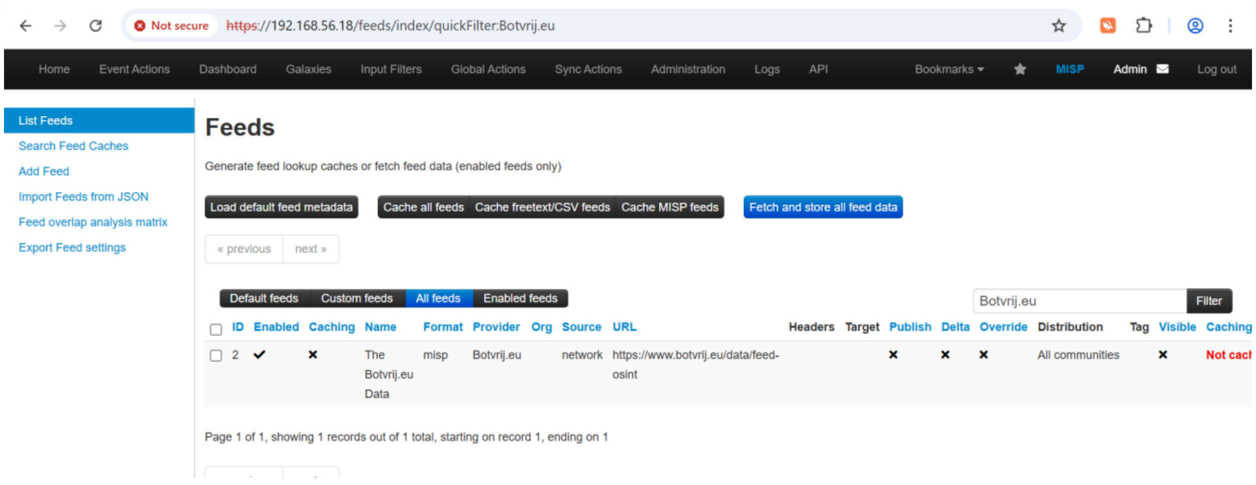


```
wolf@mispserver:~/misp-docker$ docker ps
permission denied while trying to connect to the docker API at unix:///var/run/docker.sock
wolf@mispserver:~/misp-docker$ sudo docker ps
CONTAINER ID   IMAGE                                       COMMAND                  CREATED         STATUS                    PORTS      NAMES
caa65e94b19b   mariadb:10.11                               "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes (healthy)    3306/tcp   misp-docker-db-1
4306c2b15ce1   ghcr.io/misp/misp-docker/misp-modules:latest   "/usr/local/bin/misp…"   4 minutes ago   Up 16 seconds (healthy)              misp-docker-misp-modules-1
36b77ea55316   valkey/valkey:7.2                           "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes (healthy)    6379/tcp   misp-docker-redis-1
3beb9d2109be   ghcr.io/egos-tech/smtp:1.1.3                "/bin/entrypoint.sh …"   4 minutes ago   Up 4 minutes              25/tcp     misp-docker-mail-1
wolf@mispserver:~/misp-docker$
```

This screenshot shows the execution of `sudo docker compose up -d` to start all MISP containers in detached mode, followed by `docker ps` to verify that the containers are running. This step confirms that all MISP services, including the web interface, database, and background jobs, are active and operational



This screenshot shows successful access to the MISP web interface using the configured base URL (`https://192.168.56.18`). It confirms that the MISP Docker deployment is running correctly and that the web service is reachable from the lab network.



Feeds

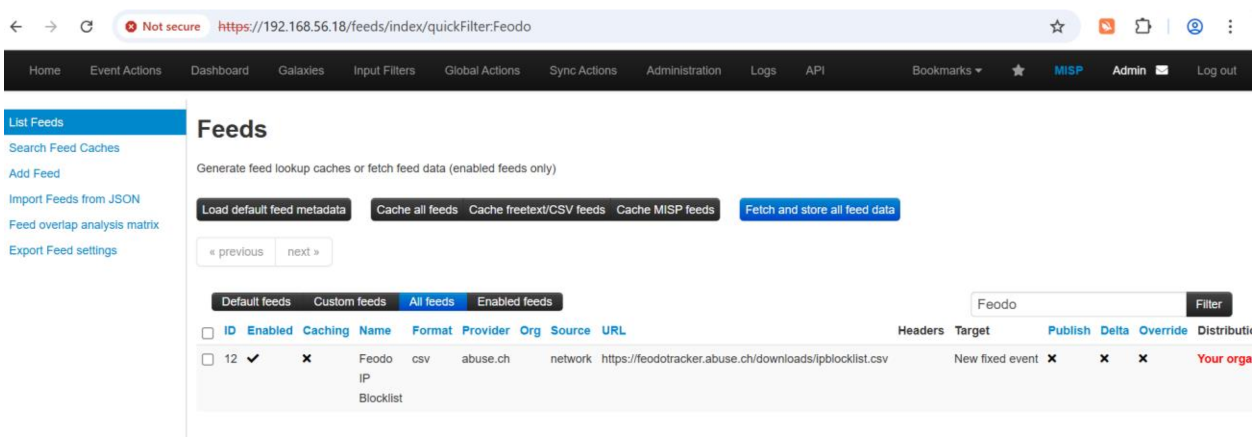
Generate feed lookup caches or fetch feed data (enabled feeds only)

Botvrij.eu

ID	Enabled	Caching	Name	Format	Provider	Org	Source	URL	Headers	Target	Publish	Delta	Override	Distribution	Tag	Visible	Caching
2	✔	✖	The Botvrij.eu Data	misp	Botvrij.eu		network	https://www.botvrij.eu/data/feed-osint			✖	✖	✖	All communities		✖	Not cach

Page 1 of 1, showing 1 records out of 1 total, starting on record 1, ending on 1

This screenshot shows the Botvrij.eu threat intelligence feed being enabled within the MISP platform. Botvrij.eu provides high-quality indicators related to botnet activity, malicious IP addresses, and command-and-control infrastructure. Enabling this feed allows MISP to continuously ingest updated threat indicators, which are later shared with Wazuh for enhanced detection and correlation in the SOC environment



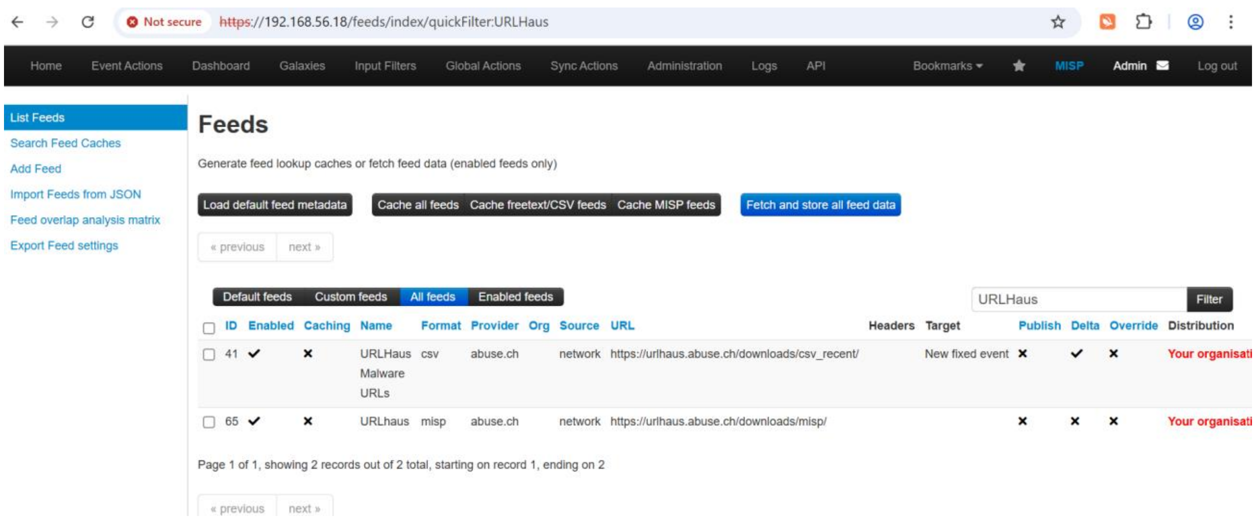
Feeds

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

Feodo

ID	Enabled	Caching	Name	Format	Provider	Org	Source	URL	Headers	Target	Publish	Delta	Override	Distributi
12	✔	✖	Feodo IP Blocklist	csv	abuse.ch		network	https://feodotracker.abuse.ch/downloads/ipblocklist.csv		New fixed event	✖	✖	✖	Your orga

This screenshot displays the activation of the Feodo Tracker feed in MISP. The Feodo Tracker feed supplies indicators associated with Feodo (Dridex) banking trojans, including malicious IPs and network indicators. Integrating this feed strengthens threat intelligence coverage and improves the ability of Wazuh to detect known malware-related network activity whencorrelating honeypot events.



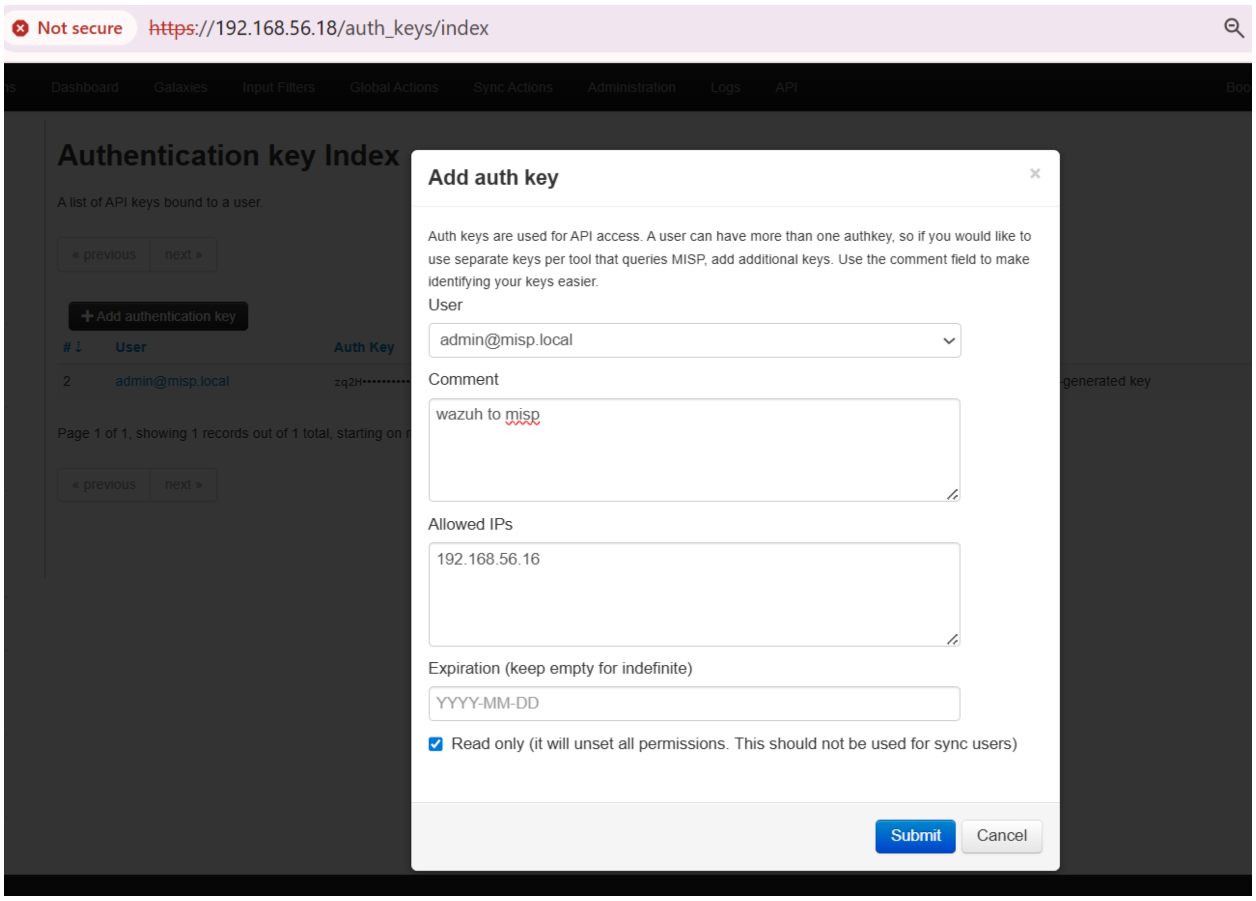
Feeds

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

ID	Enabled	Caching	Name	Format	Provider	Org	Source	URL	Headers	Target	Publish	Delta	Override	Distribution
41	✓	✗	URLHaus Malware URLs	csv	abuse.ch		network	https://urlhaus.abuse.ch/downloads/csv_recent/		New fixed event	✗	✓	✗	Your organisati
65	✓	✗	URLhaus	misp	abuse.ch		network	https://urlhaus.abuse.ch/downloads/misp/			✗	✗	✗	Your organisati

Page 1 of 1, showing 2 records out of 2 total, starting on record 1, ending on 2

This screenshot shows the URLhaus threat intelligence feed enabled in MISP. URLhaus provides indicators related to malicious URLs used for malware distribution, phishing, and payload delivery. By enabling this feed, MISP gains access to actively maintained IOC data, which enhances Wazuh’s ability to identify and correlate suspicious network activity observed by the Cowrie honeypot



Auth keys are used for API access. A user can have more than one authkey, so if you would like to use separate keys per tool that queries MISP, add additional keys. Use the comment field to make identifying your keys easier.

User: admin@misp.local

Comment: wazuh to misp

Allowed IPs: 192.168.56.16

Expiration (keep empty for indefinite): YYYY-MM-DD

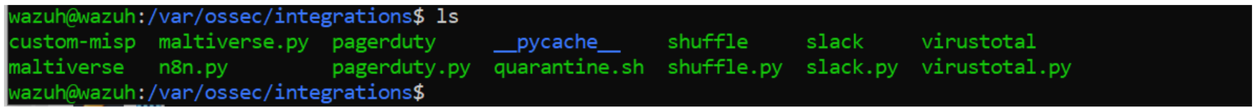
[✓] Read only (it will unset all permissions. This should not be used for sync users)

This screenshot shows the generation of an API key in MISP for the Wazuh server (IP: `192.168.56.16`). The API key allows Wazuh to securely access MISP threat intelligence indicators, such as malicious IPs, domains, and hashes. Generating and using a dedicated API key ensures secure communication between Wazuh and MISP,

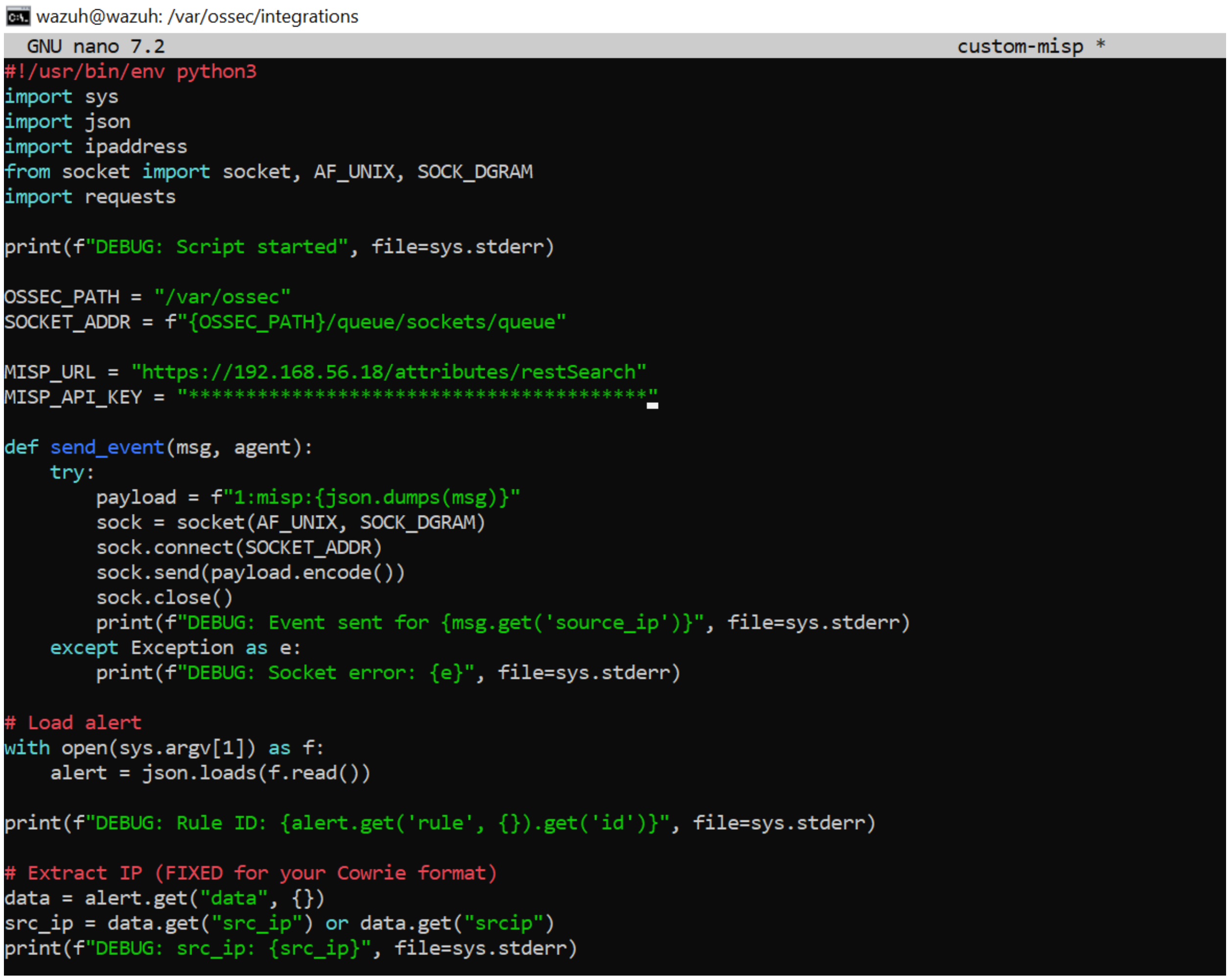
MISP Integration with Wazuh

The purpose of integrating MISP with Wazuh is to allow the SIEM **to** automatically ingest external threat intelligence and correlate it with live attack events from the honeypot. This improves detection accuracy, reduces false positives, and provides actionable alerts for SOC analysts.

The Wazuh Manager was deployed on a dedicated **Ubuntu server** to act as the central SIEM in the SOC lab environment. This server is responsible for collecting and analyzing logs from agents, correlating events, and generating alerts. The installation included all necessary Wazuh components, ensuring that the manager is fully operational and ready to receive logs from the honeypot (Cowrie) and threat intelligence indicators from MISP.



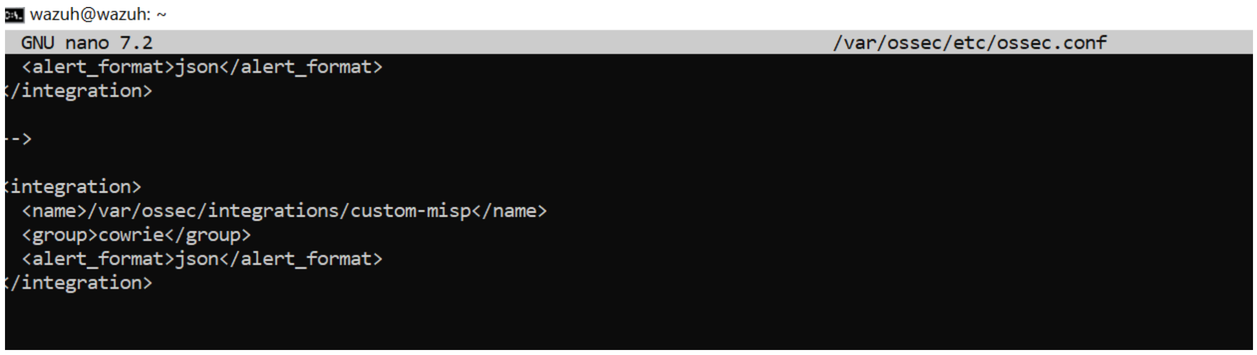
```
wazuh@wazuh:/var/ossec/integrations$ ls
custom-misp  maltiverse.py  pagerduty     __pycache__   shuffle     slack     virustotal
maltiverse   n8n.py         pagerduty.py  quarantine.sh  shuffle.py  slack.py  virustotal.py
wazuh@wazuh:/var/ossec/integrations$
```



```
wazuh@wazuh: /var/ossec/integrations
  GNU nano 7.2                                                              custom-misp *
#!/usr/bin/env python3
import sys
import json
import ipaddress
from socket import socket, AF_UNIX, SOCK_DGRAM
import requests

print(f"DEBUG: Script started", file=sys.stderr)

OSSEC_PATH = "/var/ossec"
SOCKET_ADDR = f"{OSSEC_PATH}/queue/sockets/queue"

MISP_URL = "https://192.168.56.18/attributes/restSearch"
MISP_API_KEY = "****************************************"

def send_event(msg, agent):
    try:
        payload = f"1:misp:{json.dumps(msg)}"
        sock = socket(AF_UNIX, SOCK_DGRAM)
        sock.connect(SOCKET_ADDR)
        sock.send(payload.encode())
        sock.close()
        print(f"DEBUG: Event sent for {msg.get('source_ip')}", file=sys.stderr)
    except Exception as e:
        print(f"DEBUG: Socket error: {e}", file=sys.stderr)

# Load alert
with open(sys.argv[1]) as f:
    alert = json.loads(f.read())

print(f"DEBUG: Rule ID: {alert.get('rule', {}).get('id')}", file=sys.stderr)

# Extract IP (FIXED for your Cowrie format)
data = alert.get("data", {})
src_ip = data.get("src_ip") or data.get("srcip")
print(f"DEBUG: src_ip: {src_ip}", file=sys.stderr)
```

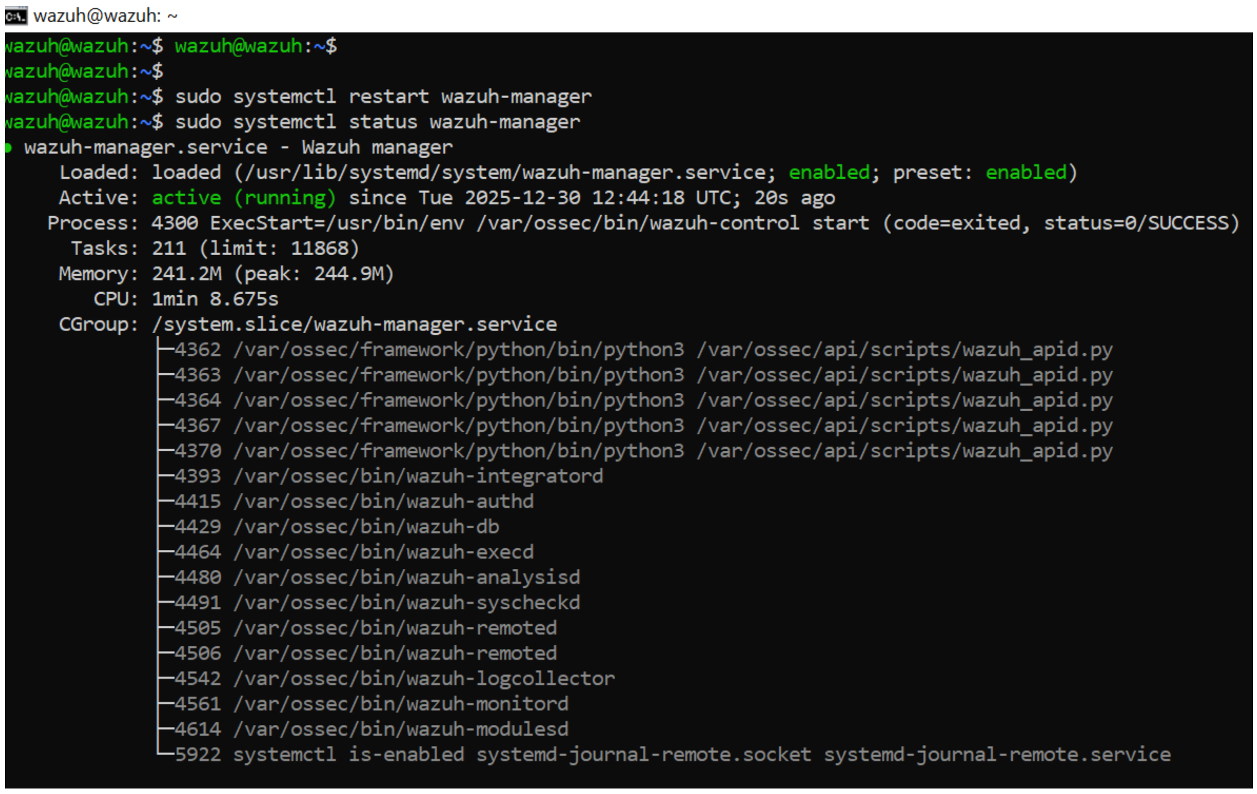
This screenshot shows the creation of a new custom integration script (`custom-misp`) in the Wazuh integration directory (`/var/ossec/integration`). The script is designed to fetch threat intelligence indicators from the MISP server. Key configurations, such as the MISP `BASEURL` (`https://192.168.56.18`) and the API key generated for Wazuh, were added to the script. This allows Wazuh to securely communicate with MISP and ingest threat intelligence for correlation with honeypot events, enhancing SOC detection capabilities.

commands used to set the ownership and permissions of the `custom-misp` script. The command `chown root:wazuh custom-misp` assigns ownership to the root user and the Wazuh group, while `chmod 750 custom-misp` sets read, write, and execute permissions for the owner and read-execute for the group.



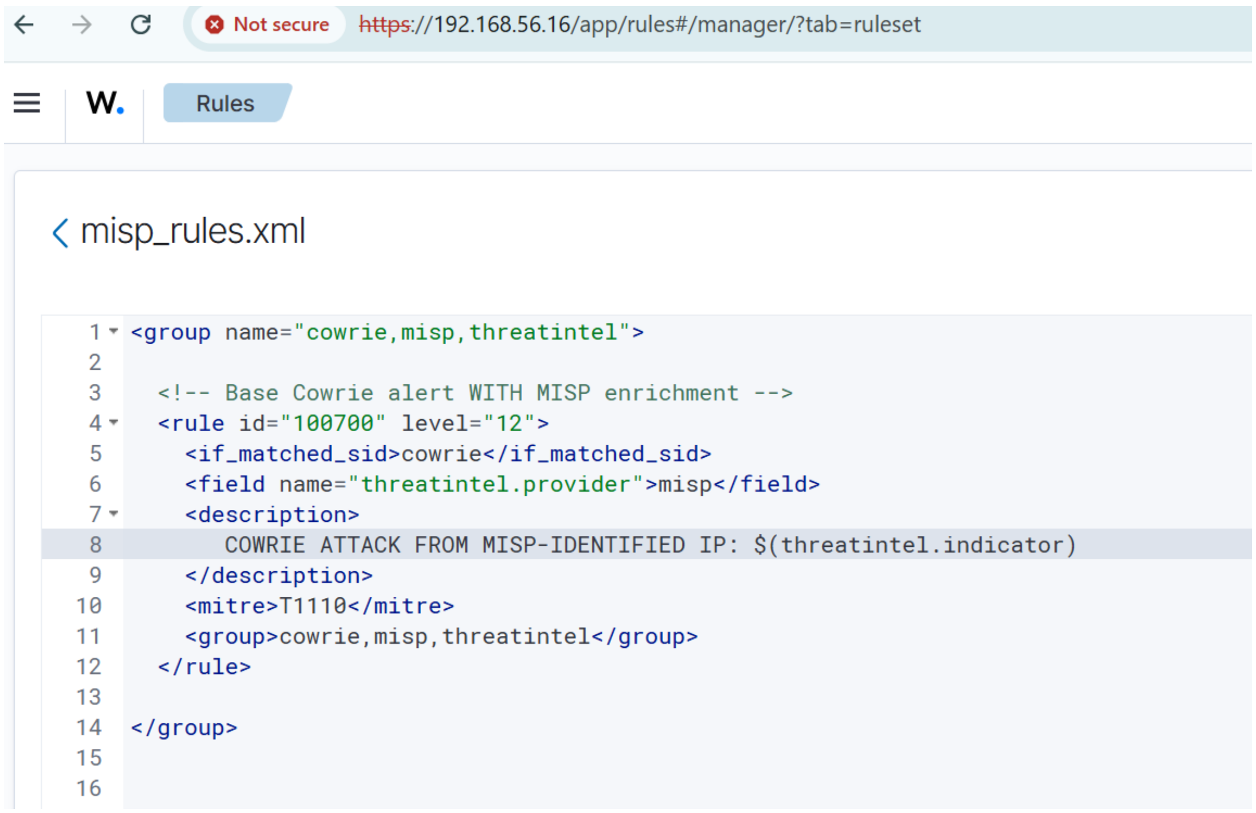
```
wazuh@wazuh: ~
  GNU nano 7.2                                                        /var/ossec/etc/ossec.conf
  <alert_format>json</alert_format>
</integration>

-->

<integration>
  <name>/var/ossec/integrations/custom-misp</name>
  <group>cowrie</group>
  <alert_format>json</alert_format>
</integration>
```

This screenshot shows the addition of a custom integration block in the Wazuh configuration file (`ossec.conf`). The block specifies that the `custom-misp` script should be executed by Wazuh to retrieve threat intelligence indicators from the MISP server.



```
wazuh@wazuh: ~
wazuh@wazuh:~$ wazuh@wazuh:~$
wazuh@wazuh:~$
wazuh@wazuh:~$ sudo systemctl restart wazuh-manager
wazuh@wazuh:~$ sudo systemctl status wazuh-manager
● wazuh-manager.service - Wazuh manager
     Loaded: loaded (/usr/lib/systemd/system/wazuh-manager.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-12-30 12:44:18 UTC; 20s ago
    Process: 4300 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0/SUCCESS)
      Tasks: 211 (limit: 11868)
     Memory: 241.2M (peak: 244.9M)
        CPU: 1min 8.675s
     CGroup: /system.slice/wazuh-manager.service
             ├─4362 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─4363 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─4364 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─4367 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─4370 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─4393 /var/ossec/bin/wazuh-integratord
             ├─4415 /var/ossec/bin/wazuh-authd
             ├─4429 /var/ossec/bin/wazuh-db
             ├─4464 /var/ossec/bin/wazuh-execd
             ├─4480 /var/ossec/bin/wazuh-analysisd
             ├─4491 /var/ossec/bin/wazuh-syscheckd
             ├─4505 /var/ossec/bin/wazuh-remoted
             ├─4506 /var/ossec/bin/wazuh-remoted
             ├─4542 /var/ossec/bin/wazuh-logcollector
             ├─4561 /var/ossec/bin/wazuh-monitord
             ├─4614 /var/ossec/bin/wazuh-modulesd
             └─5922 systemctl is-enabled systemd-journal-remote.socket systemd-journal-remote.service
```

This screenshot shows the Wazuh Manager being restarted after adding the custom MISP integration, followed by checking its status. The `active` status confirms that the manager is running correctly and the integration script is ready to fetch threat intelligence from MISP. Restarting the manager ensures that all configuration changes are applied,



```xml
<group name="cowrie,misp,threatintel">

  <!-- Base Cowrie alert WITH MISP enrichment -->
  <rule id="100700" level="12">
    <if_matched_sid>cowrie</if_matched_sid>
    <field name="threatintel.provider">misp</field>
    <description>
       COWRIE ATTACK FROM MISP-IDENTIFIED IP: $(threatintel.indicator)
    </description>
    <mitre>T1110</mitre>
    <group>cowrie,misp,threatintel</group>
  </rule>

</group>
```

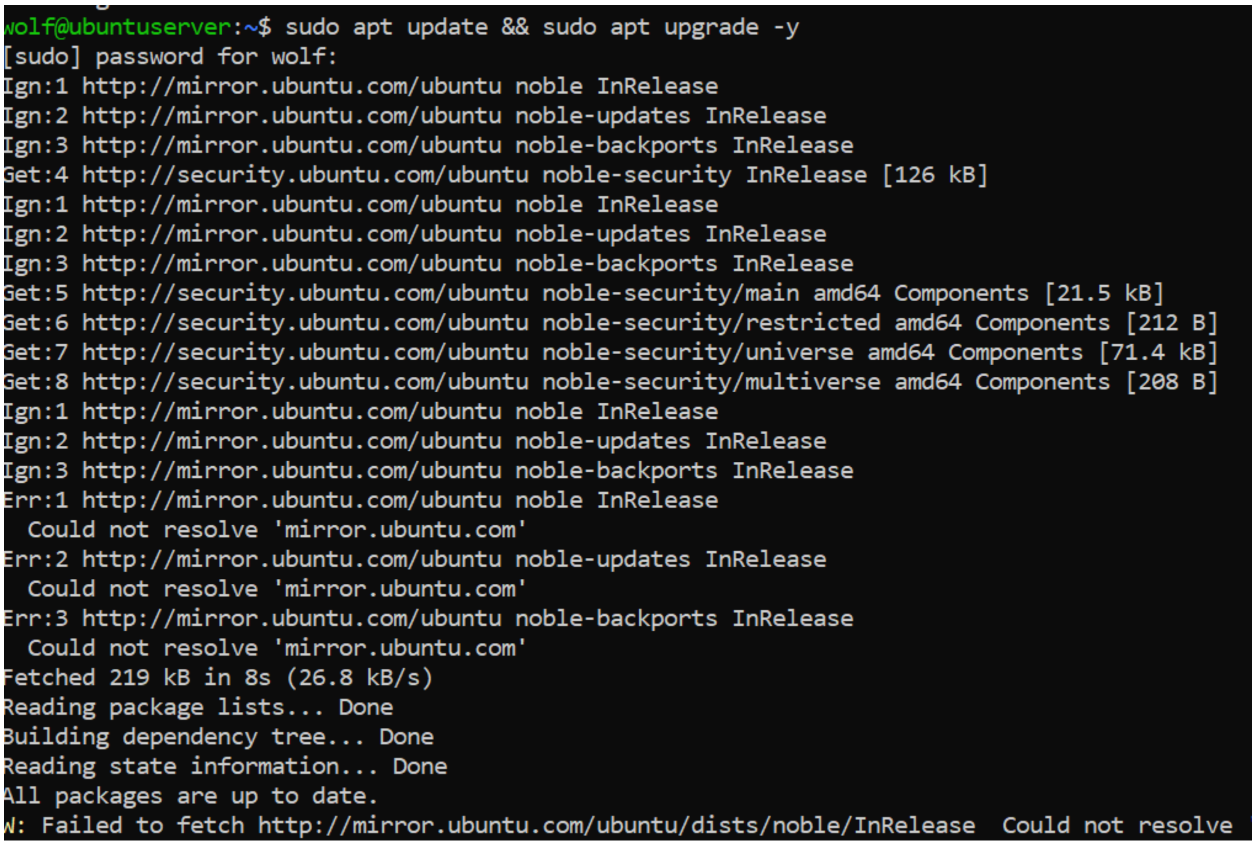
This screenshot shows the creation of custom rules in the Wazuh web interface for MISP integration. These rules define how alerts are generated when incoming honeypot logs match threat intelligence indicators fetched from MISP. Creating custom rules allows the SOC to prioritize critical threats, reduce false positives, and ensure accurate detection of malicious activity, enhancing the overall efficiency of the monitoring and response process.

Cowrie Honeypot Setup

Cowrie is a medium-interaction SSH **honeypot** designed to simulate a vulnerable SSH service in order to attract attackers. It captures attacker behavior such as brute-force attempts, login activity, and executed commands. In this task, Cowrie was deployed to generate real attack data, which is then forwarded to Wazuh for detection and correlation with MISP threat intelligence.

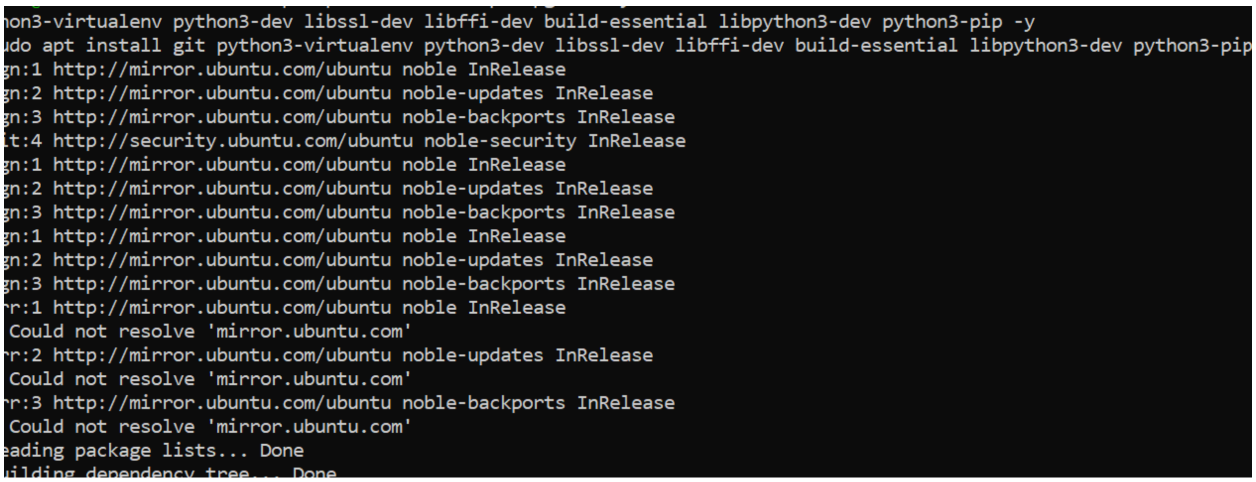
Cowrie Deployment on Ubuntu Server

The Cowrie honeypot was installed on a dedicated Ubuntu server, which also hosts the **Wazuh agent**. This separation ensures that attacker activity is isolated from the SIEM and threat intelligence servers, reflecting real-world SOC architecture.



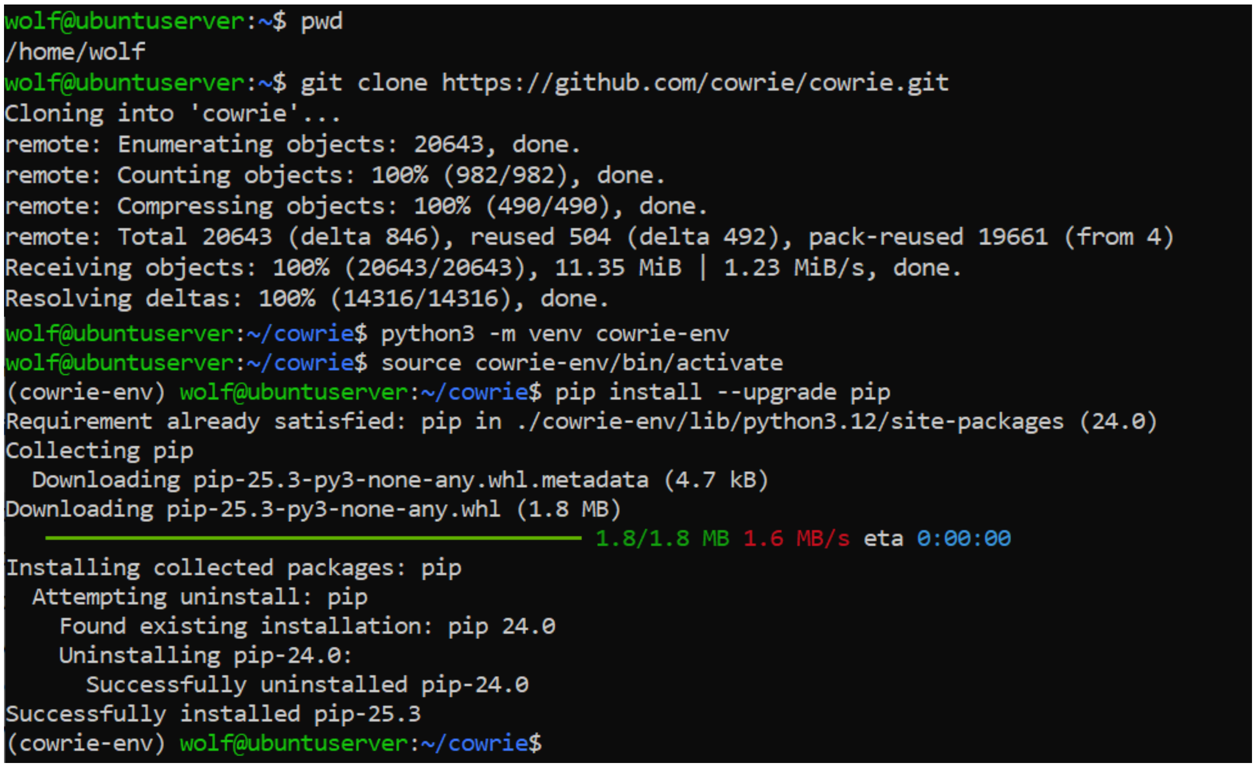
```
wolf@ubuntuserver:~$ sudo apt update && sudo apt upgrade -y
[sudo] password for wolf:
Ign:1 http://mirror.ubuntu.com/ubuntu noble InRelease
Ign:2 http://mirror.ubuntu.com/ubuntu noble-updates InRelease
Ign:3 http://mirror.ubuntu.com/ubuntu noble-backports InRelease
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Ign:1 http://mirror.ubuntu.com/ubuntu noble InRelease
Ign:2 http://mirror.ubuntu.com/ubuntu noble-updates InRelease
Ign:3 http://mirror.ubuntu.com/ubuntu noble-backports InRelease
Get:5 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.5 kB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:7 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [71.4 kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [208 B]
Ign:1 http://mirror.ubuntu.com/ubuntu noble InRelease
Ign:2 http://mirror.ubuntu.com/ubuntu noble-updates InRelease
Ign:3 http://mirror.ubuntu.com/ubuntu noble-backports InRelease
Err:1 http://mirror.ubuntu.com/ubuntu noble InRelease
  Could not resolve 'mirror.ubuntu.com'
Err:2 http://mirror.ubuntu.com/ubuntu noble-updates InRelease
  Could not resolve 'mirror.ubuntu.com'
Err:3 http://mirror.ubuntu.com/ubuntu noble-backports InRelease
  Could not resolve 'mirror.ubuntu.com'
Fetched 219 kB in 8s (26.8 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
W: Failed to fetch http://mirror.ubuntu.com/ubuntu/dists/noble/InRelease  Could not resolve
```

This screenshot shows the Ubuntu system being updated and upgraded on the server designated for the Cowrie honeypot. Performing system updates ensures that all packages, dependencies, and security patches are up to date before installing and running the honeypot.



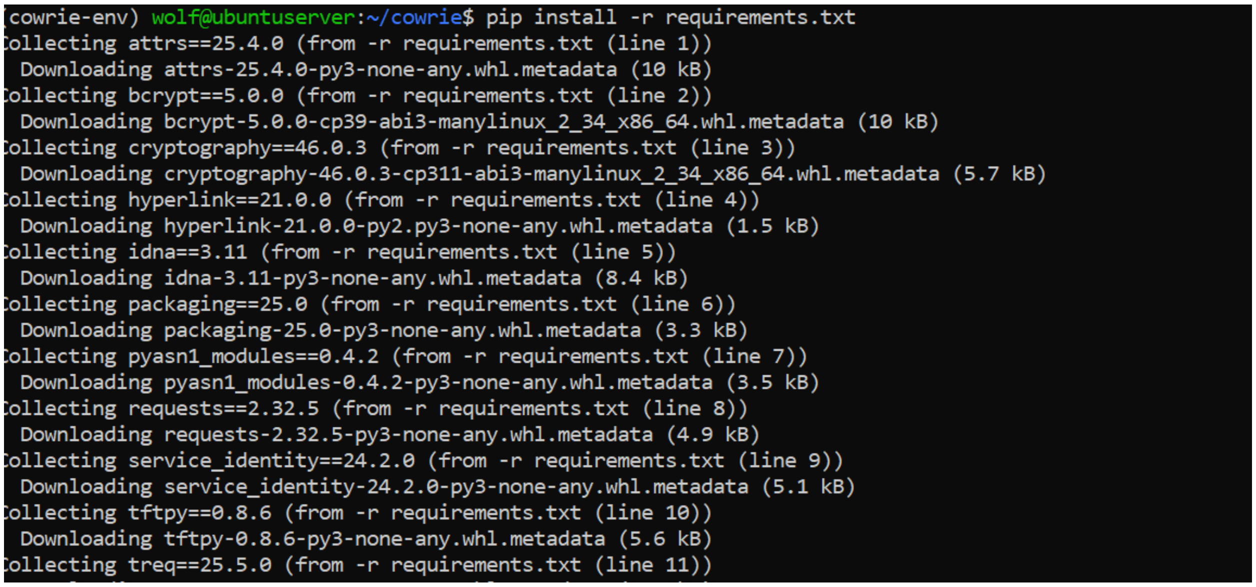
```
hon3-virtualenv python3-dev libssl-dev libffi-dev build-essential libpython3-dev python3-pip -y
udo apt install git python3-virtualenv python3-dev libssl-dev libffi-dev build-essential libpython3-dev python3-pip
gn:1 http://mirror.ubuntu.com/ubuntu noble InRelease
gn:2 http://mirror.ubuntu.com/ubuntu noble-updates InRelease
gn:3 http://mirror.ubuntu.com/ubuntu noble-backports InRelease
it:4 http://security.ubuntu.com/ubuntu noble-security InRelease
gn:1 http://mirror.ubuntu.com/ubuntu noble InRelease
gn:2 http://mirror.ubuntu.com/ubuntu noble-updates InRelease
gn:3 http://mirror.ubuntu.com/ubuntu noble-backports InRelease
gn:1 http://mirror.ubuntu.com/ubuntu noble InRelease
gn:2 http://mirror.ubuntu.com/ubuntu noble-updates InRelease
gn:3 http://mirror.ubuntu.com/ubuntu noble-backports InRelease
rr:1 http://mirror.ubuntu.com/ubuntu noble InRelease
 Could not resolve 'mirror.ubuntu.com'
rr:2 http://mirror.ubuntu.com/ubuntu noble-updates InRelease
 Could not resolve 'mirror.ubuntu.com'
rr:3 http://mirror.ubuntu.com/ubuntu noble-backports InRelease
 Could not resolve 'mirror.ubuntu.com'
eading package lists... Done
uilding dependency tree... Done
```

This screenshot shows the installation of Python 3 and required Python environment dependencies needed to run the Cowrie honeypot. Cowrie is a Python-based application, and installing the correct Python packages and virtual environment ensures proper execution and isolation of the honeypot.



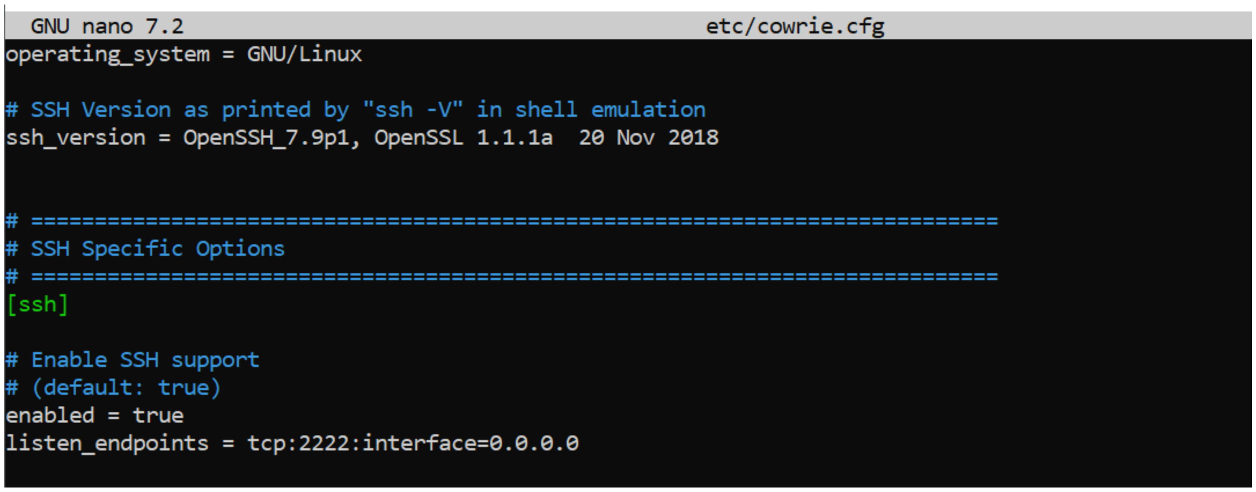
```
wolf@ubuntuserver:~$ pwd
/home/wolf
wolf@ubuntuserver:~$ git clone https://github.com/cowrie/cowrie.git
Cloning into 'cowrie'...
remote: Enumerating objects: 20643, done.
remote: Counting objects: 100% (982/982), done.
remote: Compressing objects: 100% (490/490), done.
remote: Total 20643 (delta 846), reused 504 (delta 492), pack-reused 19661 (from 4)
Receiving objects: 100% (20643/20643), 11.35 MiB | 1.23 MiB/s, done.
Resolving deltas: 100% (14316/14316), done.
wolf@ubuntuserver:~/cowrie$ python3 -m venv cowrie-env
wolf@ubuntuserver:~/cowrie$ source cowrie-env/bin/activate
(cowrie-env) wolf@ubuntuserver:~/cowrie$ pip install --upgrade pip
Requirement already satisfied: pip in ./cowrie-env/lib/python3.12/site-packages (24.0)
Collecting pip
  Downloading pip-25.3-py3-none-any.whl.metadata (4.7 kB)
Downloading pip-25.3-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 1.6 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 24.0
    Uninstalling pip-24.0:
      Successfully uninstalled pip-24.0
Successfully installed pip-25.3
(cowrie-env) wolf@ubuntuserver:~/cowrie$
```

This screenshot shows the cloning of the official Cowrie honeypot repository from GitHub, followed by the creation and activation of a Python virtual environment. The virtual environment is used to isolate Cowrie’s dependencies from the system-wide Python installation. Additionally, `pip` was upgraded to ensure compatibility with required Python packages.



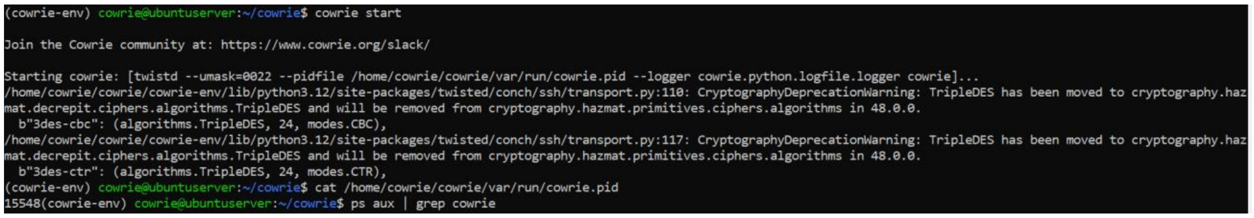
```
(cowrie-env) wolf@ubuntuserver:~/cowrie$ pip install -r requirements.txt
ollecting attrs==25.4.0 (from -r requirements.txt (line 1))
 Downloading attrs-25.4.0-py3-none-any.whl.metadata (10 kB)
ollecting bcrypt==5.0.0 (from -r requirements.txt (line 2))
 Downloading bcrypt-5.0.0-cp39-abi3-manylinux_2_34_x86_64.whl.metadata (10 kB)
ollecting cryptography==46.0.3 (from -r requirements.txt (line 3))
 Downloading cryptography-46.0.3-cp311-abi3-manylinux_2_34_x86_64.whl.metadata (5.7 kB)
ollecting hyperlink==21.0.0 (from -r requirements.txt (line 4))
 Downloading hyperlink-21.0.0-py2.py3-none-any.whl.metadata (1.5 kB)
ollecting idna==3.11 (from -r requirements.txt (line 5))
 Downloading idna-3.11-py3-none-any.whl.metadata (8.4 kB)
ollecting packaging==25.0 (from -r requirements.txt (line 6))
 Downloading packaging-25.0-py3-none-any.whl.metadata (3.3 kB)
ollecting pyasn1_modules==0.4.2 (from -r requirements.txt (line 7))
 Downloading pyasn1_modules-0.4.2-py3-none-any.whl.metadata (3.5 kB)
ollecting requests==2.32.5 (from -r requirements.txt (line 8))
 Downloading requests-2.32.5-py3-none-any.whl.metadata (4.9 kB)
ollecting service_identity==24.2.0 (from -r requirements.txt (line 9))
 Downloading service_identity-24.2.0-py3-none-any.whl.metadata (5.1 kB)
ollecting tftpy==0.8.6 (from -r requirements.txt (line 10))
 Downloading tftpy-0.8.6-py3-none-any.whl.metadata (5.6 kB)
ollecting treq==25.5.0 (from -r requirements.txt (line 11))
```

This screenshot shows the installation of required Python dependencies for the Cowrie honeypot using the `requirements.txt` file. Installing these dependencies ensures that all necessary libraries are available for Cowrie to function correctly.



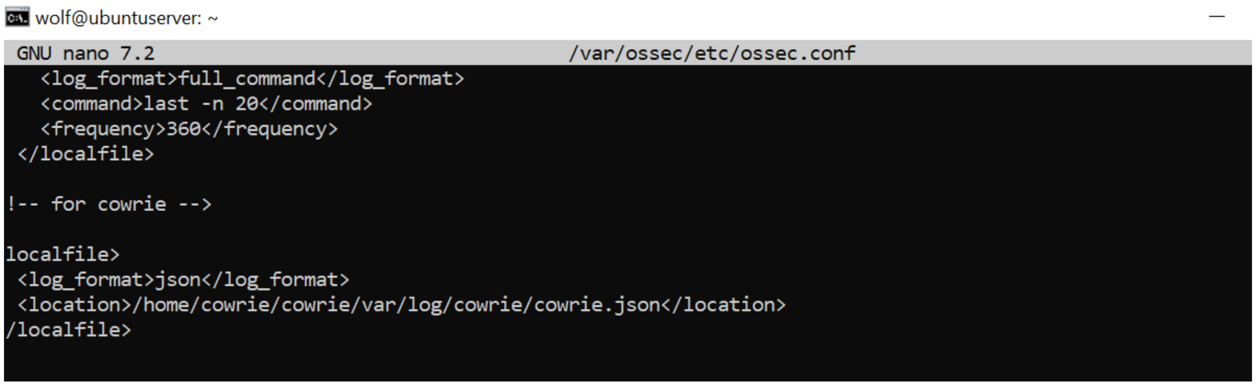
```
  GNU nano 7.2                                   etc/cowrie.cfg
operating_system = GNU/Linux

# SSH Version as printed by "ssh -V" in shell emulation
ssh_version = OpenSSH_7.9p1, OpenSSL 1.1.1a  20 Nov 2018


# ============================================================================
# SSH Specific Options
# ============================================================================
[ssh]

# Enable SSH support
# (default: true)
enabled = true
listen_endpoints = tcp:2222:interface=0.0.0.0
```

This screenshot shows the Cowrie configuration file (`cowrie.cfg`) being edited to enable the SSH service and configure the listening port. The SSH service was explicitly enabled, and port **2222** was specified to allow Cowrie to simulate an SSH server without conflicting with the system's default SSH service.



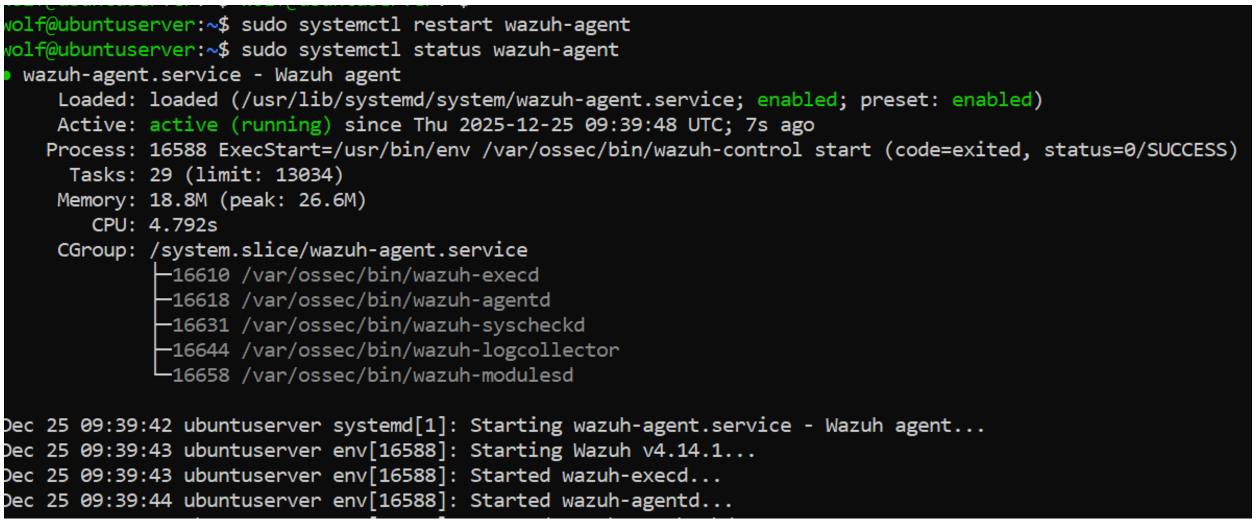
```
(cowrie-env) cowrie@ubuntuserver:~/cowrie$ cowrie start

Join the Cowrie community at: https://www.cowrie.org/slack/

Starting cowrie: [twistd --umask=0022 --pidfile /home/cowrie/cowrie/var/run/cowrie.pid --logger cowrie.python.logfile.logger cowrie]...
/home/cowrie/cowrie/cowrie-env/lib/python3.12/site-packages/twisted/conch/ssh/transport.py:110: CryptographyDeprecationWarning: TripleDES has been moved to cryptography.haz
mat.decrepit.ciphers.algorithms.TripleDES and will be removed from cryptography.hazmat.primitives.ciphers.algorithms in 48.0.0.
  b"3des-cbc": (algorithms.TripleDES, 24, modes.CBC),
/home/cowrie/cowrie/cowrie-env/lib/python3.12/site-packages/twisted/conch/ssh/transport.py:117: CryptographyDeprecationWarning: TripleDES has been moved to cryptography.haz
mat.decrepit.ciphers.algorithms.TripleDES and will be removed from cryptography.hazmat.primitives.ciphers.algorithms in 48.0.0.
  b"3des-ctr": (algorithms.TripleDES, 24, modes.CTR),
(cowrie-env) cowrie@ubuntuserver:~/cowrie$ cat /home/cowrie/cowrie/var/run/cowrie.pid
15548(cowrie-env) cowrie@ubuntuserver:~/cowrie$ ps aux | grep cowrie
```

This screenshot shows the Cowrie honeypot being started and its status verified. The runing status confirms that Cowrie is running correctly and listening on the configured SSH port (2222).



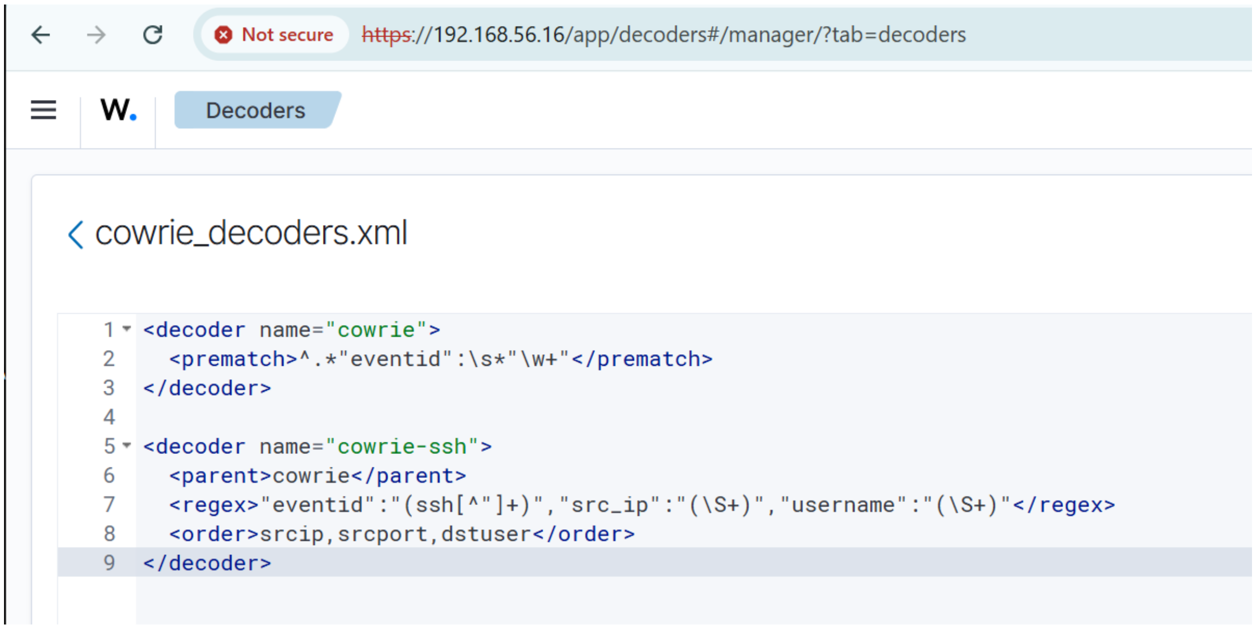
```
wolf@ubuntuserver: ~
 GNU nano 7.2                                /var/ossec/etc/ossec.conf
   <log_format>full_command</log_format>
   <command>last -n 20</command>
   <frequency>360</frequency>
 </localfile>

!-- for cowrie -->

localfile>
 <log_format>json</log_format>
 <location>/home/cowrie/cowrie/var/log/cowrie/cowrie.json</location>
/localfile>
```

This screenshot shows the addition of a local file block in the Wazuh agent configuration file (`/var/ossec/etc/ossec.conf`) on the Cowrie server. The block specifies the location of Cowrie's log files so that the Wazuh agent can monitor them in real time.



```
wolf@ubuntuserver:~$ sudo systemctl restart wazuh-agent
wolf@ubuntuserver:~$ sudo systemctl status wazuh-agent
● wazuh-agent.service - Wazuh agent
     Loaded: loaded (/usr/lib/systemd/system/wazuh-agent.service; enabled; preset: enabled)
     Active: active (running) since Thu 2025-12-25 09:39:48 UTC; 7s ago
    Process: 16588 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0/SUCCESS)
      Tasks: 29 (limit: 13034)
     Memory: 18.8M (peak: 26.6M)
        CPU: 4.792s
     CGroup: /system.slice/wazuh-agent.service
             ├─16610 /var/ossec/bin/wazuh-execd
             ├─16618 /var/ossec/bin/wazuh-agentd
             ├─16631 /var/ossec/bin/wazuh-syscheckd
             ├─16644 /var/ossec/bin/wazuh-logcollector
             └─16658 /var/ossec/bin/wazuh-modulesd

Dec 25 09:39:42 ubuntuserver systemd[1]: Starting wazuh-agent.service - Wazuh agent...
Dec 25 09:39:43 ubuntuserver env[16588]: Starting Wazuh v4.14.1...
Dec 25 09:39:43 ubuntuserver env[16588]: Started wazuh-execd...
Dec 25 09:39:44 ubuntuserver env[16588]: Started wazuh-agentd...
```

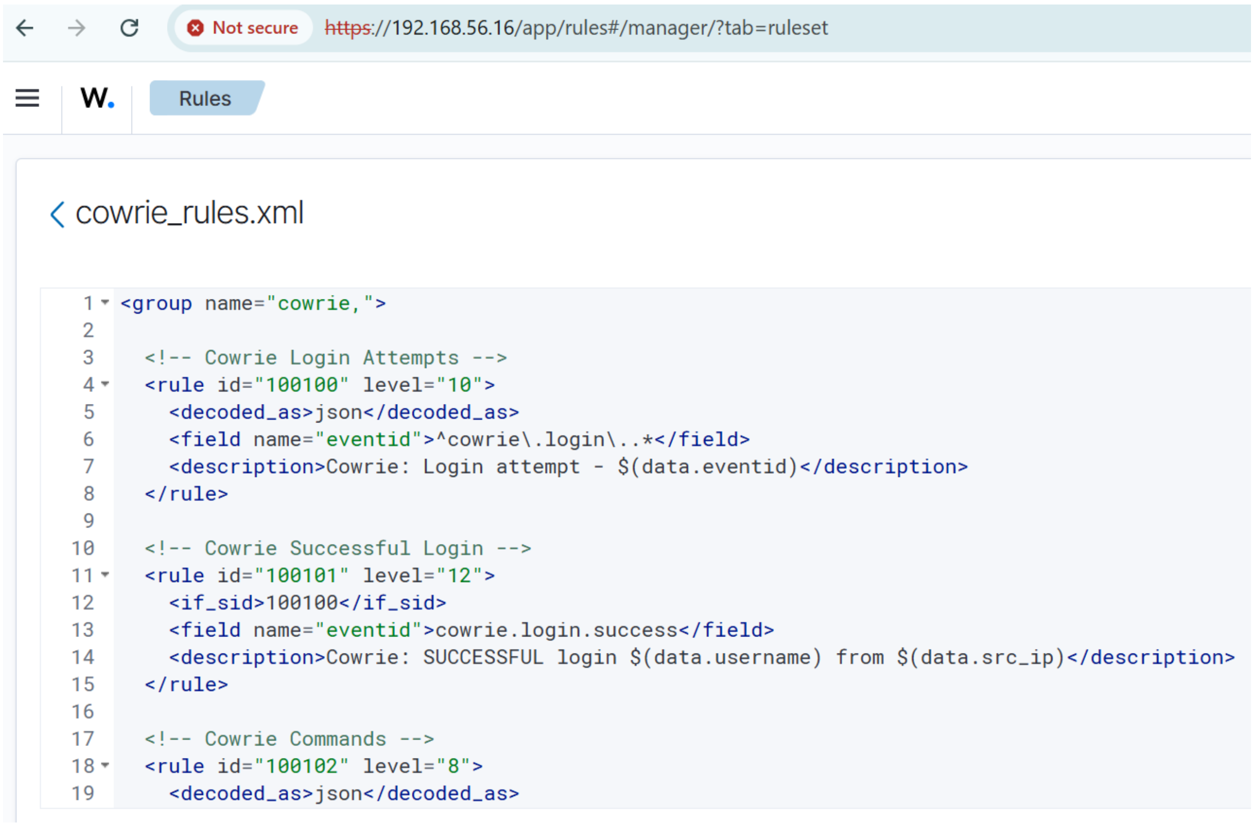
This screenshot shows the Wazuh agent being restarted on the Cowrie server and its status checked. The `active` status confirms that the agent is running correctly and ready to forward Cowrie honeypot logs to the Wazuh manager. Restarting the agent ensures that configuration changes, such as the local file block for Cowrie logs, are applied



cowrie_decoders.xml

```
<decoder name="cowrie">
  <prematch>^.*"eventid":\s*"\w+"</prematch>
</decoder>

<decoder name="cowrie-ssh">
  <parent>cowrie</parent>
  <regex>"eventid":"(ssh[^"]+)","src_ip":"(\S+)","username":"(\S+)"</regex>
  <order>srcip,srcport,dstuser</order>
</decoder>
```

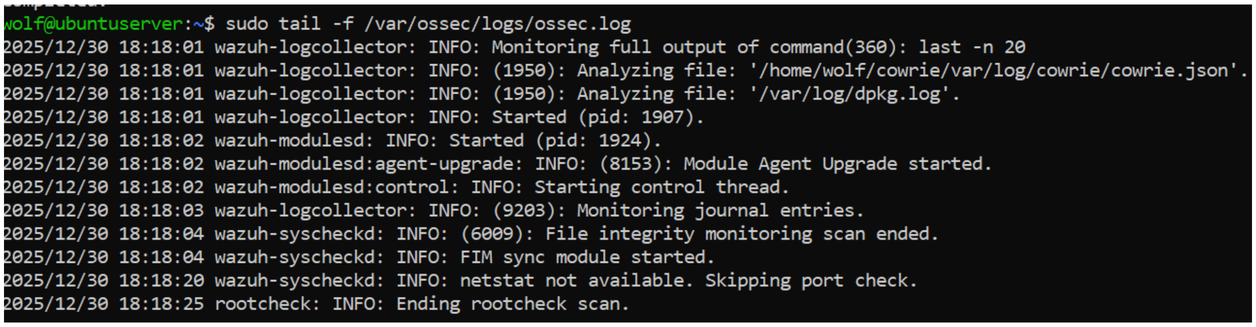
This screenshot shows the creation of custom decoders in the Wazuh web interface for Cowrie honeypot logs. Decoders interpret the raw log data and extract meaningful information, such as attacker IP addresses, login attempts, and commands executed. Proper decoder configuration ensures that Wazuh can correctly parse Cowrie logs,



cowrie_rules.xml

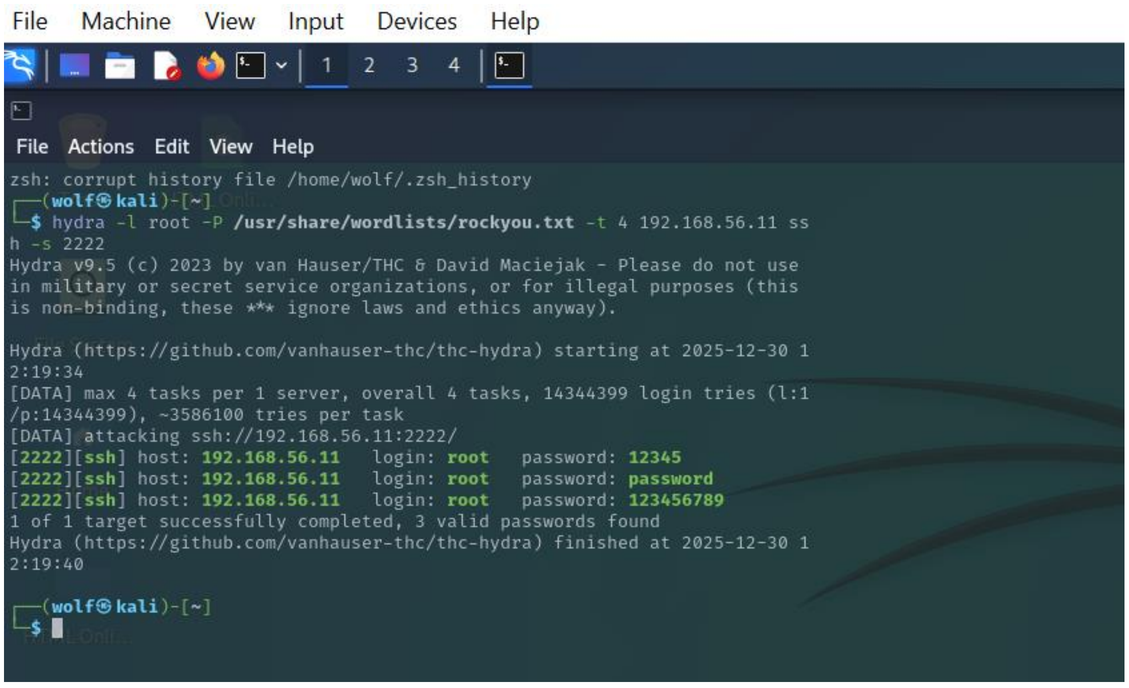
```
<group name="cowrie,">

  <!-- Cowrie Login Attempts -->
  <rule id="100100" level="10">
    <decoded_as>json</decoded_as>
    <field name="eventid">^cowrie\.login\..*</field>
    <description>Cowrie: Login attempt - $(data.eventid)</description>
  </rule>

  <!-- Cowrie Successful Login -->
  <rule id="100101" level="12">
    <if_sid>100100</if_sid>
    <field name="eventid">cowrie.login.success</field>
    <description>Cowrie: SUCCESSFUL login $(data.username) from $(data.src_ip)</description>
  </rule>

  <!-- Cowrie Commands -->
  <rule id="100102" level="8">
    <decoded_as>json</decoded_as>
```

This screenshot shows the creation of custom rules in the Wazuh web interface specifically for Cowrie honeypot logs. These rules define how alerts are triggered when specific patterns or malicious behaviors are detected, such as SSH brute-force attempts or suspicious login activity. Creating custom rules ensures that the SOC receives accurate, actionable alerts from Cowrie events,



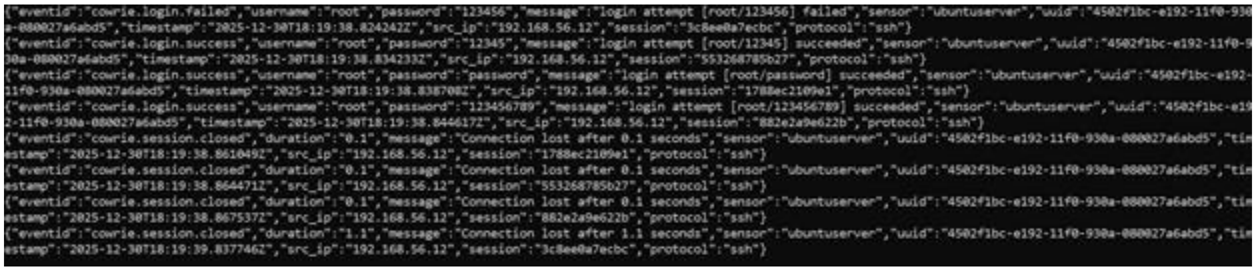
```
wolf@ubuntuserver:~$ sudo tail -f /var/ossec/logs/ossec.log
2025/12/30 18:18:01 wazuh-logcollector: INFO: Monitoring full output of command(360): last -n 20
2025/12/30 18:18:01 wazuh-logcollector: INFO: (1950): Analyzing file: '/home/wolf/cowrie/var/log/cowrie/cowrie.json'.
2025/12/30 18:18:01 wazuh-logcollector: INFO: (1950): Analyzing file: '/var/log/dpkg.log'.
2025/12/30 18:18:01 wazuh-logcollector: INFO: Started (pid: 1907).
2025/12/30 18:18:02 wazuh-modulesd: INFO: Started (pid: 1924).
2025/12/30 18:18:02 wazuh-modulesd:agent-upgrade: INFO: (8153): Module Agent Upgrade started.
2025/12/30 18:18:02 wazuh-modulesd:control: INFO: Starting control thread.
2025/12/30 18:18:03 wazuh-logcollector: INFO: (9203): Monitoring journal entries.
2025/12/30 18:18:04 wazuh-syscheckd: INFO: (6009): File integrity monitoring scan ended.
2025/12/30 18:18:04 wazuh-syscheckd: INFO: FIM sync module started.
2025/12/30 18:18:20 wazuh-syscheckd: INFO: netstat not available. Skipping port check.
2025/12/30 18:18:25 rootcheck: INFO: Ending rootcheck scan.
```

This screenshot shows the Wazuh agent actively monitoring its log file (`ossec.log`) in real time. It confirms that the Cowrie honeypot logs (`cowrie.json`) are being analyzed by Wazuh.



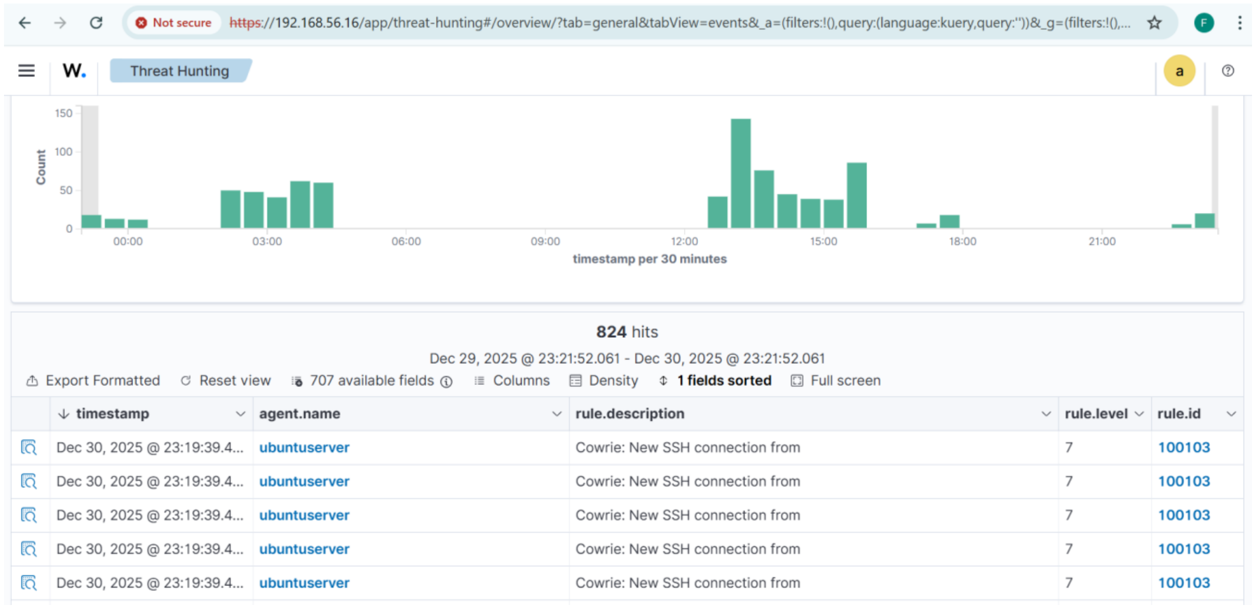
```
zsh: corrupt history file /home/wolf/.zsh_history
┌──(wolf㉿kali)-[~]
└─$ hydra -l root -P /usr/share/wordlists/rockyou.txt -t 4 192.168.56.11 ss
h -s 2222
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use
in military or secret service organizations, or for illegal purposes (this
is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-12-30 1
2:19:34
[DATA] max 4 tasks per 1 server, overall 4 tasks, 14344399 login tries (l:1
/p:14344399), ~3586100 tries per task
[DATA] attacking ssh://192.168.56.11:2222/
[2222][ssh] host: 192.168.56.11   login: root   password: 12345
[2222][ssh] host: 192.168.56.11   login: root   password: password
[2222][ssh] host: 192.168.56.11   login: root   password: 123456789
1 of 1 target successfully completed, 3 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-12-30 1
2:19:40

┌──(wolf㉿kali)-[~]
└─$ 
```

This screenshot shows a simulated SSH brute-force attack launched from the Kali Linux machine using the Hydra tool against the Cowrie honeypot on port `2222`. The attack uses the `rockyou.txt` wordlist to attempt multiple password combinations for the `root` user. This simulation generates high-volume malicious authentication attempts, allowing Cowrie to capture detailed attacker behavior and enabling Wazuh to detect, log, and alert on brute-force activity



```
{"eventid":"cowrie.login.failed","username":"root","password":"123456","message":"login attempt [root/123456] failed","sensor":"ubuntuserver","uuid":"4502f1bc-e192-11f0-930
a-080027a6abd5","timestamp":"2025-12-30T18:19:38.824242Z","src_ip":"192.168.56.12","session":"3c8ee0a7ecbc","protocol":"ssh"}
{"eventid":"cowrie.login.success","username":"root","password":"12345","message":"login attempt [root/12345] succeeded","sensor":"ubuntuserver","uuid":"4502f1bc-e192-11f0-9
30a-080027a6abd5","timestamp":"2025-12-30T18:19:38.834233Z","src_ip":"192.168.56.12","session":"553268785b27","protocol":"ssh"}
{"eventid":"cowrie.login.success","username":"root","password":"password","message":"login attempt [root/password] succeeded","sensor":"ubuntuserver","uuid":"4502f1bc-e192-
11f0-930a-080027a6abd5","timestamp":"2025-12-30T18:19:38.838708Z","src_ip":"192.168.56.12","session":"1788ec2109e1","protocol":"ssh"}
{"eventid":"cowrie.login.success","username":"root","password":"123456789","message":"login attempt [root/123456789] succeeded","sensor":"ubuntuserver","uuid":"4502f1bc-e19
2-11f0-930a-080027a6abd5","timestamp":"2025-12-30T18:19:38.844617Z","src_ip":"192.168.56.12","session":"882e2a9e622b","protocol":"ssh"}
{"eventid":"cowrie.session.closed","duration":"0.1","message":"Connection lost after 0.1 seconds","sensor":"ubuntuserver","uuid":"4502f1bc-e192-11f0-930a-080027a6abd5","tim
estamp":"2025-12-30T18:19:38.861049Z","src_ip":"192.168.56.12","session":"1788ec2109e1","protocol":"ssh"}
{"eventid":"cowrie.session.closed","duration":"0.1","message":"Connection lost after 0.1 seconds","sensor":"ubuntuserver","uuid":"4502f1bc-e192-11f0-930a-080027a6abd5","tim
estamp":"2025-12-30T18:19:38.864471Z","src_ip":"192.168.56.12","session":"553268785b27","protocol":"ssh"}
{"eventid":"cowrie.session.closed","duration":"0.1","message":"Connection lost after 0.1 seconds","sensor":"ubuntuserver","uuid":"4502f1bc-e192-11f0-930a-080027a6abd5","tim
estamp":"2025-12-30T18:19:38.867537Z","src_ip":"192.168.56.12","session":"882e2a9e622b","protocol":"ssh"}
{"eventid":"cowrie.session.closed","duration":"1.1","message":"Connection lost after 1.1 seconds","sensor":"ubuntuserver","uuid":"4502f1bc-e192-11f0-930a-080027a6abd5","tim
estamp":"2025-12-30T18:19:39.837746Z","src_ip":"192.168.56.12","session":"3c8ee0a7ecbc","protocol":"ssh"}
```

This screenshot displays the `cowrie.json` log file, which records detailed events generated by the SSH brute-force attack against the Cowrie honeypot. The logs include attacker IP address, attempted usernames, authentication failures, timestamps, and the targeted SSH port (`2222`). These structured JSON logs confirm that Cowrie successfully captured malicious activity

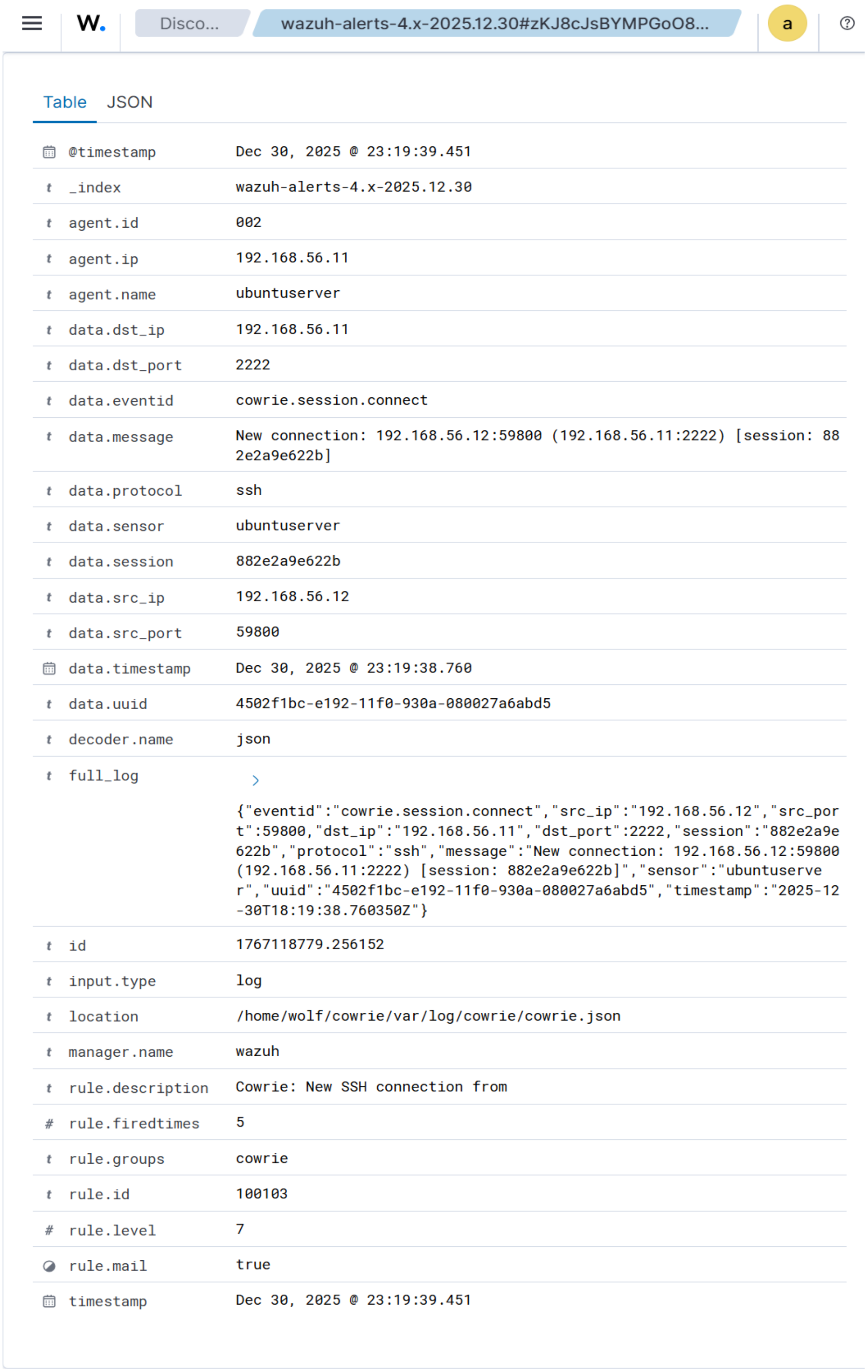


824 hits

Dec 29, 2025 @ 23:21:52.061 - Dec 30, 2025 @ 23:21:52.061

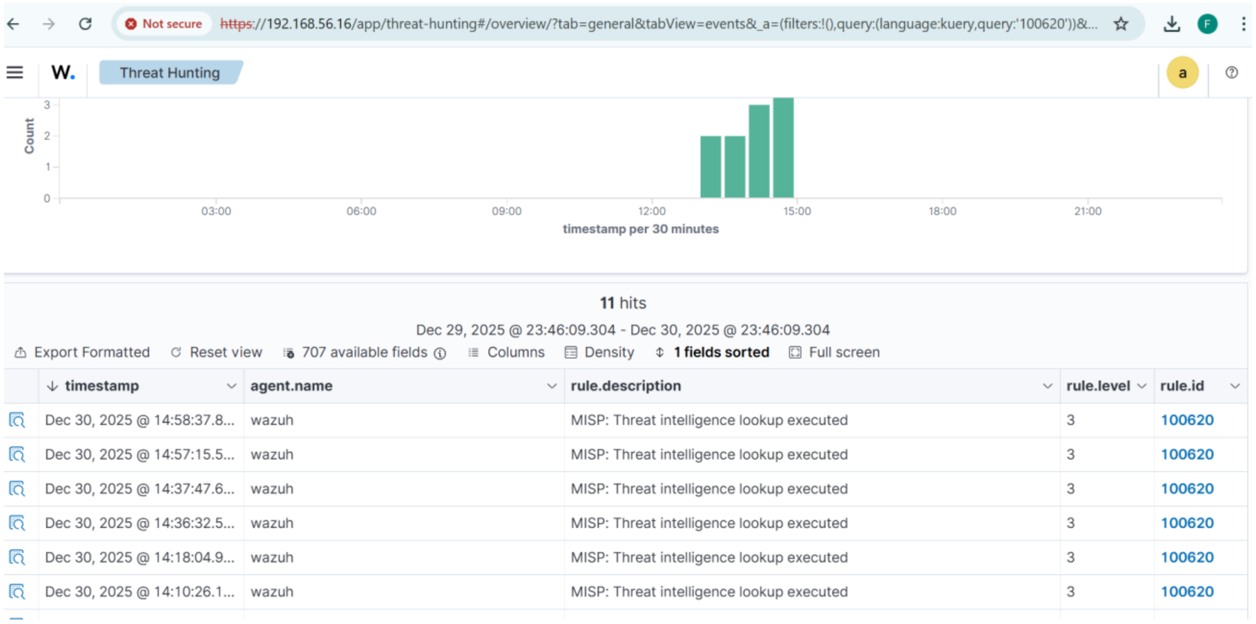
timestamp	agent.name	rule.description	rule.level	rule.id
Dec 30, 2025 @ 23:19:39.4...	ubuntuserver	Cowrie: New SSH connection from	7	100103
Dec 30, 2025 @ 23:19:39.4...	ubuntuserver	Cowrie: New SSH connection from	7	100103
Dec 30, 2025 @ 23:19:39.4...	ubuntuserver	Cowrie: New SSH connection from	7	100103
Dec 30, 2025 @ 23:19:39.4...	ubuntuserver	Cowrie: New SSH connection from	7	100103
Dec 30, 2025 @ 23:19:39.4...	ubuntuserver	Cowrie: New SSH connection from	7	100103

This screenshot displays multiple Cowrie honeypot events visible on the Wazuh dashboard, captured from the recent SSH attack simulations. Each event includes details such as attacker IPs, attempted usernames, timestamps, and the matched rules triggered by the attack. This confirms that the Wazuh manager successfully ingests Cowrie logs, applies the custom decoders and rules, and generates actionable alerts



@timestamp	Dec 30, 2025 @ 23:19:39.451
_index	wazuh-alerts-4.x-2025.12.30
agent.id	002
agent.ip	192.168.56.11
agent.name	ubuntuserver
data.dst_ip	192.168.56.11
data.dst_port	2222
data.eventid	cowrie.session.connect
data.message	New connection: 192.168.56.12:59800 (192.168.56.11:2222) [session: 882e2a9e622b]
data.protocol	ssh
data.sensor	ubuntuserver
data.session	882e2a9e622b
data.src_ip	192.168.56.12
data.src_port	59800
data.timestamp	Dec 30, 2025 @ 23:19:38.760
data.uuid	4502f1bc-e192-11f0-930a-080027a6abd5
decoder.name	json
full_log	{"eventid":"cowrie.session.connect","src_ip":"192.168.56.12","src_port":59800,"dst_ip":"192.168.56.11","dst_port":2222,"session":"882e2a9e622b","protocol":"ssh","message":"New connection: 192.168.56.12:59800 (192.168.56.11:2222) [session: 882e2a9e622b]","sensor":"ubuntuserver","uuid":"4502f1bc-e192-11f0-930a-080027a6abd5","timestamp":"2025-12-30T18:19:38.760350Z"}
id	1767118779.256152
input.type	log
location	/home/wolf/cowrie/var/log/cowrie/cowrie.json
manager.name	wazuh
rule.description	Cowrie: New SSH connection from
rule.firedtimes	5
rule.groups	cowrie
rule.id	100103
rule.level	7
rule.mail	true
timestamp	Dec 30, 2025 @ 23:19:39.451

This screenshot presents a detailed view of a single Cowrie-related security event within the Wazuh dashboard. It includes comprehensive metadata such as the source IP address of the attacker, destination IP of the honeypot, agent name, event timestamp, rule ID, rule description, and alert severity level. The presence of this detailed alert confirms that Wazuh successfully decoded the Cowrie logs, applied custom detection rules, and produced enriched alerts.

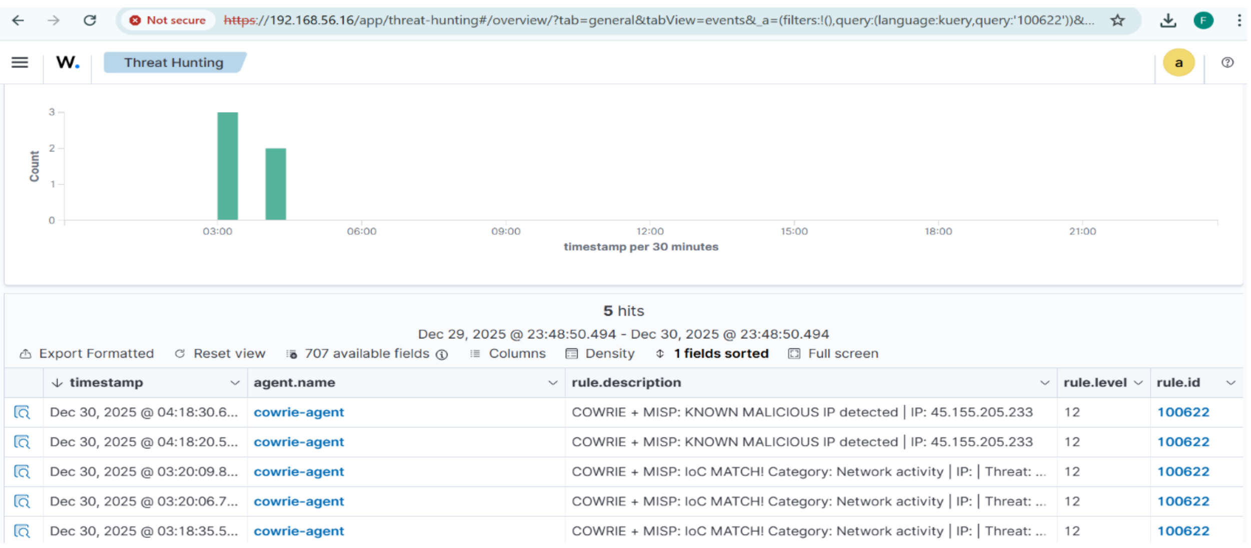


11 hits

Dec 29, 2025 @ 23:46:09.304 - Dec 30, 2025 @ 23:46:09.304

timestamp	agent.name	rule.description	rule.level	rule.id
Dec 30, 2025 @ 14:58:37.8...	wazuh	MISP: Threat intelligence lookup executed	3	100620
Dec 30, 2025 @ 14:57:15.5...	wazuh	MISP: Threat intelligence lookup executed	3	100620
Dec 30, 2025 @ 14:37:47.6...	wazuh	MISP: Threat intelligence lookup executed	3	100620
Dec 30, 2025 @ 14:36:32.5...	wazuh	MISP: Threat intelligence lookup executed	3	100620
Dec 30, 2025 @ 14:18:04.9...	wazuh	MISP: Threat intelligence lookup executed	3	100620
Dec 30, 2025 @ 14:10:26.1...	wazuh	MISP: Threat intelligence lookup executed	3	100620

This screenshot displays an event in the Wazuh dashboard where a **MISP threat intelligence lookup** was executed. It indicates that Wazuh successfully queried the MISP server for indicators related to incoming events (e.g., attacker IPs from Cowrie logs). When a match occurs,

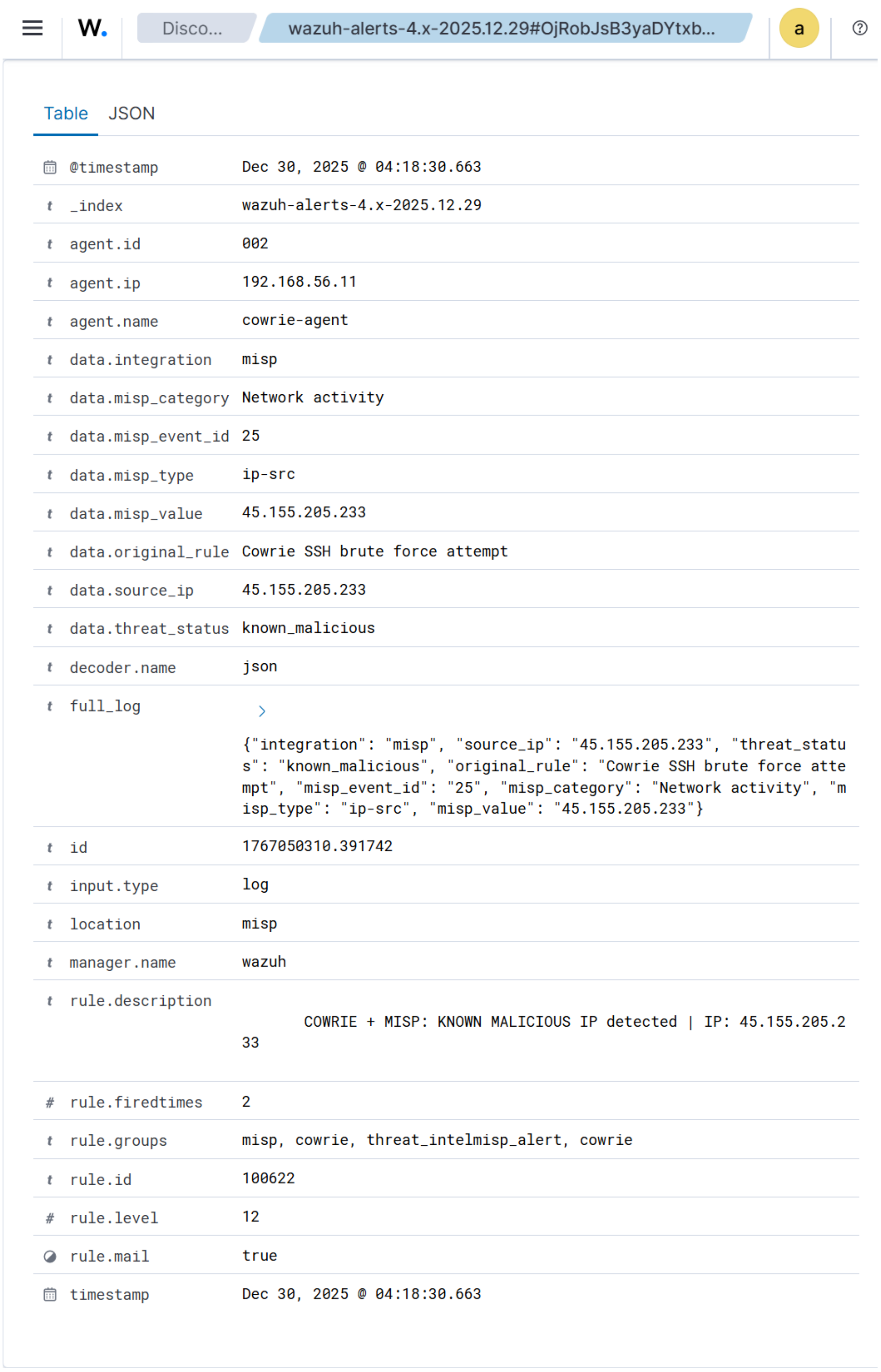


5 hits

Dec 29, 2025 @ 23:48:50.494 - Dec 30, 2025 @ 23:48:50.494

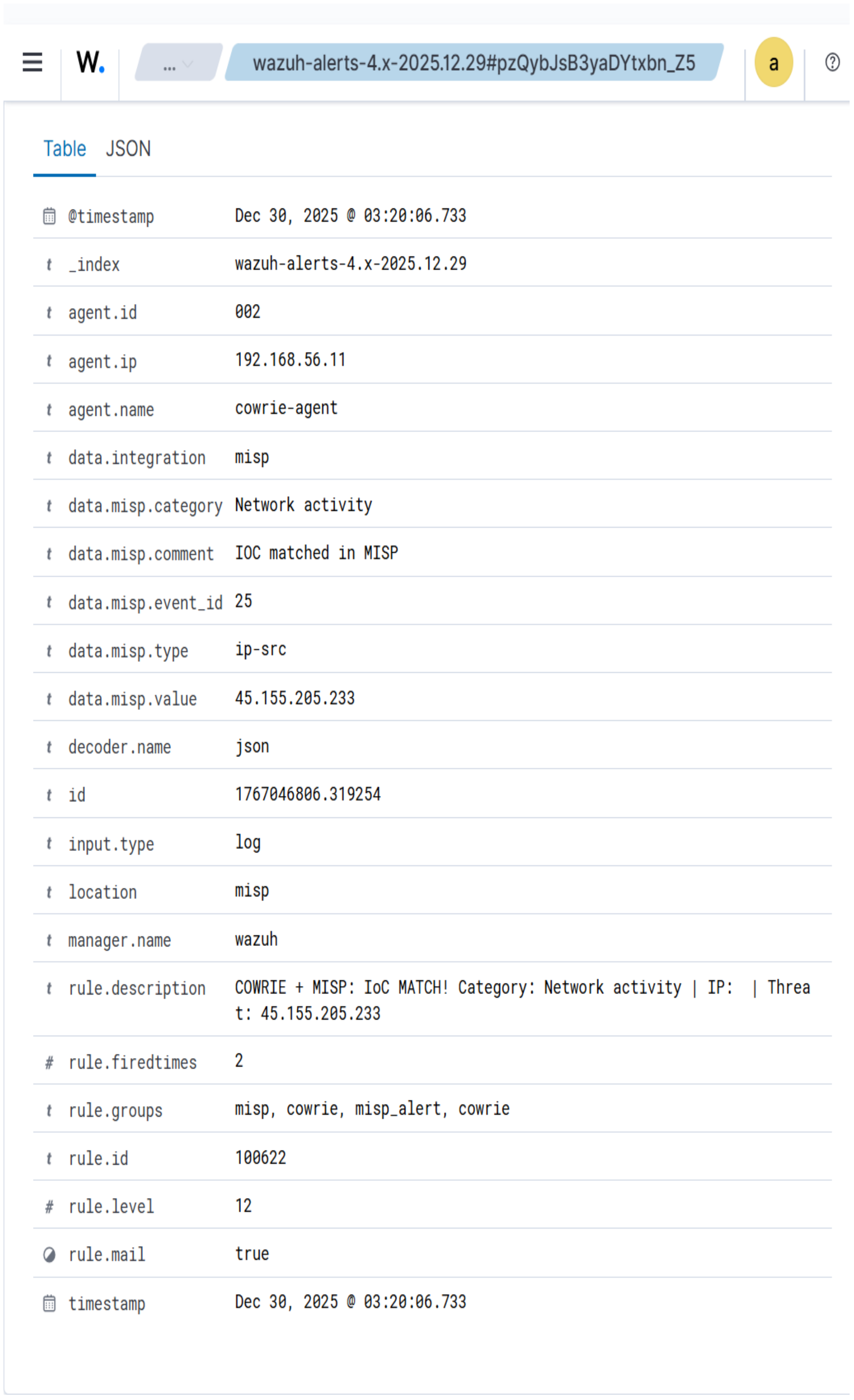
timestamp	agent.name	rule.description	rule.level	rule.id
Dec 30, 2025 @ 04:18:30.6...	cowrie-agent	COWRIE + MISP: KNOWN MALICIOUS IP detected | IP: 45.155.205.233	12	100622
Dec 30, 2025 @ 04:18:20.5...	cowrie-agent	COWRIE + MISP: KNOWN MALICIOUS IP detected | IP: 45.155.205.233	12	100622
Dec 30, 2025 @ 03:20:09.8...	cowrie-agent	COWRIE + MISP: IoC MATCH! Category: Network activity | IP: | Threat: ...	12	100622
Dec 30, 2025 @ 03:20:06.7...	cowrie-agent	COWRIE + MISP: IoC MATCH! Category: Network activity | IP: | Threat: ...	12	100622
Dec 30, 2025 @ 03:18:35.5...	cowrie-agent	COWRIE + MISP: IoC MATCH! Category: Network activity | IP: | Threat: ...	12	100622

This screenshot displays multiple events in the Wazuh dashboard where incoming Cowrie honeypot logs were correlated with MISP threat intelligence indicators. Some alerts show **known malicious IPs detected**, while others indicate **IOC matches** in the network category. This demonstrates that Wazuh is not only ingesting honeypot logs but also enriching them with external threat intelligence from MISP.



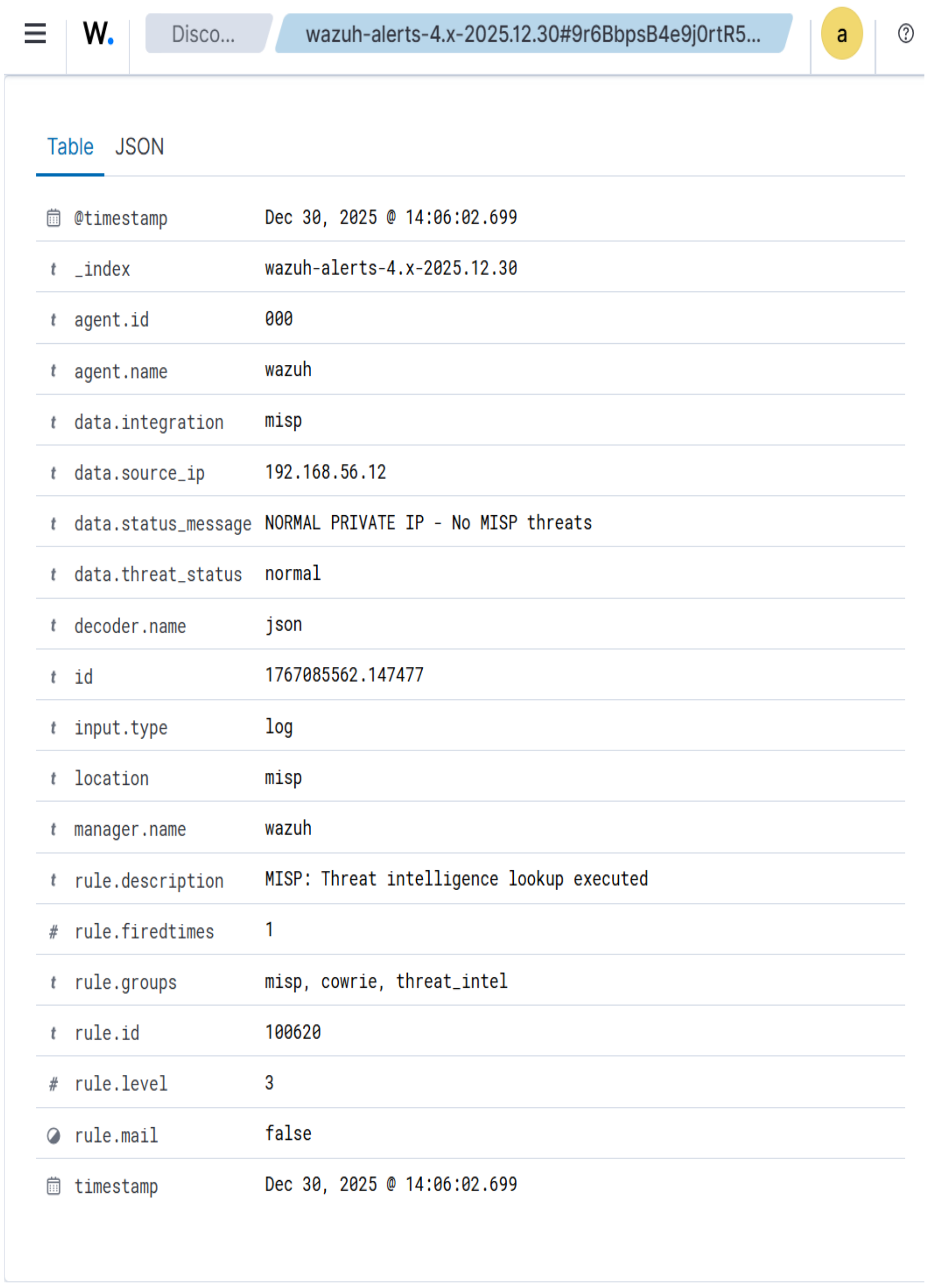
@timestamp	Dec 30, 2025 @ 04:18:30.663
_index	wazuh-alerts-4.x-2025.12.29
agent.id	002
agent.ip	192.168.56.11
agent.name	cowrie-agent
data.integration	misp
data.misp_category	Network activity
data.misp_event_id	25
data.misp_type	ip-src
data.misp_value	45.155.205.233
data.original_rule	Cowrie SSH brute force attempt
data.source_ip	45.155.205.233
data.threat_status	known_malicious
decoder.name	json
full_log	{"integration": "misp", "source_ip": "45.155.205.233", "threat_status": "known_malicious", "original_rule": "Cowrie SSH brute force attempt", "misp_event_id": "25", "misp_category": "Network activity", "misp_type": "ip-src", "misp_value": "45.155.205.233"}
id	1767050310.391742
input.type	log
location	misp
manager.name	wazuh
rule.description	COWRIE + MISP: KNOWN MALICIOUS IP detected | IP: 45.155.205.233
rule.firedtimes	2
rule.groups	misp, cowrie, threat_intelmisp_alert, cowrie
rule.id	100622
rule.level	12
rule.mail	true
timestamp	Dec 30, 2025 @ 04:18:30.663

This screenshot presents a detailed view of a Cowrie honeypot event in the Wazuh dashboard that was matched with a known malicious IP from MISP threat intelligence. The alert includes comprehensive metadata such as the timestamp, rule ID, event severity, threat status, source IP, and MISP category. This demonstrates that Wazuh successfully correlated honeypot activity with external threat intelligence,



@timestamp	Dec 30, 2025 @ 03:20:06.733
_index	wazuh-alerts-4.x-2025.12.29
agent.id	002
agent.ip	192.168.56.11
agent.name	cowrie-agent
data.integration	misp
data.misp.category	Network activity
data.misp.comment	IOC matched in MISP
data.misp.event_id	25
data.misp.type	ip-src
data.misp.value	45.155.205.233
decoder.name	json
id	1767046806.319254
input.type	log
location	misp
manager.name	wazuh
rule.description	COWRIE + MISP: IoC MATCH! Category: Network activity | IP: | Threat: 45.155.205.233
rule.firedtimes	2
rule.groups	misp, cowrie, misp_alert, cowrie
rule.id	100622
rule.level	12
rule.mail	true
timestamp	Dec 30, 2025 @ 03:20:06.733

This screenshot displays a detailed Wazuh alert where a Cowrie honeypot event was matched against a MISP IOC (Indicator of Compromise). The alert contains full details, including timestamp, source and destination IPs, rule ID, event severity, agent name, threat status, and MISP category. This confirms that Wazuh is successfully correlating live honeypot data with external threat intelligence, allowing SOC analysts to detect and prioritize known malicious activity efficiently. The IOC match provides context-rich alerts for faster incident investigation and response



```
@timestamp          Dec 30, 2025 @ 14:06:02.699
_index              wazuh-alerts-4.x-2025.12.30
agent.id            000
agent.name          wazuh
data.integration    misp
data.source_ip      192.168.56.12
data.status_message NORMAL PRIVATE IP - No MISP threats
data.threat_status  normal
decoder.name        json
id                  1767085562.147477
input.type          log
location            misp
manager.name        wazuh
rule.description    MISP: Threat intelligence lookup executed
rule.firedtimes     1
rule.groups         misp, cowrie, threat_intel
rule.id             100620
rule.level          3
rule.mail           false
timestamp           Dec 30, 2025 @ 14:06:02.699
```

This screenshot displays a detailed Wazuh alert for a normal private IP that was captured by the Cowrie honeypot but **did not match any MISP threat intelligence indicators**. The alert includes details such as timestamp, agent name, source and destination IPs, and rule ID, showing that Wazuh can log benign activity without generating false-positive threat alerts.

During the SSH brute-force attack against the Cowrie honeypot, attacker IP `45.155.205.233` was captured and forwarded to Wazuh. Wazuh queried the MISP platform and successfully matched the IP against the Feodo Tracker threat intelligence feed. This correlation elevated the alert severity, demonstrating how combining honeypot telemetry with external threat intelligence reduces false positives and increases SOC confidence

Challenges faced & Solution implemented

➢ Cowrie Logs Not Appearing in Wazuh Dashboard

Challenge:

Although Cowrie was successfully installed and generating logs (`cowrie.json`), no Cowrie related alerts were initially visible in the Wazuh dashboard. This caused confusion as the honeypot appeared to be working correctly, but Wazuh was not detecting the activity.

Root Cause:

The Wazuh agent was not properly configured to monitor the Cowrie log file path. Additionally, initial attempts involved unnecessary custom decoders and rules, which conflicted with Wazuh's default log parsing behavior.

Solution:

- Verified Cowrie log generation locally.
- Corrected the `ossec.conf` configuration to explicitly monitor the `cowrie.json` file.
- Removed incorrect and unnecessary custom decoders and rules.
- Restarted the Wazuh agent and manager to apply changes.

Outcome:

Cowrie SSH attacks and session events became visible in the Wazuh dashboard, confirming successful log collection.

➢ Docker Installation and Configuration Issues

Challenge:

During the initial setup of MISP using Docker, multiple errors were encountered while following installation steps from unofficial and random online sources. These issues included container startup failures, missing environment variables, and dependency conflicts, which prevented MISP services from running correctly.

Root Cause:

The Docker setup instructions used initially were outdated and not aligned with the current MISP Docker images. Unofficial guides lacked proper version compatibility and did not reflect best practices recommended by the MISP maintainers.

Solution:

- Abandoned unofficial installation guides.
- Reinstalled Docker using the **official Docker documentation**.
- Followed the official MISP Docker repository documentation step-by-step.
- Verified container health using `docker ps` and service logs.
- Ensured correct environment variables and network settings were applied.

Outcome:

MISP was successfully deployed and stabilized using Docker, with all required services running correctly. This enabled secure API access and successful integration with Wazuh.