Integrating Wazuh with AbusIPDB

Created By: Amir Raza

Follow Me: www.linkedin.com/in/amirsoc

Integrating Wazuh with AbusIPDB

Goal of This Guide

This document walks you step-by-step through integrating Wazuh with AbuseIPDB, a real-time IP threat intelligence service. The goal is to:

Detect suspicious SSH login attempts (both failed and successful).

Automatically send the source IP to AbuseIPDB for scoring.

If AbuseIPDB confirms it as malicious, raise a security alert in Wazuh.

Why Do This?

This setup is ideal if you want to:

Automatically validate incoming IPs against a global threat database.

Spot SSH brute-force or credential stuffing attempts.

Leverage community-sourced intelligence for automated threat detection.

Take action based on abuse confidence scores (e.g., alert or block).

What is AbuseIPDB?

AbuseIPDB (Abuse IP Database) is an open and collaborative threat intelligence platform designed to track and report malicious IP addresses across the internet. It allows users and systems to report IPs involved in suspicious activities and provides actionable data to security professionals and automated defense systems.

AbuseIPDB focuses on identifying IP addresses engaged in various types of unauthorized or harmful behavior, including:

Brute-force login attempts
Spamming and email abuse
Unauthorized port scanning
Exploit attempts targeting known vulnerabilities
Other forms of suspicious or malicious network activity

Through its RESTful API, AbuseIPDB provides valuable information for each IP address, such as:

Abuse Confidence Score (0–100): Indicates how likely the IP is involved in abusive behavior
ISP and Hosting Provider Information
Associated Domain Name(s)
Total Number of Reports Filed Against the IP
Last Reported Date

This data helps system administrators, cybersecurity teams, and intrusion detection systems (like Wazuh) to identify threats early, block abusive IPs, and strengthen network defense strategies.

Real-World Example

Imagine someone tries to connect to your server using SSH (remote login), but they fail to log in. The attack comes from the IP address: 45.159.112.120.

Here's what happens step by step:

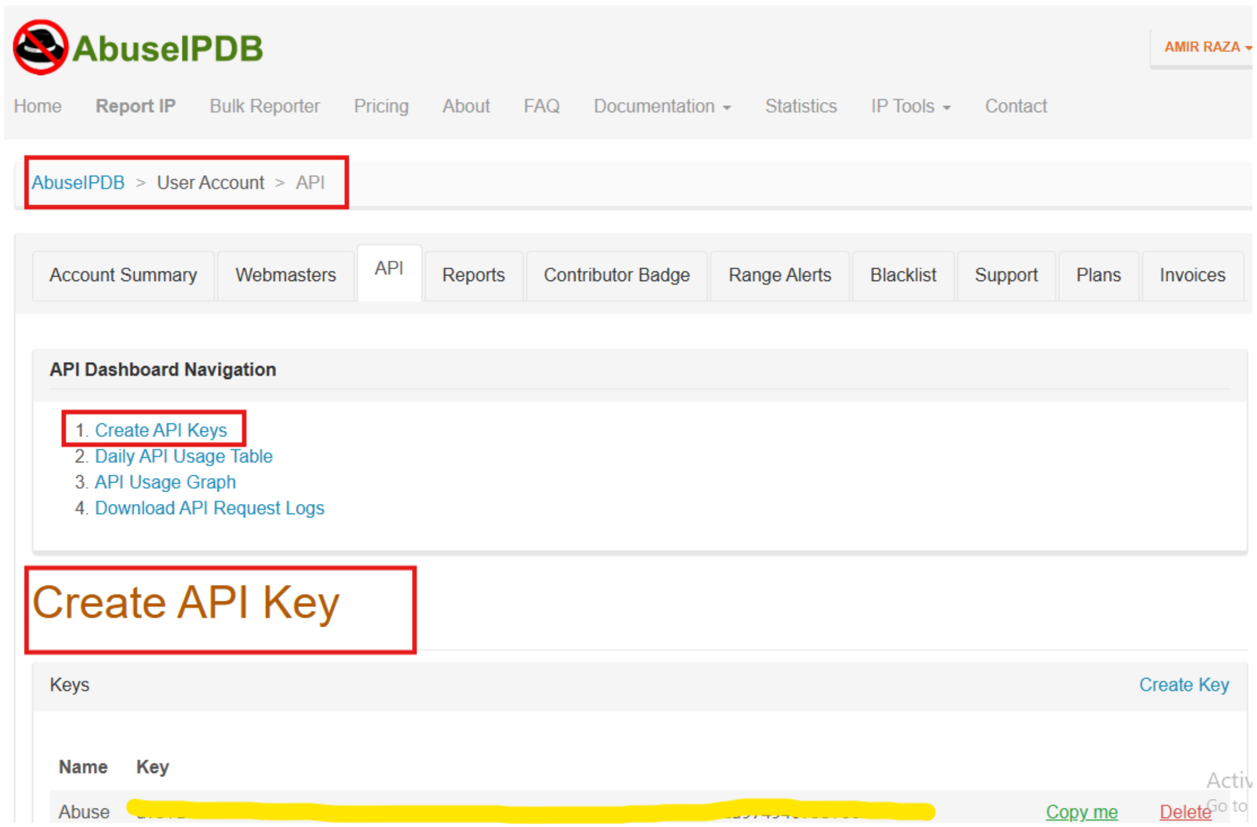
Wazuh notices this failed login attempt and logs it.
Wazuh sends this IP address (45.159.112.120) to AbuseIPDB to check if this IP is bad.
AbuseIPDB replies and says:
"Yes, this IP is dangerous. It has a 90% abuse score, which means it has done bad things before (like attacks or spam)."
Now, Wazuh creates a high-priority alert because it knows this IP is a real threat.
You or your system can now take action, such as:
Blocking the IP address
Sending a warning
Notifying the system admin

Step-by-Step Integration Instructions

API Key Generation for AbuseIPDB

To enable integration between Wazuh and AbuseIPDB for reporting malicious IP addresses, an API key is required. Follow the steps below to generate your key:

Create an Account
 Visit https://www.abuseipdb.com and sign up using a valid email address.
Verify Your Email
 Check your inbox and click the verification link sent by AbuseIPDB to activate your account.
Log In
 Access your account using your registered credentials.
Access the API Section
 Navigate to your profile and select the "API" option or directly visit: https://www.abuseipdb.com/account/api.
Generate and Copy the API Key
 Click "Reveal Key" to generate your personal API key. Copy and securely store it for use in the Wazuh integration script.



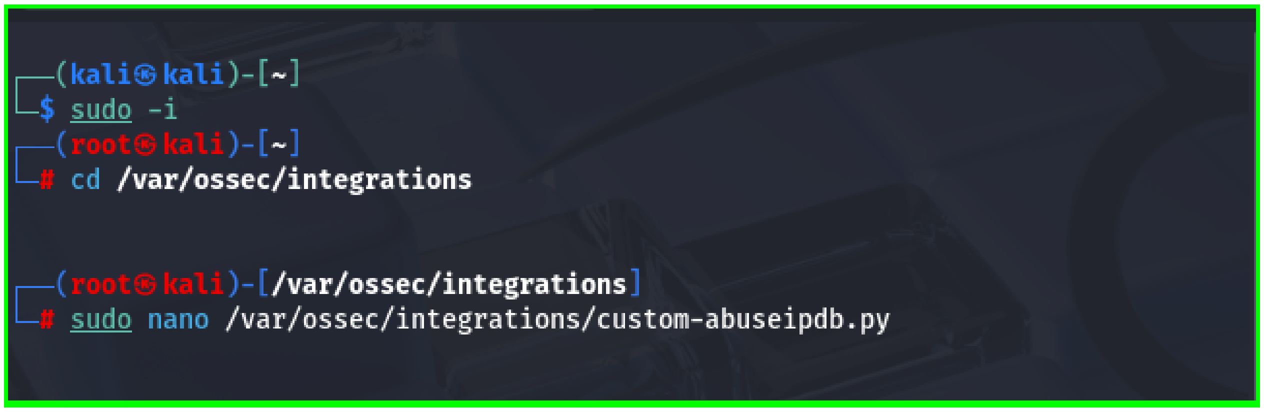
Name	Key		
Abuse	[redacted]	Copy me	Delete

Replace < your AbuseIPDB API Key here> with your real API key.

Create the Custom Script on the Manager Side

We will place the script inside the Wazuh `integrations` directory.

```
cd /var/ossec/integrations
```



```
┌──(kali㉿kali)-[~]
└─$ sudo -i
┌──(root㉿kali)-[~]
└─# cd /var/ossec/integrations


┌──(root㉿kali)-[/var/ossec/integrations]
└─# sudo nano /var/ossec/integrations/custom-abuseipdb.py
```

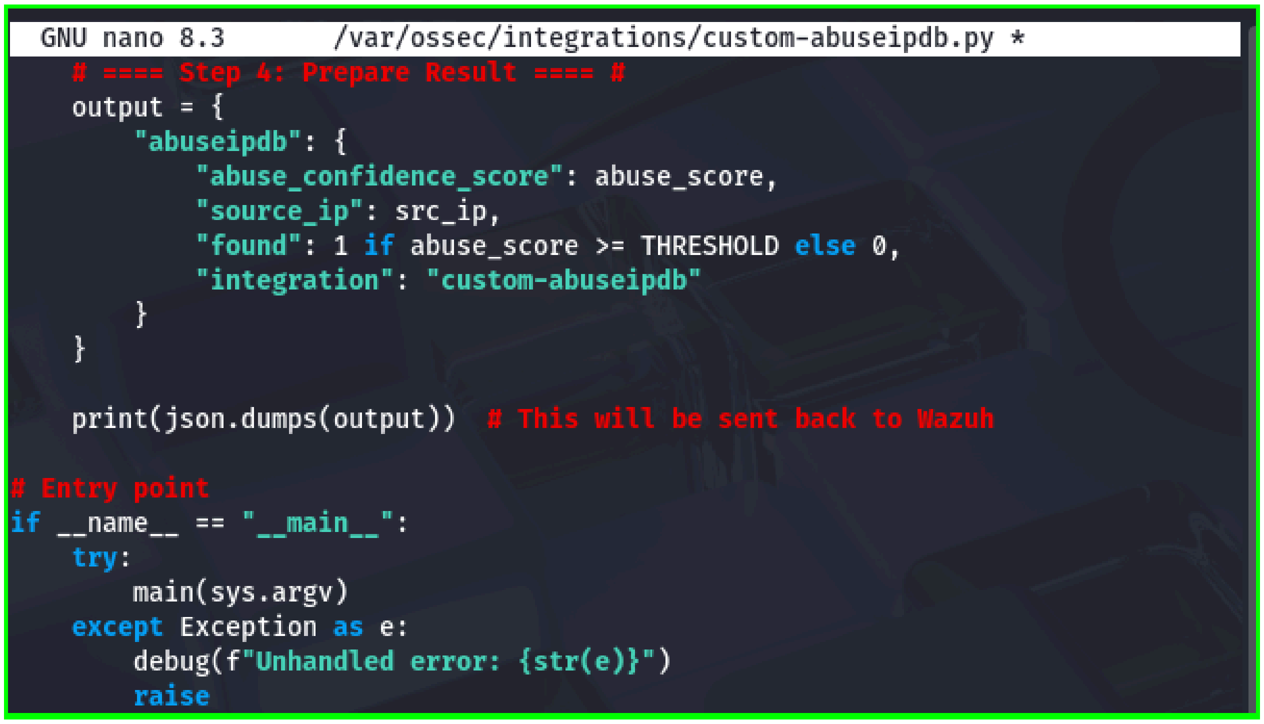
sudo nano /var/ossec/integrations/custom-abuseipdb.py

Add this:

```
import json

import sys

import time

import os

from socket import socket, AF_UNIX, SOCK_DGRAM

try:

    import requests

    from requests.auth import HTTPBasicAuth

except Exception as e:

    print("No module 'requests' found. Install: pip install requests")

    sys.exit(1)

# Global vars

debug_enabled = False

pwd = os.path.dirname(os.path.dirname(os.path.realpath(__file__)))

json_alert = {}

now = time.strftime("%a %b %d %H:%M:%S %Z %Y")

# Set paths
```

```
log_file = '{0}/logs/integrations.log'.format(pwd)

socket_addr = '{0}/queue/sockets/queue'.format(pwd)

def main(args):

    debug("# Starting")

    # Read args

    alert_file_location = args[1]

    apikey = args[2]

    debug("# API Key")

    debug(apikey)

    debug("# File location")

    debug(alert_file_location)

     # Load alert. Parse JSON object.

    with open(alert_file_location) as alert_file:

        json_alert = json.load(alert_file)

    debug("# Processing alert")

    debug(json_alert)

    # Request AbuseIPDB info

    msg = request_abuseipdb_info(json_alert,apikey)

 # If positive match, send event to Wazuh Manager

    if msg:

        send_event(msg, json_alert["agent"])

def debug(msg):

    if debug_enabled:

        msg = "{0}: {1}\n".format(now, msg)


        print(msg)
        f = open(log_file,"a")

        f.write(msg)

        f.close()

def collect(data):

  abuse_confidence_score = data['abuseConfidenceScore']

  country_code = data['countryCode']

  usage_type = data['usageType']

  isp = data['isp']

  domain = data['domain']

  total_reports = data['totalReports']

  last_reported_at = data['lastReportedAt']

  return abuse_confidence_score, country_code, usage_type, isp, domain,
total_reports, last_reported_at

def in_database(data, srcip):

  result = data['totalReports']

  if result == 0:

    return False

  return True

def query_api(srcip, apikey):

  params = {'maxAgeInDays': '90', 'ipAddress': srcip,}

  headers = {

  "Accept-Encoding": "gzip, deflate",

  'Accept': 'application/json',

  "Key": apikey

  }

  response =
requests.get('https://api.abuseipdb.com/api/v2/check',params=params,
headers=headers)
```

```
  if response.status_code == 200:

      json_response = response.json()

      data = json_response["data"]

      return data

  else:

      alert_output = {}

      alert_output["abuseipdb"] = {}

      alert_output["integration"] = "custom-abuseipdb"

      json_response = response.json()

      debug("# Error: The AbuseIPDB encountered an error")

      alert_output["abuseipdb"]["error"] = response.status_code

      alert_output["abuseipdb"]["description"] =
json_response["errors"][0]["detail"]

      send_event(alert_output)

      exit(0)

def request_abuseipdb_info(alert, apikey):

    alert_output = {}

    # If there is no source ip address present in the alert. Exit.

    if not "srcip" in alert["data"]:

       return(0)

 # Request info using AbuseIPDB API

    data = query_api(alert["data"]["srcip"], apikey)


    # Create alert

    alert_output["abuseipdb"] = {}

    alert_output["integration"] = "custom-abuseipdb"

    alert_output["abuseipdb"]["found"] = 0
    alert_output["abuseipdb"]["source"] = {}

    alert_output["abuseipdb"]["source"]["alert_id"] = alert["id"]

    alert_output["abuseipdb"]["source"]["rule"] = alert["rule"]["id"]

    alert_output["abuseipdb"]["source"]["description"] =
alert["rule"]["description"]

    alert_output["abuseipdb"]["source"]["full_log"] = alert["full_log"]

    alert_output["abuseipdb"]["source"]["srcip"] = alert["data"]["srcip"]

    srcip = alert["data"]["srcip"]

    # Check if AbuseIPDB has any info about the srcip

    if in_database(data, srcip):

      alert_output["abuseipdb"]["found"] = 1

# Info about the IP found in AbuseIPDB

    if alert_output["abuseipdb"]["found"] == 1:

        abuse_confidence_score, country_code, usage_type, isp, domain,
total_reports, last_reported_at = collect(data)

 # Populate JSON Output object with AbuseIPDB request

        alert_output["abuseipdb"]["abuse_confidence_score"] =
abuse_confidence_score

        alert_output["abuseipdb"]["country_code"] = country_code

        alert_output["abuseipdb"]["usage_type"] = usage_type

        alert_output["abuseipdb"]["isp"] = isp

        alert_output["abuseipdb"]["domain"] = domain

        alert_output["abuseipdb"]["total_reports"] = total_reports

        alert_output["abuseipdb"]["last_reported_at"] = last_reported_at

    debug(alert_output)

    return(alert_output)

def send_event(msg, agent = None):

    if not agent or agent["id"] == "000":
        string = '1:abuseipdb:{0}'.format(json.dumps(msg))
    else:
        string = '1:[{0}] ({1}) {2}->abuseipdb:{3}'.format(agent["id"],
agent["name"], agent["ip"] if "ip" in agent else "any", json.dumps(msg))


    debug(string)
    sock = socket(AF_UNIX, SOCK_DGRAM)
    sock.connect(socket_addr)
    sock.send(string.encode())
 sock.close()
if __name__ == "__main__":
    try:
        # Read arguments
        bad_arguments = False
        if len(sys.argv) >= 4:
            msg = '{0} {1} {2} {3} {4}'.format(now, sys.argv[1], sys.argv[2],
sys.argv[3], sys.argv[4] if len(sys.argv) > 4 else '')
            debug_enabled = (len(sys.argv) > 4 and sys.argv[4] == 'debug')
        else:
            msg = '{0} Wrong arguments'.format(now)
            bad_arguments = True
 # Logging the call
        f = open(log_file, 'a')
        f.write(msg +'\n')
        f.close()
        if bad_arguments:
            debug("# Exiting: Bad arguments.")
            sys.exit(1)
      # Main function
        main(sys.argv)
     except Exception as e:
        debug(str(e))
        raise
```



```
  GNU nano 8.3        /var/ossec/integrations/custom-abuseipdb.py *
    # ==== Step 4: Prepare Result ==== #
    output = {
        "abuseipdb": {
            "abuse_confidence_score": abuse_score,
            "source_ip": src_ip,
            "found": 1 if abuse_score >= THRESHOLD else 0,
            "integration": "custom-abuseipdb"
        }
    }

    print(json.dumps(output))  # This will be sent back to Wazuh

# Entry point
if __name__ == "__main__":
    try:
        main(sys.argv)
    except Exception as e:
        debug(f"Unhandled error: {str(e)}")
        raise
```

Purpose of the Script

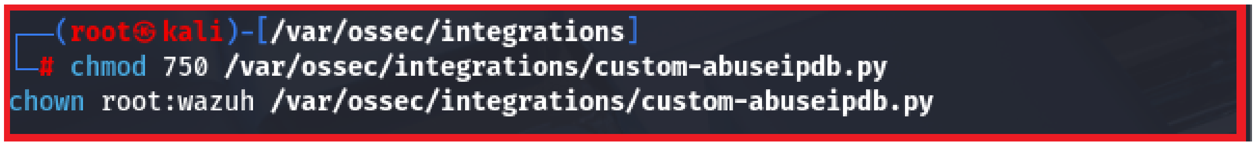
The purpose of the `custom-abuseipdb.py` script is to enhance Wazuh's detection capabilities by integrating it with AbuseIPDB, a threat intelligence platform. When Wazuh generates an alert involving an IP address (e.g., after a failed login attempt), this script:

Submits the suspicious IP to AbuseIPDB.
Retrieves the abuse confidence score and relevant threat data.
Adds this external intelligence to Wazuh alerts.
Helps identify malicious IPs and improves response decisions.
Supports automation for blocking or alerting based on abuse scores.

This integration strengthens Wazuh's ability to detect, classify, and respond to external threats in real-time.

Set Proper Permissions:

```
chmod 750 /var/ossec/integrations/custom-abuseipdb.py
chown root:wazuh /var/ossec/integrations/custom-abuseipdb.py
```



```
┌──(root㉿kali)-[/var/ossec/integrations]
└─# chmod 750 /var/ossec/integrations/custom-abuseipdb.py
chown root:wazuh /var/ossec/integrations/custom-abuseipdb.py
```

`chmod 750`: Makes the script executable by owner, readable by group (Wazuh).

`chown root:wazuh`: Ensures correct ownership to avoid execution issues.

Configure the Integration in Wazuh Manager (`ossec.conf`)

Edit the Wazuh Manager config:

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

Add This Block inside `<ossec_config>`:

```
<!-- AbuseIPDB Integration -->
<integration>
  <name>custom-abuseipdb.py</name>
  <hook_url>https://api.abuseipdb.com/api/v2/check</hook_url>
  <api_key>YOUR_ABUSEIPDB_API_KEY</api_key>
  <rule_id>100002,100003</rule_id>
  <alert_format>json</alert_format>
</integration>
```

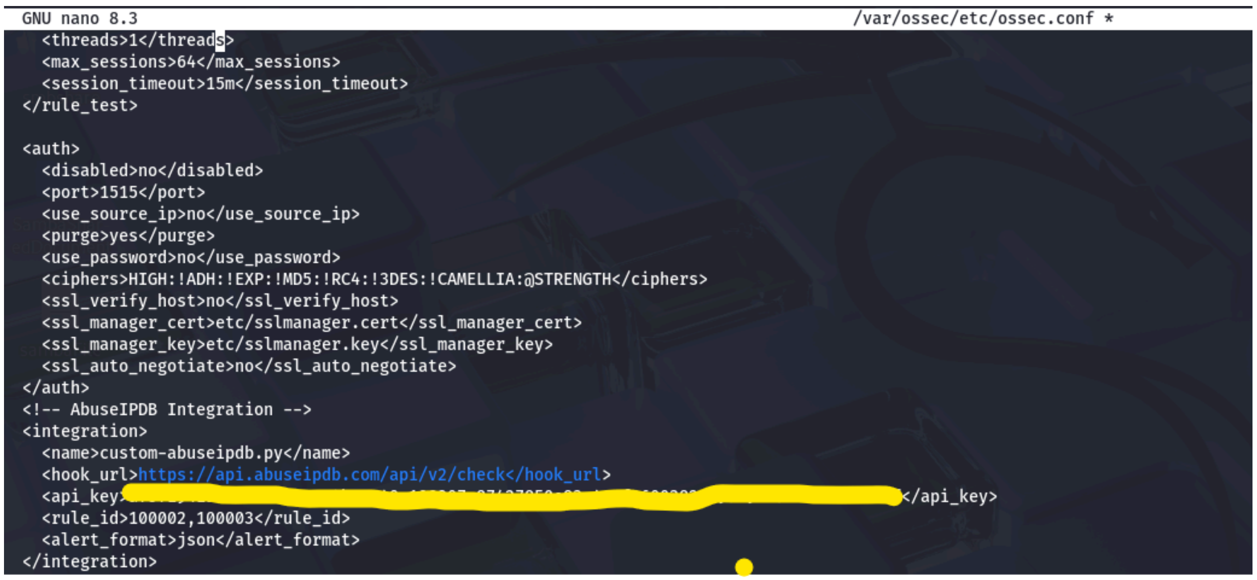
What this does:

Tells Wazuh to run `custom-abuseipdb.py` when rules `100002` and `100003` fire.

Sends the alert content in JSON format.

Provides the API key (which your script can read via `stdin`, env, or argument).

Why:
 This step registers the integration, allowing Wazuh to trigger your script automatically just like it does with Slack, VirusTotal, or any third-party integration.



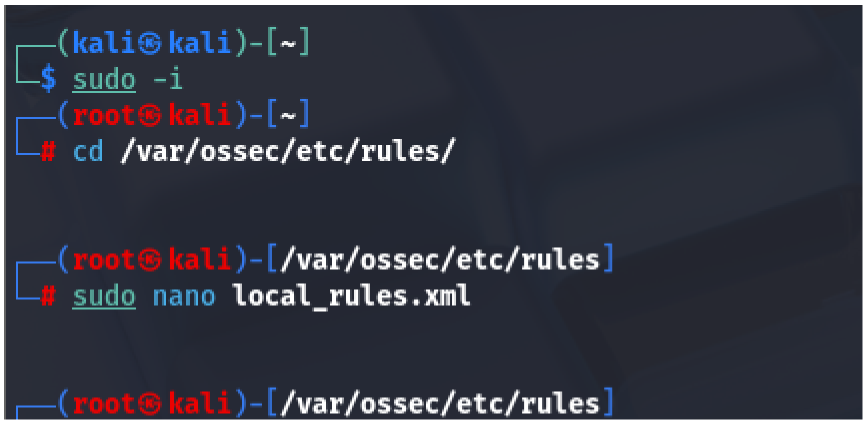
```
GNU nano 8.3                                                   /var/ossec/etc/ossec.conf *
    <threads>1</threads>
    <max_sessions>64</max_sessions>
    <session_timeout>15m</session_timeout>
  </rule_test>

  <auth>
    <disabled>no</disabled>
    <port>1515</port>
    <use_source_ip>no</use_source_ip>
    <purge>yes</purge>
    <use_password>no</use_password>
    <ciphers>HIGH:!ADH:!EXP:!MD5:!RC4:!3DES:!CAMELLIA:@STRENGTH</ciphers>
    <ssl_verify_host>no</ssl_verify_host>
    <ssl_manager_cert>etc/sslmanager.cert</ssl_manager_cert>
    <ssl_manager_key>etc/sslmanager.key</ssl_manager_key>
    <ssl_auto_negotiate>no</ssl_auto_negotiate>
  </auth>
  <!-- AbuseIPDB Integration -->
  <integration>
    <name>custom-abuseipdb.py</name>
    <hook_url>https://api.abuseipdb.com/api/v2/check</hook_url>
    <api_key>[redacted]</api_key>
    <rule_id>100002,100003</rule_id>
    <alert_format>json</alert_format>
  </integration>
```

Define Custom Rules for SSH Login Events

We create two rules to catch public IPs attempting SSH login:

Edit the rules file:

cd /var/ossec/etc/rules/
sudo nano local_rules.xml



```
┌──(kali㉿kali)-[~]
└─$ sudo -i
┌──(root㉿kali)-[~]
└─# cd /var/ossec/etc/rules/


┌──(root㉿kali)-[/var/ossec/etc/rules]
└─# sudo nano local_rules.xml


┌──(root㉿kali)-[/var/ossec/etc/rules]
```

Paste These Rules:

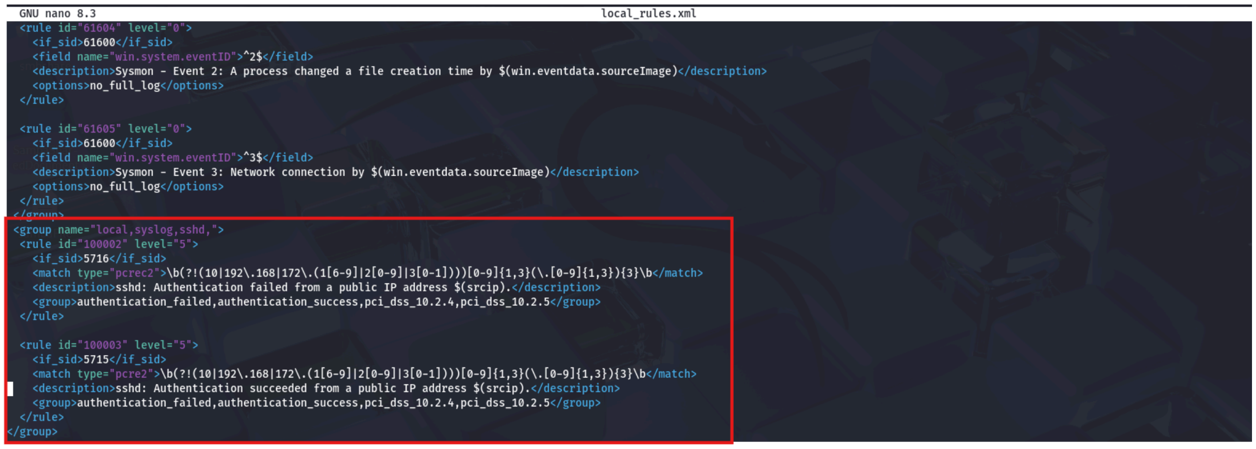
```
<group name="local,syslog,sshd,">
  <rule id="100002" level="5">
    <if_sid>5716</if_sid>
    <match
type="pcre2">\b(?!(10)|192\.168|172\.(2[0-9]|1[6-9]|3[0-1])|(25[6-9]|2[6-9][0-9]|[3-9][0-9][0-9]|
99[1-9]))[0-9]{1,3}\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\
.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)</match>
    <description>sshd: Authentication failed from a public IP address $(srcip).</description>

<group>authentication_failed,authentication_success,pci_dss_10.2.4,pci_dss_10.2.5,</gro
up>
  </rule>

  <rule id="100003" level="5">
    <if_sid>5715</if_sid>
    <match
type="pcre2">\b(?!(10)|192\.168|172\.(2[0-9]|1[6-9]|3[0-1])|(25[6-9]|2[6-9][0-9]|[3-9][0-9][0-9]|
99[1-9]))[0-9]{1,3}\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\
.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)</match>
    <description>sshd: Authentication succeeded from a public IP address
$(srcip).</description>

<group>authentication_failed,authentication_success,pci_dss_10.2.4,pci_dss_10.2.5,</gro
up>
  </rule>
</group>
```

What this does:

Creates two custom rules: one for **failed** SSH auth from a public IP (5716), one for **successful** (5715).

Only public IPs are matched using regex (ignores private/internal IPs).
Each rule triggers your integration.

Why:
You want to only check **untrusted/external IPs**, not local ones (e.g., 192.168.x.x). These rules are filters.

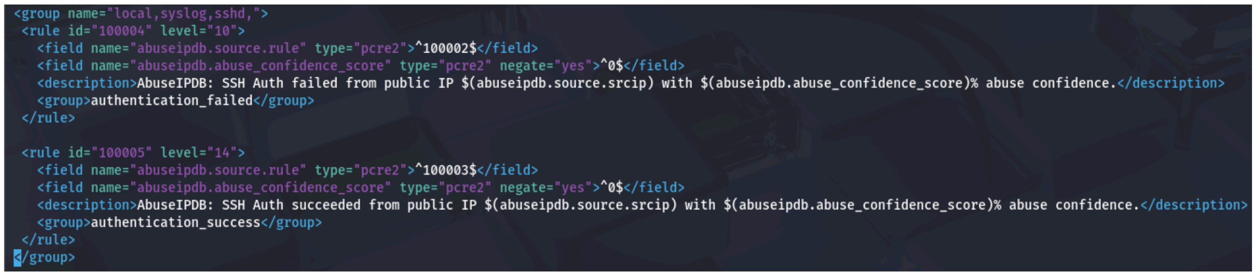


```
GNU nano 8.3                                                          local_rules.xml
  <rule id="61604" level="0">
    <if_sid>61600</if_sid>
    <field name="win.system.eventID">^2$</field>
    <description>Sysmon - Event 2: A process changed a file creation time by $(win.eventdata.sourceImage)</description>
    <options>no_full_log</options>
  </rule>

  <rule id="61605" level="0">
    <if_sid>61600</if_sid>
    <field name="win.system.eventID">^3$</field>
    <description>Sysmon - Event 3: Network connection by $(win.eventdata.sourceImage)</description>
    <options>no_full_log</options>
  </rule>
</group>
<group name="local,syslog,sshd,">
  <rule id="100002" level="5">
    <if_sid>5716</if_sid>
    <match type="pcrec2">\b(?!(10|192\.168|172\.(1[6-9]|2[0-9]|3[0-1])))[0-9]{1,3}(\.[0-9]{1,3}){3}\b</match>
    <description>sshd: Authentication failed from a public IP address $(srcip).</description>
    <group>authentication_failed,authentication_success,pci_dss_10.2.4,pci_dss_10.2.5</group>
  </rule>

  <rule id="100003" level="5">
    <if_sid>5715</if_sid>
    <match type="pcre2">\b(?!(10|192\.168|172\.(1[6-9]|2[0-9]|3[0-1])))[0-9]{1,3}(\.[0-9]{1,3}){3}\b</match>
    <description>sshd: Authentication succeeded from a public IP address $(srcip).</description>
    <group>authentication_failed,authentication_success,pci_dss_10.2.4,pci_dss_10.2.5</group>
  </rule>
</group>
```

Add AbuseIPDB Result Handling Rules

Continue editing the same `local_rules.xml` file.

```
<group name="local,syslog,sshd,">

  <rule id="100004" level="10">

    <field name="abuseipdb.source.rule" type="pcre2">^100002$</field>

    <field name="abuseipdb.abuse_confidence_score" type="pcre2" negate="yes">^0$</field>

    <description>AbuseIPDB: SSH Authentication failed from a public IP address $(abuseipdb.source.srcip) with $(abuseipdb.abuse_confidence_score)% confidence of abuse.</description>

    <group>authentication_failed,pci_dss_10.2.4,pci_dss_10.2.5,</group>

  </rule>

  <rule id="100005" level="14">

    <field name="abuseipdb.source.rule" type="pcre2">^100003$</field>

    <field name="abuseipdb.abuse_confidence_score" type="pcre2" negate="yes">^0$</field>

    <description>AbuseIPDB: SSH Authentication succeeded from a public IP address $(abuseipdb.source.srcip) with $(abuseipdb.abuse_confidence_score)% confidence of abuse.</description>
    <group>authentication_failed,pci_dss_10.2.4,pci_dss_10.2.5,</group>
  </rule>
</group>
```



```
<group name="local,syslog,sshd,">
 <rule id="100004" level="10">
   <field name="abuseipdb.source.rule" type="pcre2">^100002$</field>
   <field name="abuseipdb.abuse_confidence_score" type="pcre2" negate="yes">^0$</field>
   <description>AbuseIPDB: SSH Auth failed from public IP $(abuseipdb.source.srcip) with $(abuseipdb.abuse_confidence_score)% abuse confidence.</description>
   <group>authentication_failed</group>
 </rule>

 <rule id="100005" level="14">
   <field name="abuseipdb.source.rule" type="pcre2">^100003$</field>
   <field name="abuseipdb.abuse_confidence_score" type="pcre2" negate="yes">^0$</field>
   <description>AbuseIPDB: SSH Auth succeeded from public IP $(abuseipdb.source.srcip) with $(abuseipdb.abuse_confidence_score)% abuse confidence.</description>
   <group>authentication_success</group>
 </rule>
</group>
```

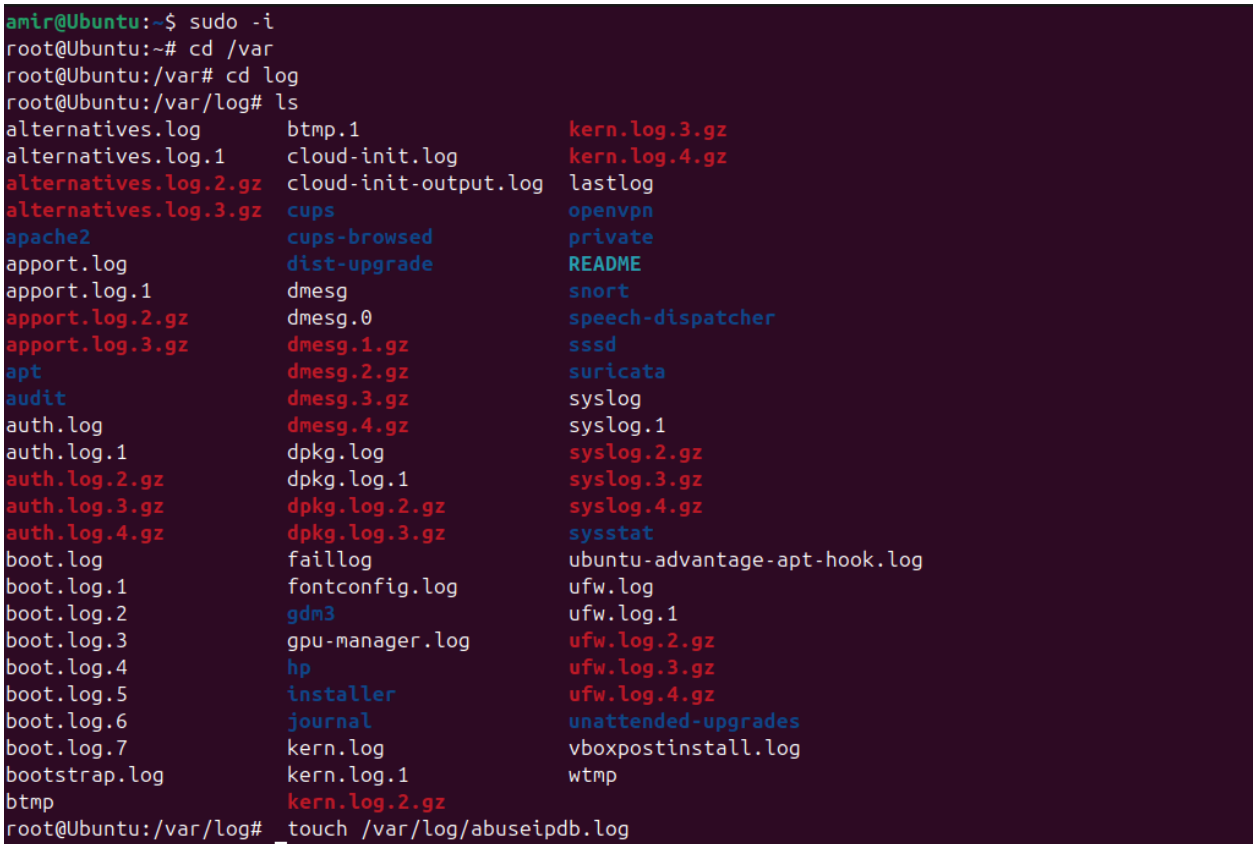
Configure Agent (Ubuntu) to Feed Logs

a. Switch to Root:

```
sudo -i
```

b. Create a Log File for Testing:

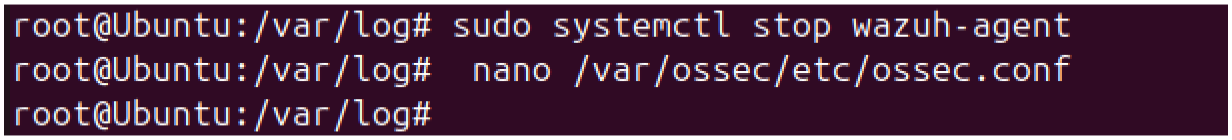
```
touch /var/log/abuseipdb.log
```



```
amir@Ubuntu:~$ sudo -i
root@Ubuntu:~# cd /var
root@Ubuntu:/var# cd log
root@Ubuntu:/var/log# ls
alternatives.log       btmp.1                 kern.log.3.gz
alternatives.log.1     cloud-init.log         kern.log.4.gz
alternatives.log.2.gz  cloud-init-output.log  lastlog
alternatives.log.3.gz  cups                   openvpn
apache2                cups-browsed           private
apport.log             dist-upgrade           README
apport.log.1           dmesg                  snort
apport.log.2.gz        dmesg.0                speech-dispatcher
apport.log.3.gz        dmesg.1.gz             sssd
apt                    dmesg.2.gz             suricata
audit                  dmesg.3.gz             syslog
auth.log               dmesg.4.gz             syslog.1
auth.log.1             dpkg.log               syslog.2.gz
auth.log.2.gz          dpkg.log.1             syslog.3.gz
auth.log.3.gz          dpkg.log.2.gz          syslog.4.gz
auth.log.4.gz          dpkg.log.3.gz          sysstat
boot.log               faillog                ubuntu-advantage-apt-hook.log
boot.log.1             fontconfig.log         ufw.log
boot.log.2             gdm3                   ufw.log.1
boot.log.3             gpu-manager.log        ufw.log.2.gz
boot.log.4             hp                     ufw.log.3.gz
boot.log.5             installer              ufw.log.4.gz
boot.log.6             journal                unattended-upgrades
boot.log.7             kern.log               vboxpostinstall.log
bootstrap.log          kern.log.1             wtmp
btmp                   kern.log.2.gz
root@Ubuntu:/var/log# touch /var/log/abuseipdb.log
```

c. Stop Wazuh Agent Temporarily:

systemctl stop wazuh-agent



```
root@Ubuntu:/var/log# sudo systemctl stop wazuh-agent
root@Ubuntu:/var/log#  nano /var/ossec/etc/ossec.conf
root@Ubuntu:/var/log#
```

d. Edit Agent Config:

nano /var/ossec/etc/ossec.conf

Add this before `</ossec_config>`:

```
<localfile>
  <log_format>syslog</log_format>
  <location>/var/log/abuseipdb.log</location>
</localfile>
```

e. Restart the Agent:

systemctl start wazuh-agent

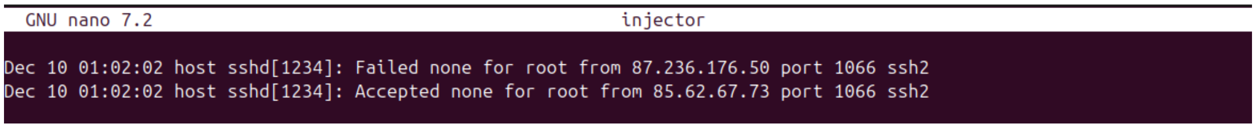
Inject Sample Log Events for Testing

Create a file with test SSH events:

nano injector

Dec 10 01:02:02 host sshd[1234]: Failed none for root from 87.236.176.50 port 1066 ssh2

Dec 10 01:02:02 host sshd[1234]: Accepted none for root from 85.62.67.73 port 1066 ssh2



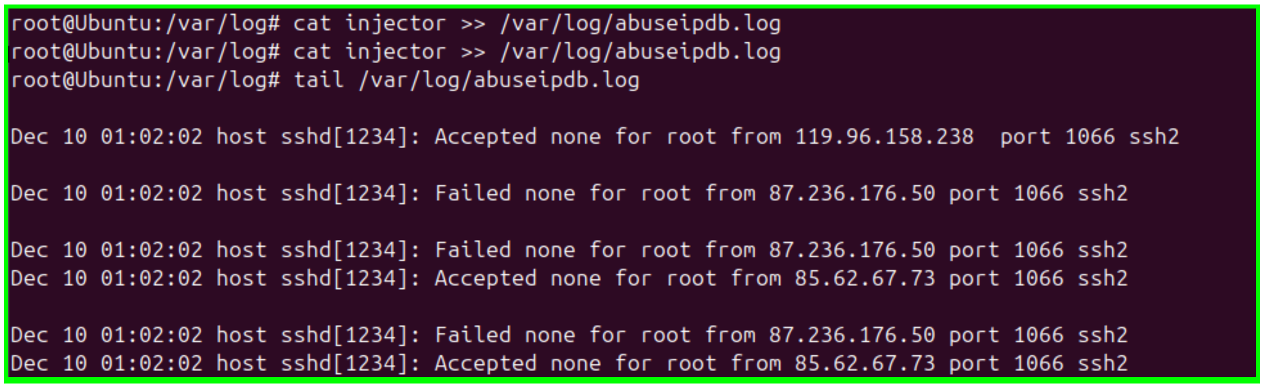
```
GNU nano 7.2                                        injector

Dec 10 01:02:02 host sshd[1234]: Failed none for root from 87.236.176.50 port 1066 ssh2
Dec 10 01:02:02 host sshd[1234]: Accepted none for root from 85.62.67.73 port 1066 ssh2
```

Append to the monitored log:

cat injector >> /var/log/abuseipdb.log

Check the file:

tail /var/log/abuseipdb.log



```
root@Ubuntu:/var/log# cat injector >> /var/log/abuseipdb.log
root@Ubuntu:/var/log# cat injector >> /var/log/abuseipdb.log
root@Ubuntu:/var/log# tail /var/log/abuseipdb.log

Dec 10 01:02:02 host sshd[1234]: Accepted none for root from 119.96.158.238  port 1066 ssh2

Dec 10 01:02:02 host sshd[1234]: Failed none for root from 87.236.176.50 port 1066 ssh2

Dec 10 01:02:02 host sshd[1234]: Failed none for root from 87.236.176.50 port 1066 ssh2
Dec 10 01:02:02 host sshd[1234]: Accepted none for root from 85.62.67.73 port 1066 ssh2

Dec 10 01:02:02 host sshd[1234]: Failed none for root from 87.236.176.50 port 1066 ssh2
Dec 10 01:02:02 host sshd[1234]: Accepted none for root from 85.62.67.73 port 1066 ssh2
```

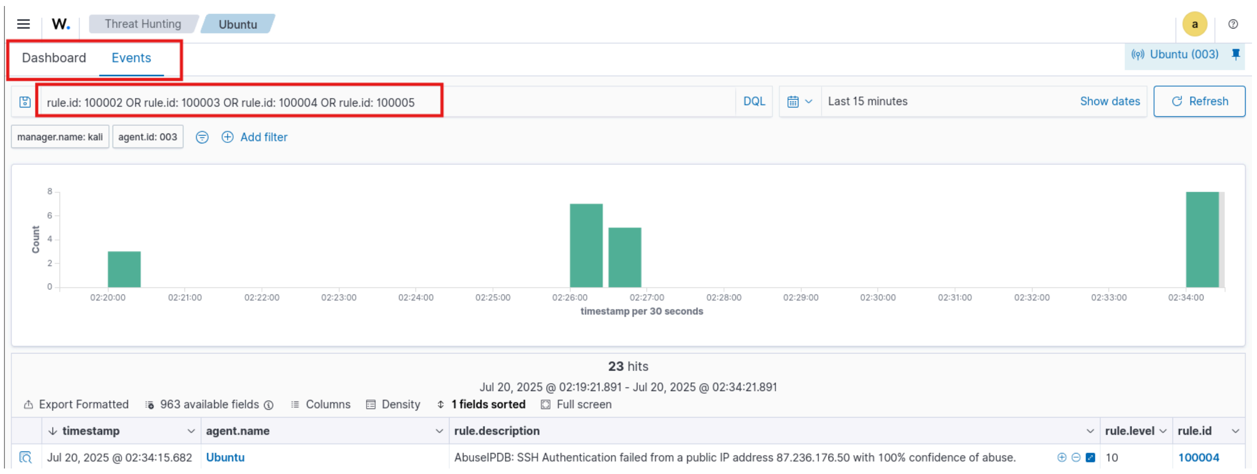
Monitor Results in Wazuh Dashboard

Go to Wazuh Dashboard → Discover, and search for:

- data.integration:custom-abuseipdb
- rule.id:100004 or 100005
- rule.id: 100002 OR rule.id: 100003 OR rule.id: 100004 OR rule.id: 100005

You should see:

- Source IP
- Abuse Score
- Full SSH log
- Rule description

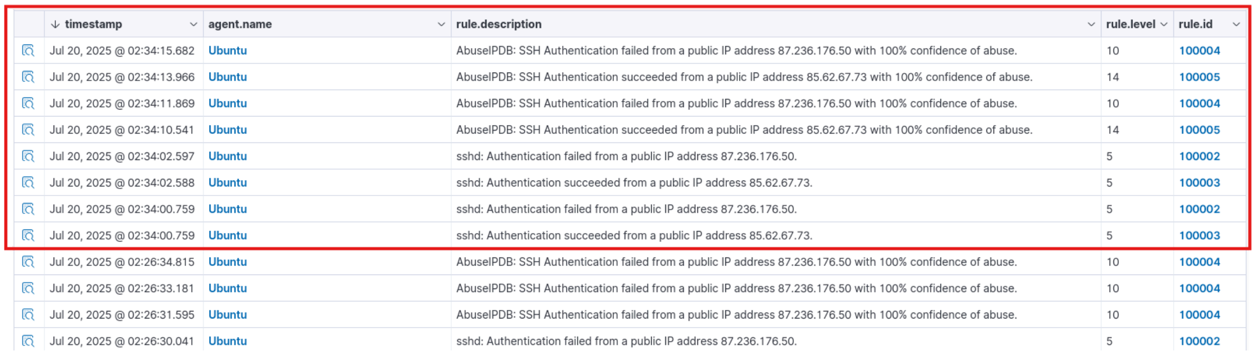


rule.id: 100002 OR rule.id: 100003 OR rule.id: 100004 OR rule.id: 100005

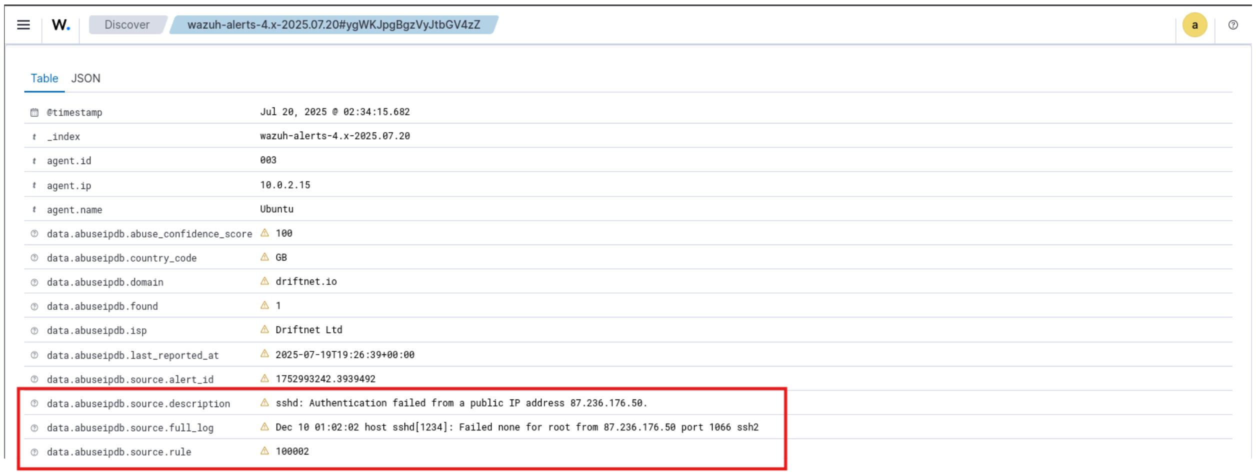
23 hits

Jul 20, 2025 @ 02:19:21.891 - Jul 20, 2025 @ 02:34:21.891

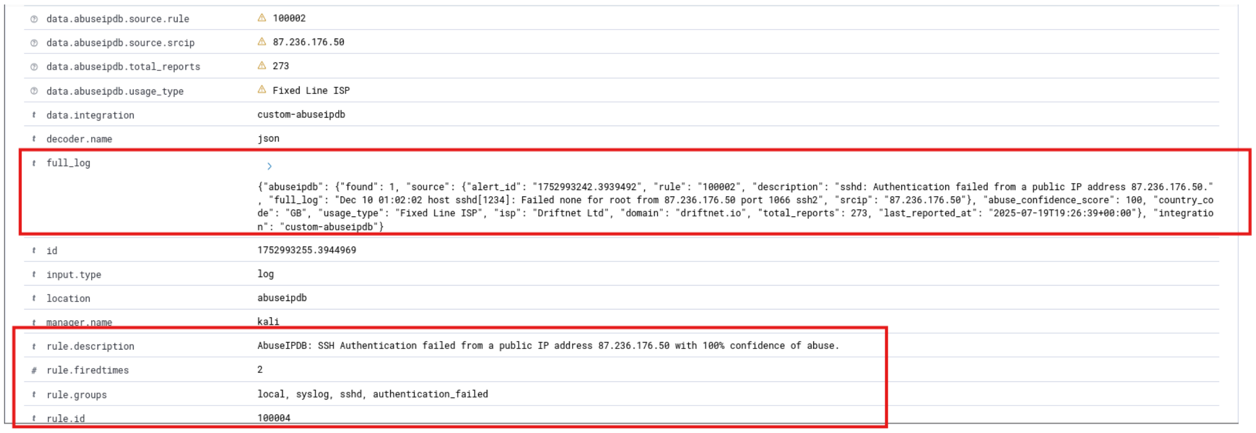
timestamp	agent.name	rule.description	rule.level	rule.id
Jul 20, 2025 @ 02:34:15.682	Ubuntu	AbuseIPDB: SSH Authentication failed from a public IP address 87.236.176.50 with 100% confidence of abuse.	10	100004



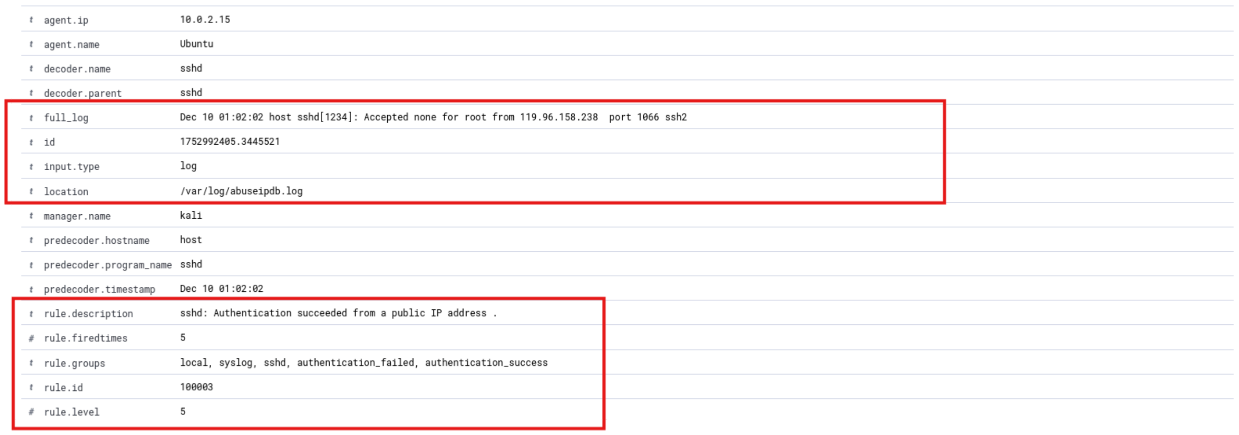
timestamp	agent.name	rule.description	rule.level	rule.id
Jul 20, 2025 @ 02:34:15.682	Ubuntu	AbuseIPDB: SSH Authentication failed from a public IP address 87.236.176.50 with 100% confidence of abuse.	10	100004
Jul 20, 2025 @ 02:34:13.966	Ubuntu	AbuseIPDB: SSH Authentication succeeded from a public IP address 85.62.67.73 with 100% confidence of abuse.	14	100005
Jul 20, 2025 @ 02:34:11.869	Ubuntu	AbuseIPDB: SSH Authentication failed from a public IP address 87.236.176.50 with 100% confidence of abuse.	10	100004
Jul 20, 2025 @ 02:34:10.541	Ubuntu	AbuseIPDB: SSH Authentication succeeded from a public IP address 85.62.67.73 with 100% confidence of abuse.	14	100005
Jul 20, 2025 @ 02:34:02.597	Ubuntu	sshd: Authentication failed from a public IP address 87.236.176.50.	5	100002
Jul 20, 2025 @ 02:34:02.588	Ubuntu	sshd: Authentication succeeded from a public IP address 85.62.67.73.	5	100003
Jul 20, 2025 @ 02:34:00.759	Ubuntu	sshd: Authentication failed from a public IP address 87.236.176.50.	5	100002
Jul 20, 2025 @ 02:34:00.759	Ubuntu	sshd: Authentication succeeded from a public IP address 85.62.67.73.	5	100003
Jul 20, 2025 @ 02:26:34.815	Ubuntu	AbuseIPDB: SSH Authentication failed from a public IP address 87.236.176.50 with 100% confidence of abuse.	10	100004
Jul 20, 2025 @ 02:26:33.181	Ubuntu	AbuseIPDB: SSH Authentication failed from a public IP address 87.236.176.50 with 100% confidence of abuse.	10	100004
Jul 20, 2025 @ 02:26:31.595	Ubuntu	AbuseIPDB: SSH Authentication failed from a public IP address 87.236.176.50 with 100% confidence of abuse.	10	100004
Jul 20, 2025 @ 02:26:30.041	Ubuntu	sshd: Authentication failed from a public IP address 87.236.176.50.	5	100002



```
@timestamp                           Jul 20, 2025 @ 02:34:15.682
_index                               wazuh-alerts-4.x-2025.07.20
agent.id                             003
agent.ip                             10.0.2.15
agent.name                           Ubuntu
data.abuseipdb.abuse_confidence_score  100
data.abuseipdb.country_code          GB
data.abuseipdb.domain                driftnet.io
data.abuseipdb.found                 1
data.abuseipdb.isp                   Driftnet Ltd
data.abuseipdb.last_reported_at      2025-07-19T19:26:39+00:00
data.abuseipdb.source.alert_id       1752993242.3939492
data.abuseipdb.source.description    sshd: Authentication failed from a public IP address 87.236.176.50.
data.abuseipdb.source.full_log       Dec 10 01:02:02 host sshd[1234]: Failed none for root from 87.236.176.50 port 1066 ssh2
data.abuseipdb.source.rule           100002
```



```
data.abuseipdb.source.rule           100002
data.abuseipdb.source.srcip          87.236.176.50
data.abuseipdb.total_reports         273
data.abuseipdb.usage_type            Fixed Line ISP
data.integration                     custom-abuseipdb
decoder.name                         json
full_log                             >
                                     {"abuseipdb": {"found": 1, "source": {"alert_id": "1752993242.3939492", "rule": "100002", "description": "sshd: Authentication failed from a public IP address 87.236.176.50."
                                     , "full_log": "Dec 10 01:02:02 host sshd[1234]: Failed none for root from 87.236.176.50 port 1066 ssh2", "srcip": "87.236.176.50"}, "abuse_confidence_score": 100, "country_co
                                     de": "GB", "usage_type": "Fixed Line ISP", "isp": "Driftnet Ltd", "domain": "driftnet.io", "total_reports": 273, "last_reported_at": "2025-07-19T19:26:39+00:00"}, "integratio
                                     n": "custom-abuseipdb"}
id                                   1752993255.3944969
input.type                           log
location                             abuseipdb
manager.name                         kali
rule.description                     AbuseIPDB: SSH Authentication failed from a public IP address 87.236.176.50 with 100% confidence of abuse.
rule.firedtimes                      2
rule.groups                          local, syslog, sshd, authentication_failed
rule.id                              100004
```



```
agent.ip                  10.0.2.15
agent.name                Ubuntu
decoder.name              sshd
decoder.parent            sshd
full_log                  Dec 10 01:02:02 host sshd[1234]: Accepted none for root from 119.96.158.238  port 1066 ssh2
id                        1752992405.3445521
input.type                log
location                  /var/log/abuseipdb.log
manager.name              kali
predecoder.hostname       host
predecoder.program_name   sshd
predecoder.timestamp      Dec 10 01:02:02
rule.description          sshd: Authentication succeeded from a public IP address .
rule.firedtimes           5
rule.groups               local, syslog, sshd, authentication_failed, authentication_success
rule.id                   100003
rule.level                5
```

Summary:

This document describes the implementation and workflow of integrating the AbuseIPDB threat intelligence service with the Wazuh SIEM (Security Information and Event Management) platform to enhance the detection and analysis of potentially malicious SSH login attempts.

The integration was accomplished by developing a custom Python script (`custom-abuseipdb.py`) placed in the Wazuh `integrations` directory. This script is executed when a matching rule triggers an alert and is responsible for querying the AbuseIPDB API to check the reputation of the source IP address involved in the event. The script also logs additional threat intelligence data back into Wazuh for enrichment and correlation.

To enable this functionality, the Wazuh manager was configured to:

Monitor a custom log file (`/var/log/abuseipdb.log`) that contains SSH activity.
Use custom detection rules defined in `local_rules.xml` to identify patterns such as failed or successful SSH login attempts.
Execute the custom integration based on specific rule IDs.

During testing, simulated SSH login events were written to the monitored log file to verify detection accuracy and script execution. Wazuh successfully processed the log entries, triggered the correct rule, and called the integration script, which queried the AbuseIPDB service and recorded the response.

For regex pattern matching to function correctly, the PCRE2 (Perl-Compatible Regular Expressions) library was required and confirmed to be installed on the Wazuh manager.

This integration demonstrates a complete flow from log ingestion and real-time detection to external threat intelligence enrichment. It enhances Wazuh's capabilities by providing actionable insights and improving security visibility, especially in the context of SSH-based brute-force or unauthorized access attempts.