AI-Based Anomaly Detection in Wazuh

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 | AI-Driven Threat Detection

AI-Based Anomaly Detection in Wazuh

Lab Summary

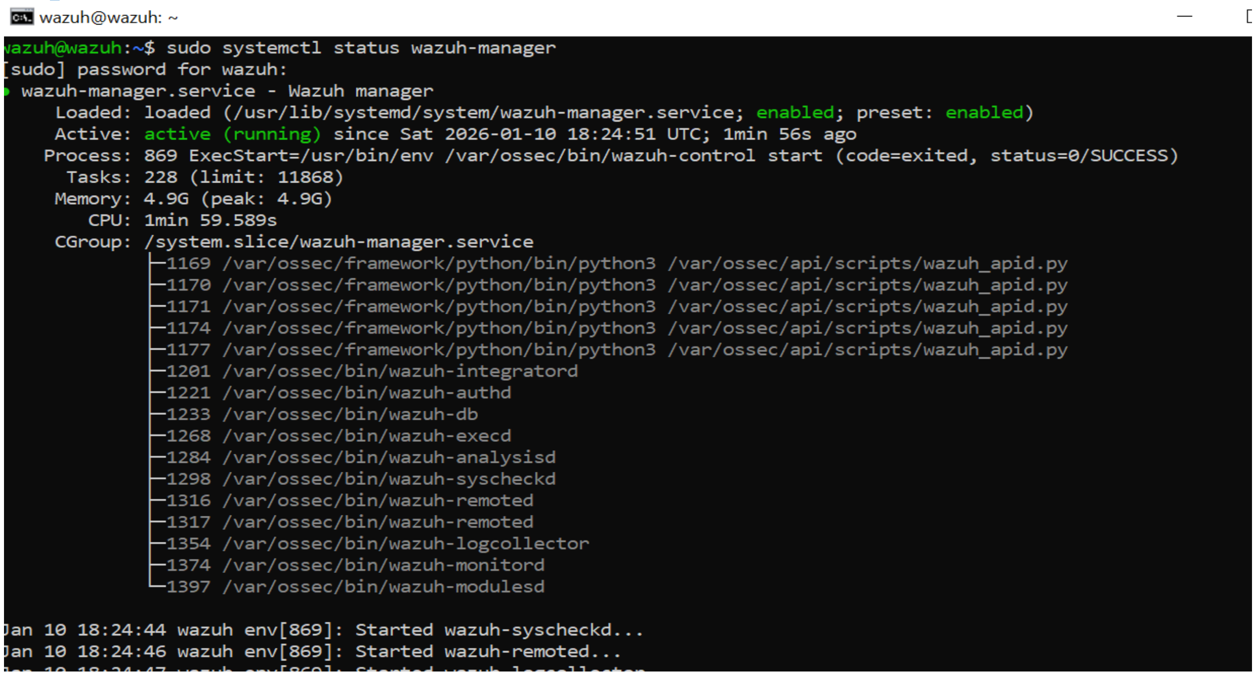
Artificial Intelligence (AI)–based anomaly detection in Wazuh leverages the OpenSearch Anomaly Detection plugin to identify abnormal patterns in security telemetry that may not be detected by traditional rule-based systems. Unlike static rules, anomaly detection uses machine learning models to establish a baseline of normal behavior and highlight deviations.

In this task, anomaly detection was enabled using Wazuh alert data (`wazuh-alerts-*`) to monitor authentication-related activities. The objective was to understand how AI-driven detection enhances SOC monitoring by identifying unusual patterns such as abnormal login attempts, spikes in alerts, or irregular system behavior.

Environment Overview

- **Wazuh Manager**: Ubuntu Server
- **Wazuh Agent**: Ubuntu Linux
- **Attack Machine**: Kali Linux
- **SIEM & ML Engine**: OpenSearch with Anomaly Detection plugin
- **Data Source**: Wazuh alerts index (`wazuh-alerts-*`)

Implementation



```
wazuh@wazuh: ~
azuh@wazuh:~$ sudo systemctl status wazuh-manager
[sudo] password for wazuh:
● wazuh-manager.service - Wazuh manager
     Loaded: loaded (/usr/lib/systemd/system/wazuh-manager.service; enabled; preset: enabled)
     Active: active (running) since Sat 2026-01-10 18:24:51 UTC; 1min 56s ago
    Process: 869 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0/SUCCESS)
      Tasks: 228 (limit: 11868)
     Memory: 4.9G (peak: 4.9G)
        CPU: 1min 59.589s
     CGroup: /system.slice/wazuh-manager.service
             ├─1169 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─1170 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─1171 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─1174 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─1177 /var/ossec/framework/python/bin/python3 /var/ossec/api/scripts/wazuh_apid.py
             ├─1201 /var/ossec/bin/wazuh-integratord
             ├─1221 /var/ossec/bin/wazuh-authd
             ├─1233 /var/ossec/bin/wazuh-db
             ├─1268 /var/ossec/bin/wazuh-execd
             ├─1284 /var/ossec/bin/wazuh-analysisd
             ├─1298 /var/ossec/bin/wazuh-syscheckd
             ├─1316 /var/ossec/bin/wazuh-remoted
             ├─1317 /var/ossec/bin/wazuh-remoted
             ├─1354 /var/ossec/bin/wazuh-logcollector
             ├─1374 /var/ossec/bin/wazuh-monitord
             └─1397 /var/ossec/bin/wazuh-modulesd

Jan 10 18:24:44 wazuh env[869]: Started wazuh-syscheckd...
Jan 10 18:24:46 wazuh env[869]: Started wazuh-remoted...
Jan 10 18:24:47 wazuh env[869]: Started wazuh-logcollector
```

Figure 1: Wazuh Manager service running successfully on Ubuntu server

Before configuring AI-based anomaly detection, the operational status of the Wazuh Manager was verified to ensure that the security monitoring infrastructure was functioning correctly. The Wazuh Manager services were confirmed to be active and running without errors. This step ensures that log collection, alert generation, and integration with OpenSearch are working as expected prior to enabling anomaly detection.

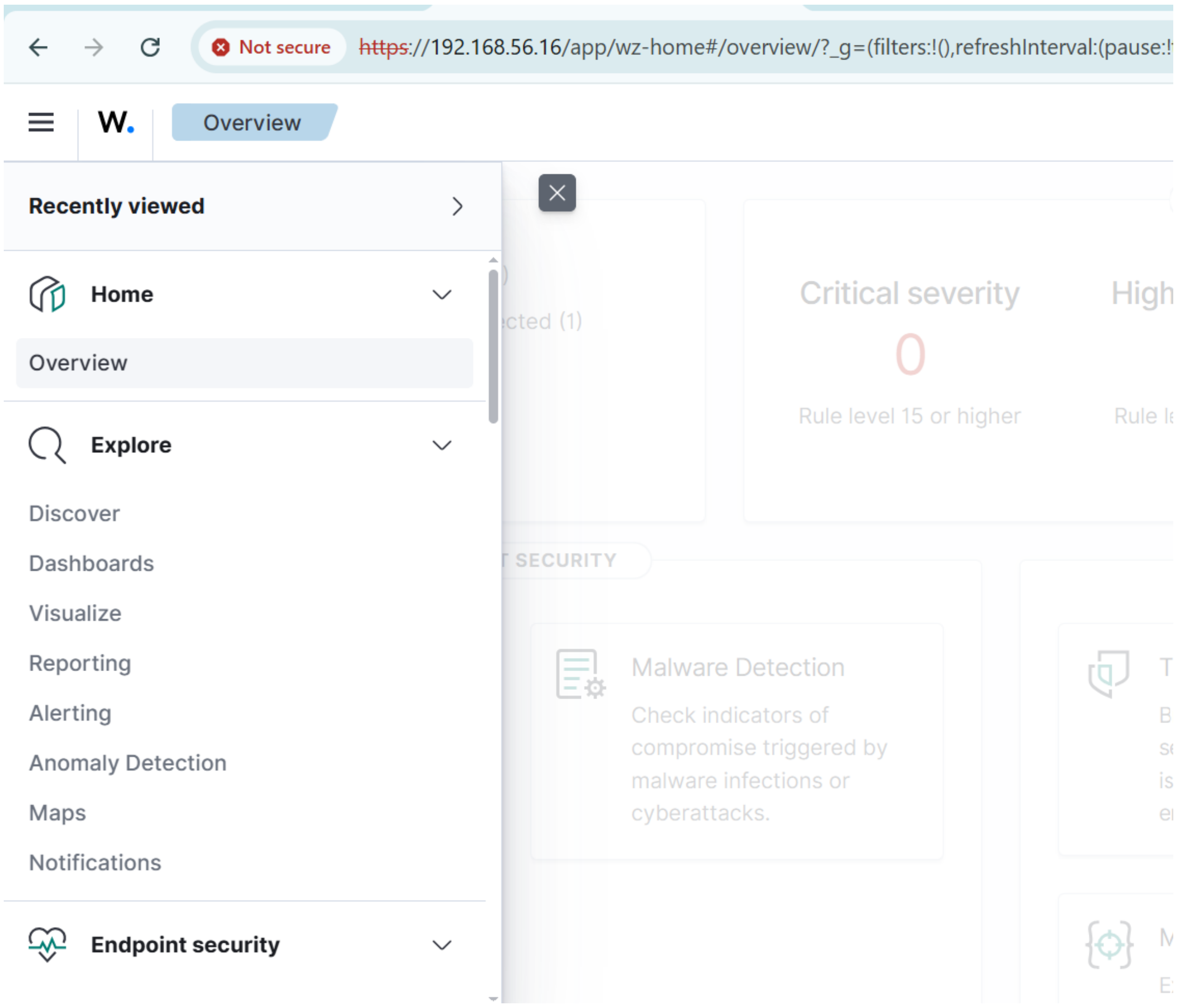
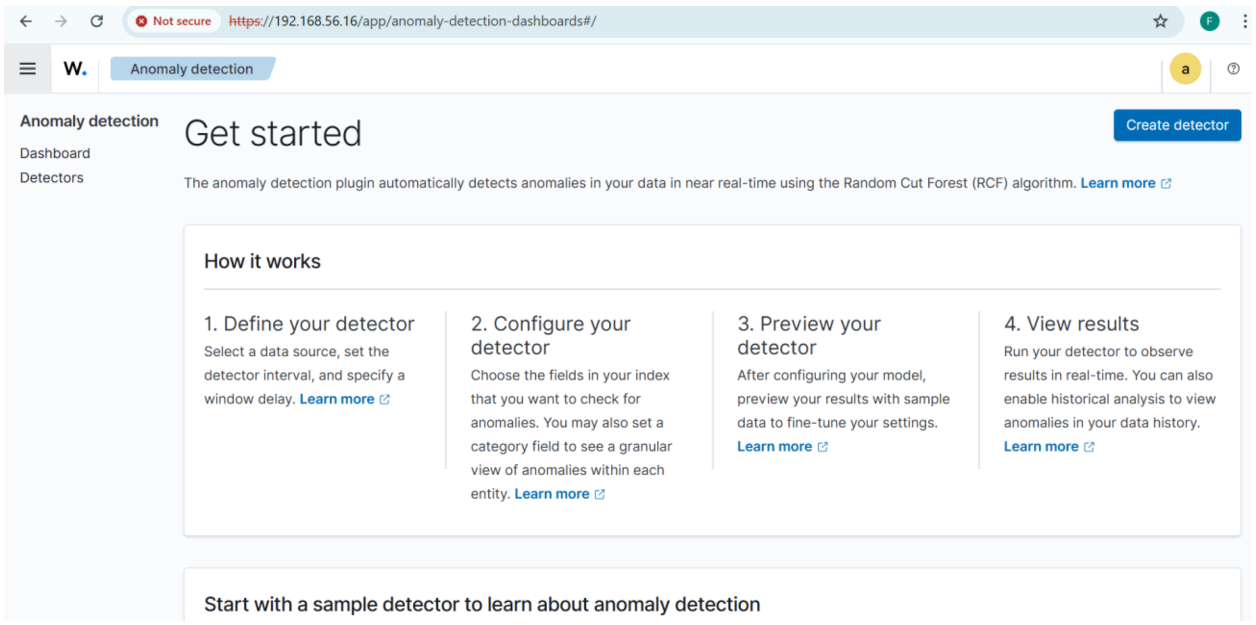


Figure 2: Wazuh Dashboard showing access to the Anomaly Detection module.

After confirming the operational status of the Wazuh Manager, the Wazuh Dashboard was accessed through the web interface. From the dashboard, the OpenSearch Anomaly Detection module was opened. This module provides machine learning–based capabilities to detect abnormal patterns in security telemetry beyond traditional rule-based alerts.


Not secure https://192.168.56.16/app/anomaly-detection-dashboards#/

Anomaly detection

Dashboard

Detectors

Get started

The anomaly detection plugin automatically detects anomalies in your data in near real-time using the Random Cut Forest (RCF) algorithm. Learn more

How it works

1. Define your detector
Select a data source, set the detector interval, and specify a window delay. Learn more

2. Configure your detector
Choose the fields in your index that you want to check for anomalies. You may also set a category field to see a granular view of anomalies within each entity. Learn more

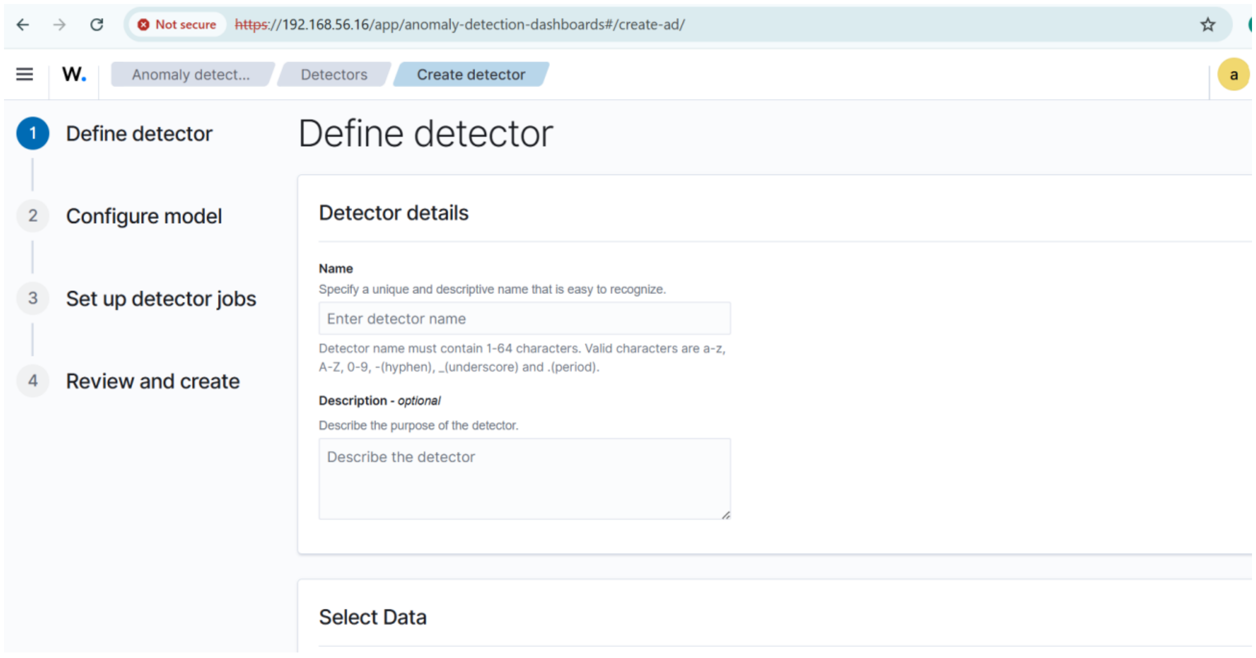
3. Preview your detector
After configuring your model, preview your results with sample data to fine-tune your settings. Learn more

4. View results
Run your detector to observe results in real-time. You can also enable historical analysis to view anomalies in your data history. Learn more

Start with a sample detector to learn about anomaly detection

Figure 3: Anomaly Detection main dashboard showing the option to create a new detector.

The Anomaly Detection section was opened from the Wazuh Dashboard, displaying the main anomaly detection interface. This page provides an overview of existing detectors and includes the option to create new anomaly detectors. From this interface, machine learning–based detectors can be configured to analyze Wazuh alert data for abnormal behavior.


Not secure https://192.168.56.16/app/anomaly-detection-dashboards#/create-ad/

Anomaly detect... / Detectors / Create detector

1 Define detector
2 Configure model
3 Set up detector jobs
4 Review and create

Define detector

Detector details

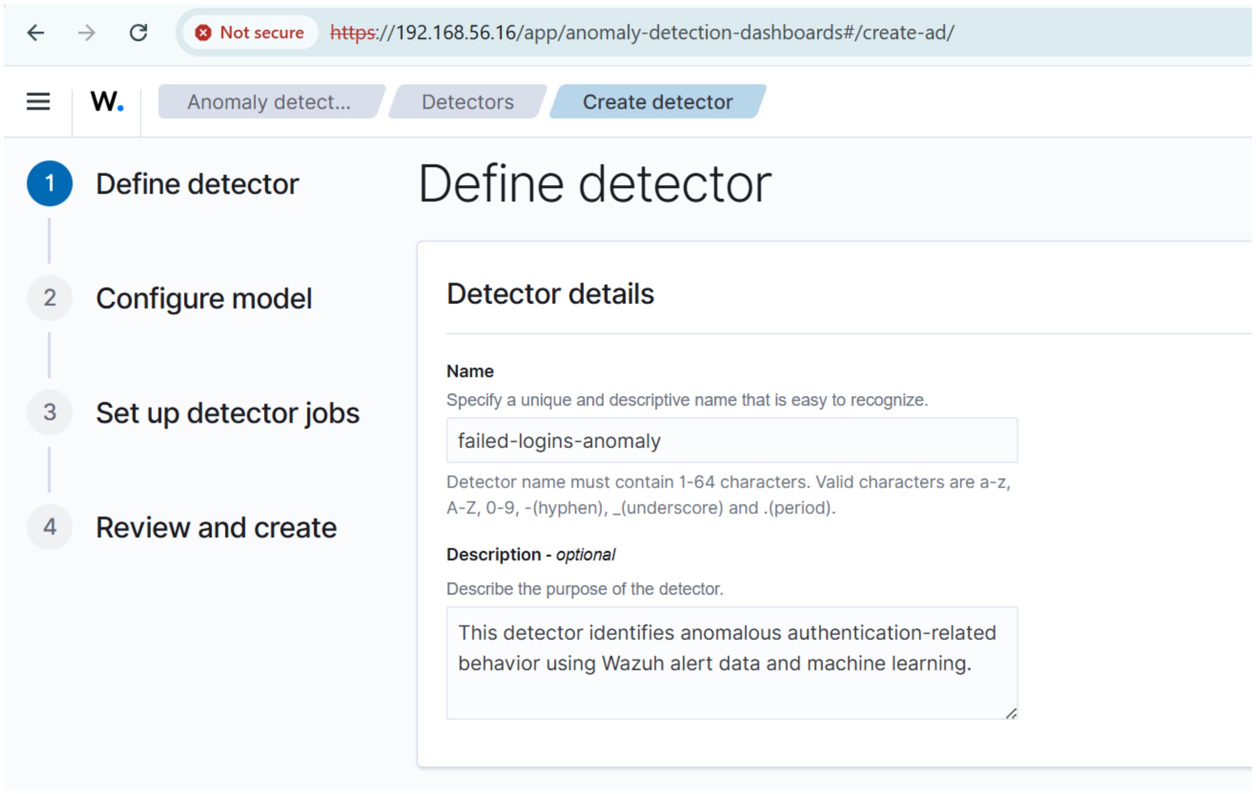
Name
Specify a unique and descriptive name that is easy to recognize.
Enter detector name
Detector name must contain 1-64 characters. Valid characters are a-z, A-Z, 0-9, -(hyphen), _(underscore) and .(period).

Description - *optional*
Describe the purpose of the detector.
Describe the detector

Select Data

Figure 4: Create new anomaly detector page with default configuration options

From the Anomaly Detection dashboard, the option to create a new detector was selected. The detector creation page was opened with default configuration settings. These default settings provide a baseline structure for configuring machine learning–based anomaly detection using OpenSearch. This step initiates the process of defining data sources, features, and detection logic for identifying anomalous security behavior.



Name

Specify a unique and descriptive name that is easy to recognize.

failed-logins-anomaly

Detector name must contain 1-64 characters. Valid characters are a-z, A-Z, 0-9, -(hyphen), _(underscore) and .(period).

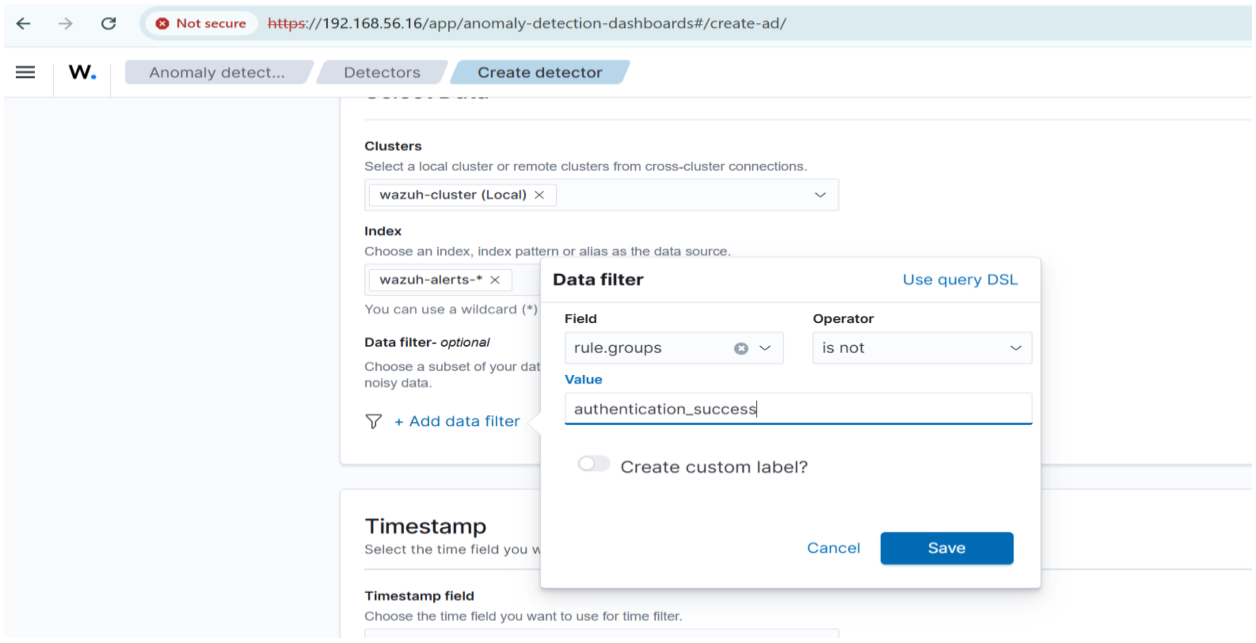
Description - *optional*

Describe the purpose of the detector.

This detector identifies anomalous authentication-related behavior using Wazuh alert data and machine learning.

Figure 5: Detector details showing the name failed-logins-anomaly and description.

A new anomaly detector was defined to monitor authentication-related anomalies using Wazuh alert data. In the detector details section, the detector was named **failed-logins-anomaly**, and a descriptive summary was provided to clearly indicate its purpose. This detector is designed to identify abnormal patterns in failed login activities, which may indicate brute-force attacks or unauthorized access attempts.


Not secure https://192.168.56.16/app/anomaly-detection-dashboards#/create-ad/

Anomaly detect... / Detectors / Create detector

Clusters
Select a local cluster or remote clusters from cross-cluster connections.
wazuh-cluster (Local)

Index
Choose an index, index pattern or alias as the data source.
wazuh-alerts-*
You can use a wildcard (*)

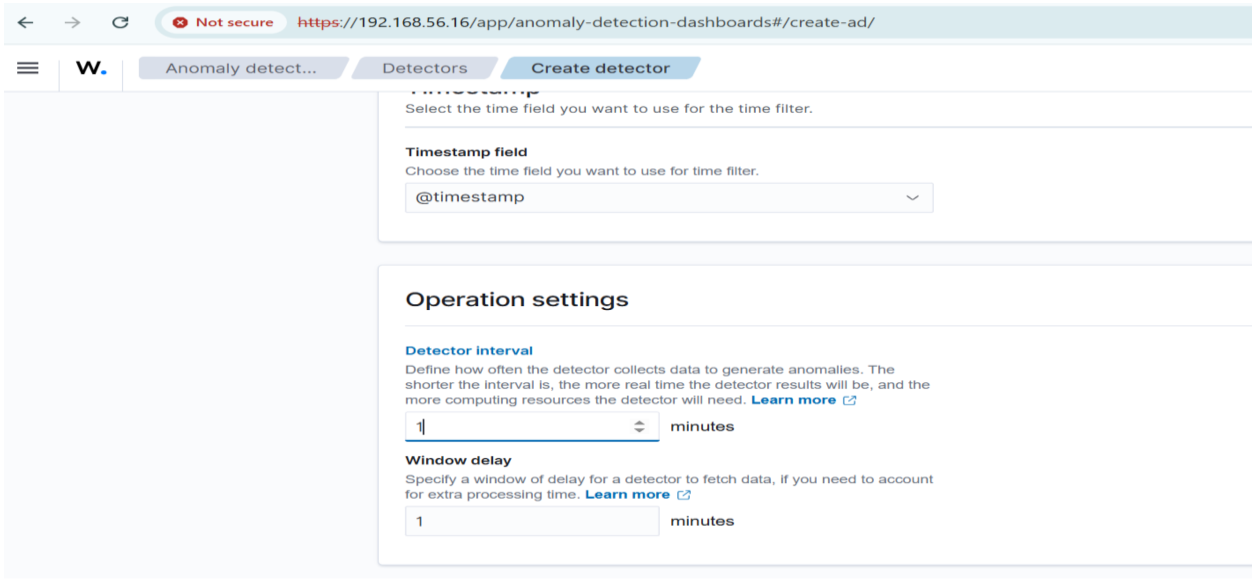
Data filter- *optional*
Choose a subset of your dat noisy data.
+ Add data filter

Data filter Use query DSL
Field: rule.groups
Operator: is not
Value: authentication_success
Create custom label?
Cancel Save

Timestamp
Select the time field you w
Timestamp field
Choose the time field you want to use for time filter.

Figure 6: Data source configuration showing wazuh-alerts- index and applied authentication failure filter.

The Wazuh alert index **wazuh-alerts-*** was selected as the data source for the anomaly detector. This index contains security alerts generated by Wazuh agents. To focus specifically on failed authentication events and reduce noise, a data filter was applied using the condition **rule.groups is not authentication_success**. This filtering ensures that only unsuccessful authentication attempts are analyzed by the machine learning model, improving the relevance and accuracy of anomaly detection.


Not secure https://192.168.56.16/app/anomaly-detection-dashboards#/create-ad/

Anomaly detect... / Detectors / Create detector

Select the time field you want to use for the time filter.
Timestamp field
Choose the time field you want to use for time filter.
@timestamp

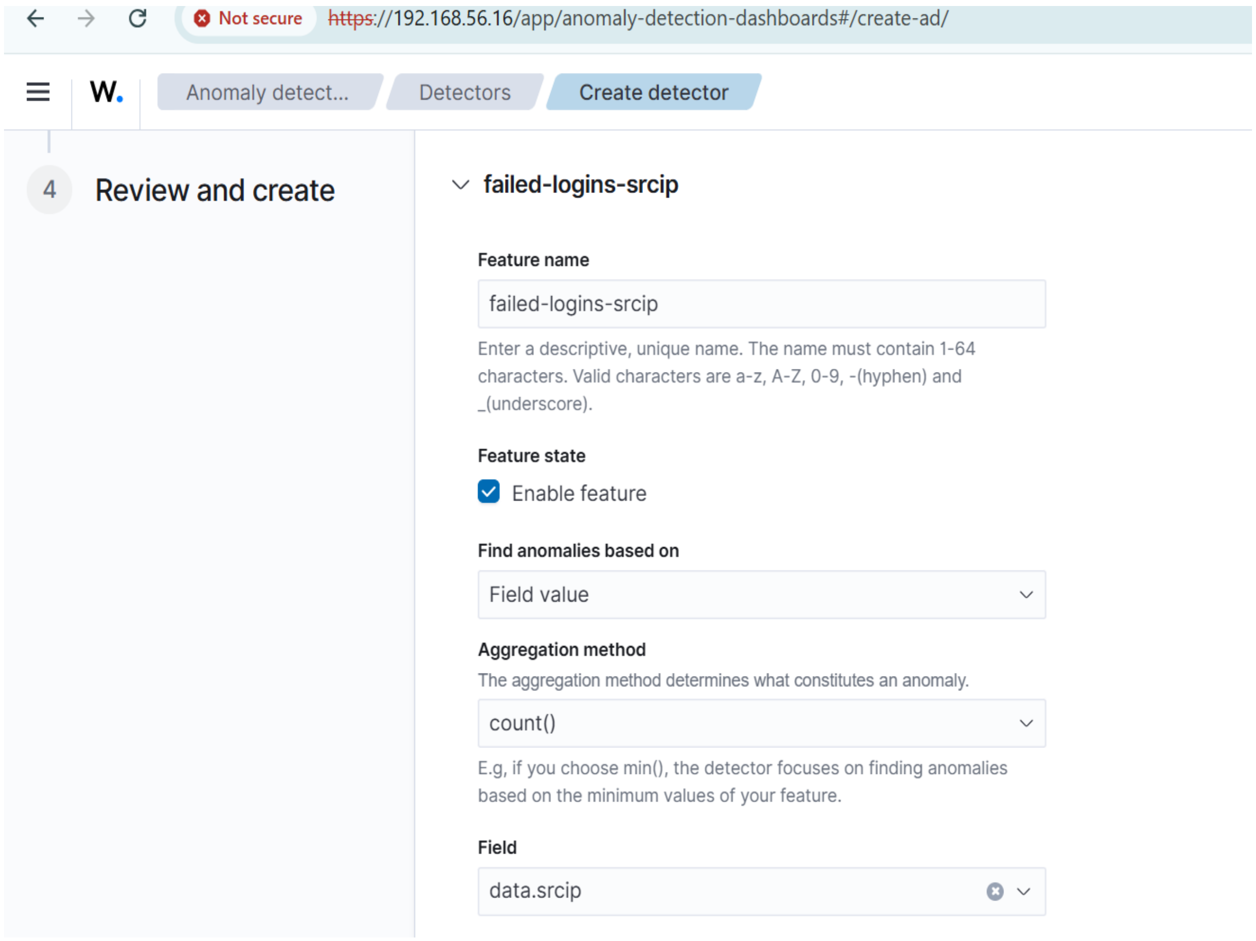
Operation settings

Detector interval
Define how often the detector collects data to generate anomalies. The shorter the interval is, the more real time the detector results will be, and the more computing resources the detector will need. Learn more
1 minutes

Window delay
Specify a window of delay for a detector to fetch data, if you need to account for extra processing time. Learn more
1 minutes

Figure 7: Operation settings showing timestamp selection, detector interval, and window delay configuration.

The timestamp field was selected to ensure that the anomaly detector accurately processes event time information from Wazuh alerts. To enable near real-time analysis, the detector interval was set to **1 minute**, allowing frequent data collection and evaluation. Additionally, a window delay of **1 minute** was configured to account for potential ingestion delays. These settings optimize the detector for timely identification of abnormal authentication behavior.



failed-logins-srcip

Feature name

failed-logins-srcip

Enter a descriptive, unique name. The name must contain 1-64 characters. Valid characters are a-z, A-Z, 0-9, -(hyphen) and _(underscore).

Feature state

✓ Enable feature

Find anomalies based on

Field value

Aggregation method

The aggregation method determines what constitutes an anomaly.

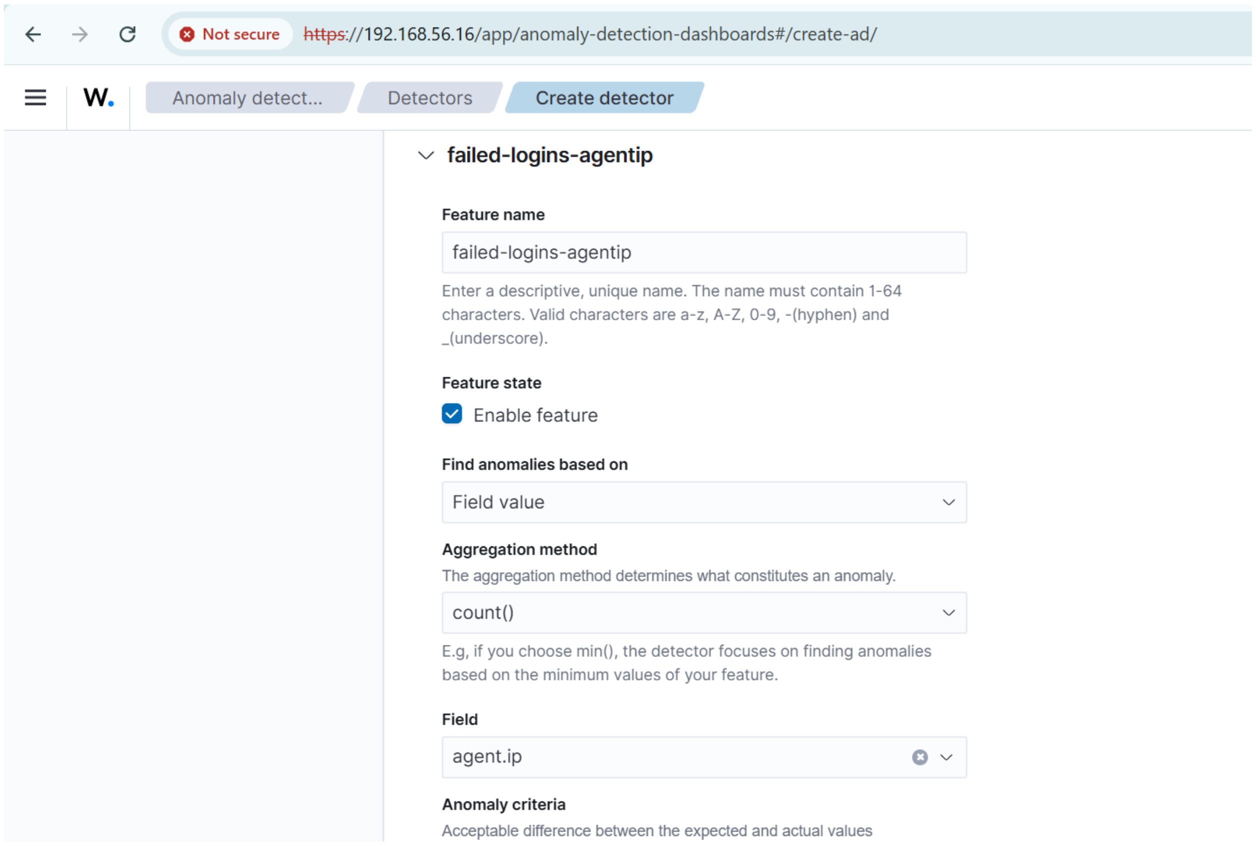
count()

E.g, if you choose min(), the detector focuses on finding anomalies based on the minimum values of your feature.

Field

data.srcip

The first feature, **failed-logins-srcip**, is designed to detect abnormal patterns in failed login attempts based on the source IP address of the attacker. The ML model uses the **`data.srcip`** field from Wazuh alerts to track which IP addresses are attempting authentication. By counting the number of occurrences (`count()` aggregation method), the model can identify unusual spikes in failed logins from a single source IP, which may indicate a brute-force attack originating from that host. The feature state is enabled to ensure it is actively monitored.


Not secure
~~https~~://192.168.56.16/app/anomaly-detection-dashboards#/create-ad/
W.
Anomaly detect...
Detectors
Create detector

failed-logins-agentip

Feature name

failed-logins-agentip

Enter a descriptive, unique name. The name must contain 1-64 characters. Valid characters are a-z, A-Z, 0-9, -(hyphen) and _(underscore).

Feature state

[✓] Enable feature

Find anomalies based on

Field value

Aggregation method

The aggregation method determines what constitutes an anomaly.

count()

E.g, if you choose min(), the detector focuses on finding anomalies based on the minimum values of your feature.

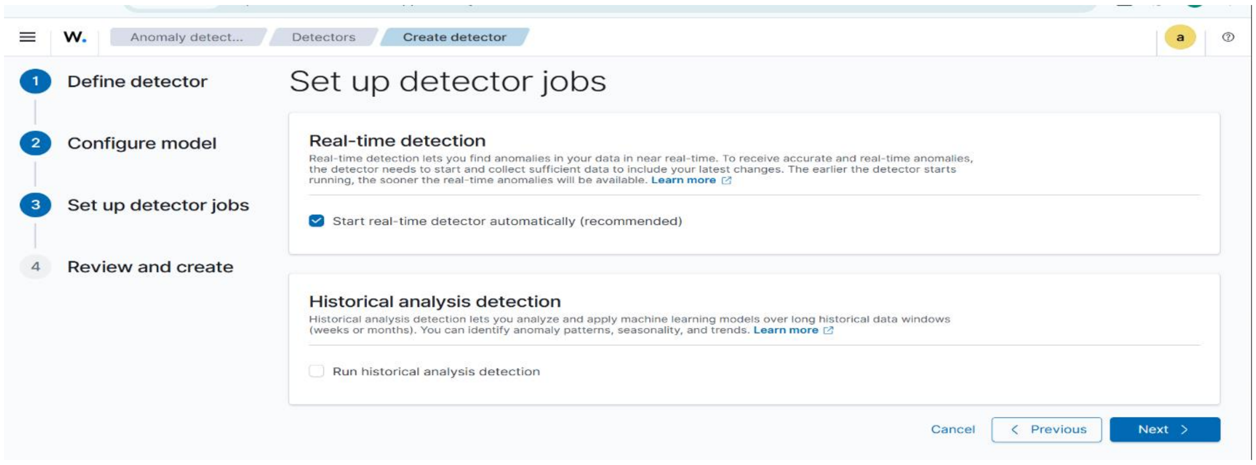
Field

agent.ip

Anomaly criteria

Acceptable difference between the expected and actual values

The second feature, **failed-logins-agentip**, focuses on monitoring abnormal login attempts targeting specific Wazuh agents. It uses the **`agent.ip`** field in the Wazuh alerts to track which agent is receiving authentication attempts. The aggregation method is set to `count()` to quantify login failures per agent. This allows the detector to recognize if one particular system is experiencing a suspiciously high number of failed login attempts, indicating potential targeted attacks or compromised accounts. This feature is also enabled to actively contribute to anomaly detection.



Set up detector jobs

Real-time detection

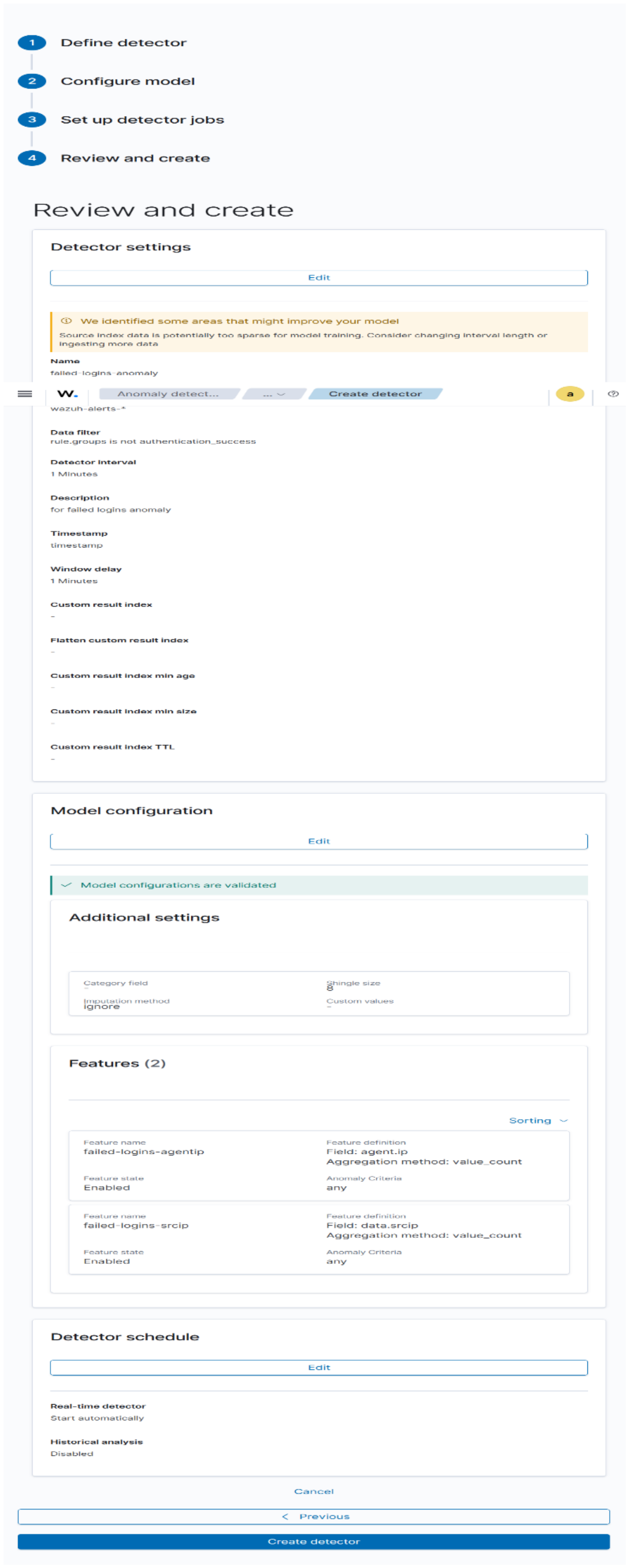
Real-time detection lets you find anomalies in your data in near real-time. To receive accurate and real-time anomalies, the detector needs to start and collect sufficient data to include your latest changes. The earlier the detector starts running, the sooner the real-time anomalies will be available. Learn more

[✓] Start real-time detector automatically (recommended)

Historical analysis detection

Historical analysis detection lets you analyze and apply machine learning models over long historical data windows (weeks or months). You can identify anomaly patterns, seasonality, and trends. Learn more

[ ] Run historical analysis detection



Review and create

Detector settings

We identified some areas that might improve your model

Source index data is potentially too sparse for model training. Consider changing interval length or ingesting more data

Name
failed-logins-anomaly

wazuh-alerts-*

Data filter
rule.groups is not authentication_success

Detector interval
1 Minutes

Description
for failed logins anomaly

Timestamp
timestamp

Window delay
1 Minutes

Custom result index
-

Flatten custom result index
-

Custom result index min age
-

Custom result index min size
-

Custom result index TTL
-

Model configuration

Model configurations are validated

Additional settings

Category field -	Shingle size 8
Imputation method ignore	Custom values -

Features (2)

Feature name failed-logins-agentip	Feature definition Field: agent.ip Aggregation method: value_count
Feature state Enabled	Anomaly Criteria any
Feature name failed-logins-srcip	Feature definition Field: data.srcip Aggregation method: value_count
Feature state Enabled	Anomaly Criteria any

Detector schedule

Real-time detector
Start automatically

Historical analysis
Disabled

Once the features and detector settings were configured, the system validated the detector configuration. If sufficient pre-existing data was available, Wazuh confirmed that **Detector settings are validated** and **Model configurations are validated**. In cases with sparse data, a warning may appear, suggesting the detector interval be extended to allow more data ingestion. After reviewing the settings, the **Create detector** button was clicked to finalize the configuration and initialize the detector. The detector then enters an **Initializing** state to begin baseline learning before real-time anomaly detection starts.

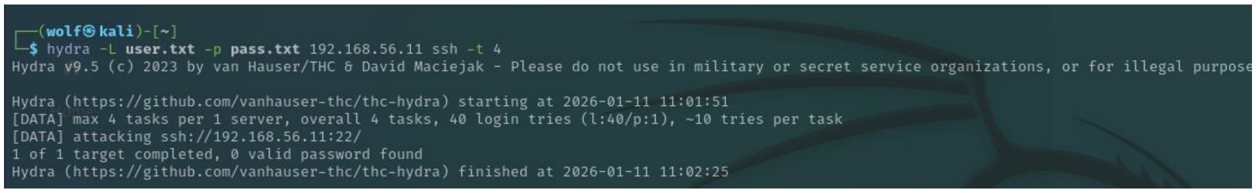
Brute-Force Attack from Kali Linux

To generate anomalous authentication behavior for testing the anomaly detector, a controlled brute-force SSH attack was launched from a Kali Linux machine against the Ubuntu system running the Wazuh agent.

Attack Tool: THC-Hydra
Attack Type: SSH brute-force authentication attempt
Attacker Machine: Kali Linux
Target Machine: Ubuntu (Wazuh Agent).

Command Used:

```
hydra -L user.txt -p pass.txt <UBUNTU_IP> ssh -t 4
```



```
┌──(wolf㉿kali)-[~]
└─$ hydra -L user.txt -p pass.txt 192.168.56.11 ssh -t 4
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purpose

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2026-01-11 11:01:51
[DATA] max 4 tasks per 1 server, overall 4 tasks, 40 login tries (l:40/p:1), ~10 tries per task
[DATA] attacking ssh://192.168.56.11:22/
1 of 1 target completed, 0 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2026-01-11 11:02:25
```

This attack generates a high volume of failed SSH login events in a short time window. These logs are collected by the Wazuh agent, forwarded to the Wazuh manager, indexed in OpenSearch, and used as input for machine-learning–based anomaly detection.

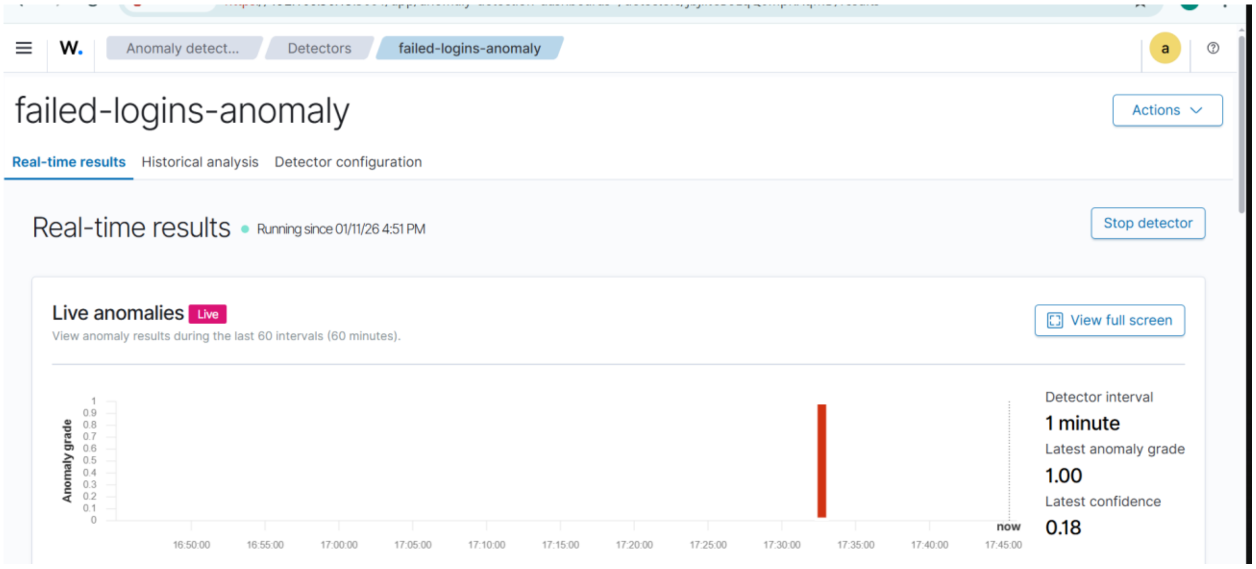
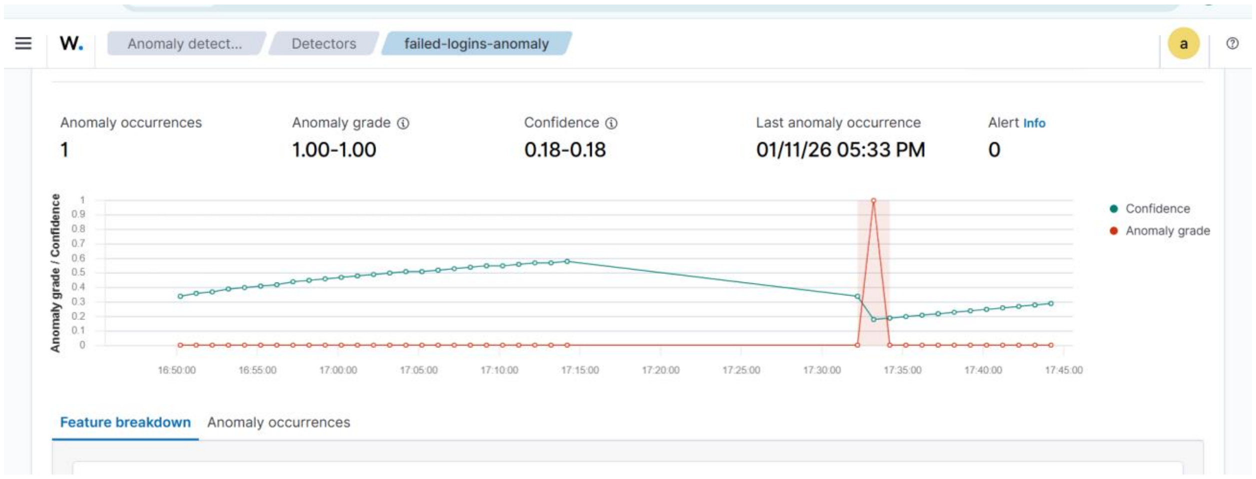


Figure 8: Live anomaly detected with anomaly grade 1.00 during real-time monitoring.

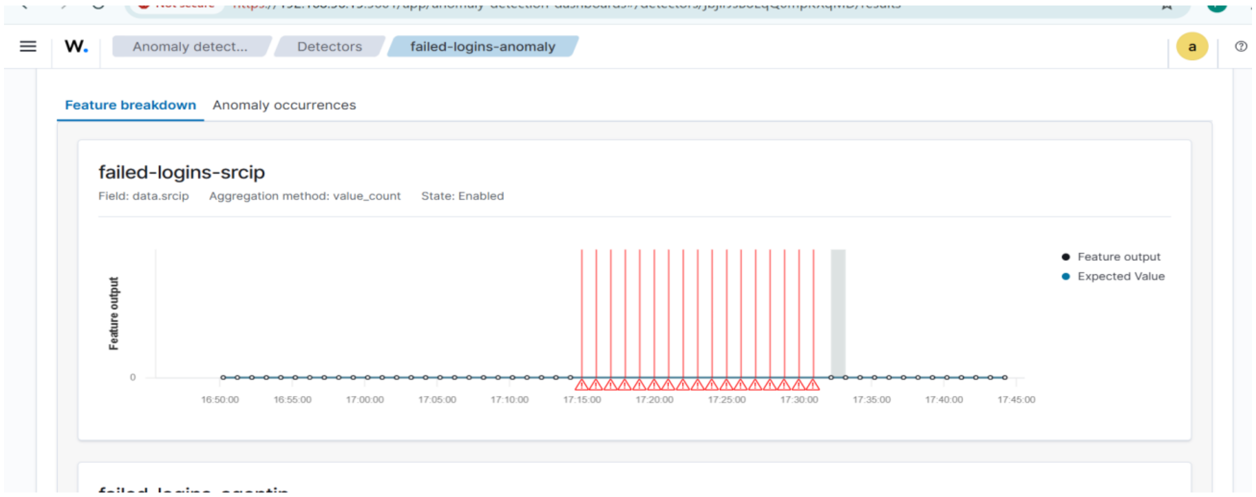
The anomaly detector successfully generated a live anomaly during real-time monitoring. A sharp spike in the anomaly grade was observed, reaching a maximum value of **1.00**, indicating a highly anomalous event compared to the learned baseline behavior. This spike corresponds to a sudden increase in failed login attempts within a short time window. Although the confidence value remained relatively low (0.18), this is expected in small-scale lab environments with limited historical data. The result confirms that the detector is actively evaluating security telemetry and detecting abnormal authentication behavior.



Anomaly occurrences	Anomaly grade	Confidence	Last anomaly occurrence	Alert Info
1	1.00-1.00	0.18-0.18	01/11/26 05:33 PM	0

Figure 9: Anomaly overview showing a sudden spike in anomaly grade with corresponding confidence score.

The anomaly overview section displays one detected anomaly occurrence. The timeline graph shows a sudden spike in the anomaly grade, highlighted in red, while the confidence score is shown in green. Prior to the anomaly, the detector maintained a stable baseline with an anomaly grade of zero. The sharp increase demonstrates the effectiveness of the machine learning model in detecting abrupt deviations in failed login behavior. The lower confidence value reflects limited historical data rather than a false detection.

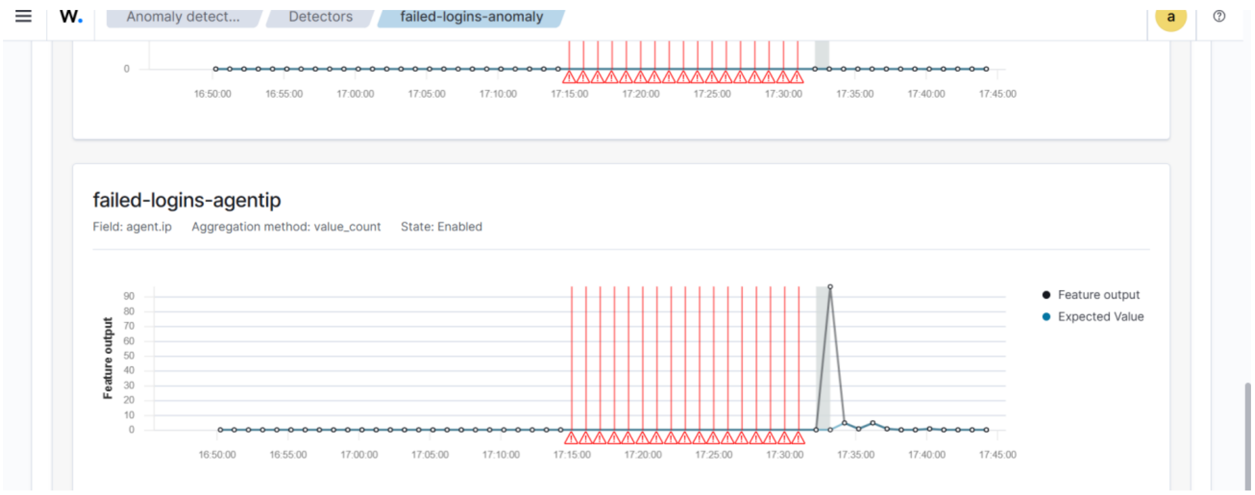


failed-logins-srcip

Field: data.srcip Aggregation method: value_count State: Enabled

Figure 10: Feature breakdown for failed-logins-srcip showing abnormal source IP activity.

The feature breakdown for **failed-logins-srcip** reveals abnormal behavior related to the source IP addresses involved in failed login attempts. The model detected repeated deviations where the observed feature output significantly exceeded the expected value. These deviations are marked with warning indicators, highlighting unusual activity from specific source IPs. This behavior is consistent with brute-force or automated login attempts originating from a single or limited set of IP addresses.



failed-logins-agentip

Field: agent.ip   Aggregation method: value_count   State: Enabled

The feature breakdown for **failed-logins-agentip** highlights anomalous authentication behavior associated with specific Wazuh agents. By analyzing the **agent.ip** field and counting failed login occurrences, the machine learning model identified abnormal spikes in authentication failures targeting a particular system. The observed feature output significantly exceeded the expected value derived from historical patterns. This behavior is security-relevant, as it may indicate a targeted brute-force attack or repeated unauthorized access attempts against a specific host within the environment.

SOC Value and Practical Impact

This AI-based anomaly detection approach:

- Enhances SOC visibility
- Reduces reliance on static rules
- Detects unknown and evolving attack patterns
- Enables early threat detection

It allows SOC analysts to prioritize incidents based on behavioral risk.

Conclusion

This task successfully demonstrated the implementation of AI-based anomaly detection in Wazuh using OpenSearch. By detecting abnormal failed login behavior at both source IP and agent IP levels, the system provided meaningful security insights beyond traditional rule-based monitoring.

The results validate the effectiveness of machine learning in strengthening SOC operations and improving threat detection accuracy in real-world environments.