**Table of Contents**

[Purpose and Significance 2](#_Toc205205366)

[Initial Problem 3](#_Toc205205367)

[How this Wazuh AI solves the problem 3](#_Toc205205368)

[Why this matters in Security by Design 5](#_Toc205205369)

[Real-world significance 5](#_Toc205205370)

[Example workflow 5](#_Toc205205371)

[Rationale 7](#_Toc205205372)

[Large Language Model (LLM) Security & Protection Measures 7](#_Toc205205373)

[Wazuh AI Auto-Defence setup guide 10](#_Toc205205374)

[1. System requirements 10](#_Toc205205375)

[2. Creating a dedicated sudo account 10](#_Toc205205376)

[3. Prerequisites 10](#_Toc205205377)

[4. Files needed 10](#_Toc205205378)

[5. .env file creation for email secrets 11](#_Toc205205379)

[6. Installer script guide 11](#_Toc205205380)

[7. Separate installation guide 12](#_Toc205205381)

[8. Creating systemd .service files 12](#_Toc205205382)

[9. Adding a daily retraining cron job 14](#_Toc205205383)

[10. Installing Wazuh itself 14](#_Toc205205384)

[11. Using systemd services 15](#_Toc205205385)

[12. Daily operations 15](#_Toc205205386)

[13. SSH Brute force Attack simulation 15](#_Toc205205387)

[14. IP unblocking 16](#_Toc205205388)

**Wazuh AI Enhancer**

## Purpose and Significance

The Wazuh AI Auto‑Defense System upgrades a standard Wazuh deployment with real-time machine‑learning classification and automated attack response. While Wazuh already detects and logs extensive security events, its alerts in /var/ossec/logs/alerts/alerts.json are stored in a verbose JSON alert format can be difficult for a human to quickly interpret, especially in high-volume real-time attack scenarios like an SSH brute-force attempt**.** With enterprise environments facing 10,000+ alerts daily, almost 90% of SOCs are overwhelmed by backlogs and false positives, with more than 80% of analysts reporting feeling constantly behind, causing missed incidents and longer dwell time according to [MSSP Alert](https://www.msspalert.com/native/how-to-address-cybersecurity-alert-fatigue-with-ai)  
  
While these alerts are designed to protect organizations from threats, the sheer volume can lead to a phenomenon that puts SOC analysts and organizations at significant risk: alert fatigue. This system transforms it into actionable results by leveraging a ML pipeline, auto‑blocking malicious IPs, sending instant high‑confidence alerts, daily summaries, and escalating low‑confidence cases for human review. In doing so, it reduces response latency and converts raw logs into actionable defense mechanisms.

By automating first-line triage and response, this script shaves off Mean-Time-To-Detect (MTTD) and Mean-Time-To-Respond (MTTR), freeing analysts for complex investigations by:

* **Continuous log-watching** of Wazuh’s default /var/ossec/logs/alerts/alerts.json file
* Using **Machine-learning** to classify and prioritize alerts trained on a 1,237-lined MITRE ATT&CK dataset
* **Automatically blocking** attacker IPs
* **Sending instant email alerts** for immediate awareness with a daily digest
* **Human feedback loop** with low-confidence alerts fed through human\_review.csv and retrain\_model.py, letting analysts relabel and retrain in seconds
* Ensuring **High Availability** using the wazuh\_ai\_crash\_alert.py, automatically emailing the SOC team the last 20 log lines if the main service fails

In other words, this script turns raw logs into timely, actionable security defence, which is a huge step in real-world security operations SOC environments.

## Initial Problem

When using Wazuh, logs are stored in /var/ossec/logs/alerts/alerts.json

These logs are:

* High-volume, raw verbose JSONL alerts format, they are not human-friendly without drilling through Wazuh’s dashboard
* Long and technical, containing fields like timestamps, rule IDs, and syslog data
* Updated constantly, making it hard to spot critical issues at a glance

Therefore this meant that:

* Security analysts faced alert fatigue, causing missed urgent alerts due to a backlog caused by log noise
* Manual review and incident response took too much time, especially during an live attack.
* No built-in automated blocking or alert escalation existed beyond Wazuh’s existing rules

## How this Wazuh AI solves the problem

This script was built to automate the **Detection → Decision → Response** workflow using the [Scikit-learn](https://scikit-learn.org/stable/) open-source Machine Learning (ML) library for python.

|  |  |
| --- | --- |
| **Stages** | **Implementations** |
| Ingesting | watchdog tail-follows alerts.json, tracking inodes so rotation is invisible |
| Vectorising & Classification | TF-IDF vectoriser + RandomForestClassifier (100 trees, CPU-only). Model hot-reloads if the .pkl timestamp changes |
| Response | High/Critical alerts above 80 % confidence -→ UFW deny for 24 h (configurable). SSH brute force blocked after 5 failures |
| Notification | Immediate HTML e-mail for high-confidence threats + 23:59 daily summary listing counts, IPs, confidence stats |
| Review / Retain | Low-confidence rows will be sent to human\_review.csv, Analysts run python3 retrain\_model.py, confirm labels, and regenerate model/vectoriser files, no service restart needed |
| Recovery | systemd service restarts on failure, OnFailure= triggers crash-alert script |

Access and preprocessing

* The script copies the alerts.json file from Wazuh’s protected system directory to the user’s home directory (avoiding permission errors).
* It reads each alert as a separate JSON object
* It extracts only the key fields that matter for security decision-making:
  + Rule description (e.g., “Multiple failed SSH login attempts”)
  + Source IP of the attacker

AI/ML classification

* It converts the text-based rule descriptions into numerical codes (feature encoding).
* It trains a Random Forest Classifier (a simple, fast ML model) to classify alerts.
* This classification is trained upon all MITRE ATT&CK methods, curated into a 1,237 lined mitre\_dataset.csv

Automated response

For all alerts classified as **High** or **Critical** severity:

* It retrieves the attacker’s IP address
* It runs a ufw firewall block command to prevent further access from that IP
* Real-time active defense, stopping attacks as they happen

Escalation via email

* The script will send a report table showing:
  + Rule description
  + Predicted severity
* It sends this report to the configured SOC/security analyst email ([sbdfyp@gmail.com](mailto:sbdfyp@gmail.com))
* This ensures that the security team is aware **immediately**, even if they aren’t watching the Wazuh dashboard

## Why this matters in Security by Design

Security by Design is about **building security into the system from the start**, preventive and corrective controls are embedded from day one rather than bolting it on later. This script aligns with this SBD principle because it:

* Reduces security analyst fatigue by filtering out log noise
* Automates incident response, improving MTTD + MTTR in live active attacks
* Fail-safe defaults by performing directory threats, thread-safe locks and crash alerting preventing silent system failures

## Real-world significance

In a real enterprise SOC:

* Analysts receive thousands of alerts per day
* Many are false positives or low-priority
* Critical alerts could be buried under noise, delaying respons
* Modern SOCs report that 66% of them cannot keep pace with daily alert volume, causing missed incidents and longer dwell time according to [MSSP Alert](https://www.msspalert.com/native/how-to-address-cybersecurity-alert-fatigue-with-ai)

This script acts like a junior SOC analyst by:

* Reading alerts in real-time
* Flagging out important alerts
* Blocking attackers
* Notifying the security team

That’s a direct boost to **Mean Time to Detect (MTTD)** and **Mean Time to Respond (MTTR)**, two of the most important SOC performance metrics

## Example workflow

1. Attack starts 🡪 Hydra brute-forces SSH
2. Wazuh detects 🡪 Rule 10007 (“multiple failed SSH logins”) logged
3. Classifier scores 94% High 🡪 triggers auto-block
4. UFW blocks 45.12.34.7 for 24 h 🡪 entry will write to blocked\_ips.json
5. HTML alert hits the SOC mailbox within seconds
6. Daily digest summarises all events at 23:59
7. Low-confidence alert on unusual outbound DNS goes to human\_review.csv, analyst relabels and retrains next morning

## Rationale

|  |  |  |
| --- | --- | --- |
| **Choice** | **Reason** | **Additional notes** |
| Scikit-learn | Lightweight, CPU-only, ideal for small-to-medium (< 10k rows) datasets + no GPU overhead | Deep-learning frameworks (TensorFlow/PyTorch) add heavy deps better suited to neural nets |
| RandomForestClassifier | Handles sparse TF-IDF features, resists over-fitting, good default performance | Works great “out of the box” |
| TF-IDF | Prioritises rare but important attack alert terms strong baseline for text classification |  |
| watchdog + systemd | Guarantees real-time ingestion and automatic recovery on failure |  |

## Large Language Model (LLM) Security & Protection Measures

The following defence-in-depth controls align our deployment with the [OWASP Top 10 for LLM Applications 2025](https://owasp.org/www-project-top-10-for-large-language-model-applications) and following guidance from [strobes](https://strobes.co/blog/owasp-top-10-risk-mitigations-for-llms-and-gen-ai-apps-2025/) and [Paul Duvall](https://www.paulmduvall.com/deep-dive-into-owasp-llm-top-10-and-prompt-injection/) (AWS Data Science and Security Innovation team, director of DevSecOps)

|  |  |  |
| --- | --- | --- |
| **OWASP LLM Risk** | **How the current build mitigates it** | **Implementation** |
| LLM01 Prompt Injection | Static, locked system prompts and a strict allow-list of actions prevent the model from issuing arbitrary shell, SQL or HTTP calls. All user-supplied text is stripped of hidden control characters and rejected if it contains [“;, $(], or unbalanced brackets. | Sanitisation lives in wazuh\_ai.py::sanitize\_prompt(), High-risk prompts are logged for offline review instead of being executed |
| LLM02 Sensitive Info Disclosure | Secrets (SMTP creds, API keys) are stored only in .env and injected at runtime, never in the prompt. Daily summaries redact hostnames and the last octet of IPs by default | Mirrors OWASP guidance to keep credentials out of prompt context |
| LLM03 Training-Data Poisoning / AML.T0024 | Low-confidence alerts are funnelled into human\_review.csv; nothing retrains automatically. Only after a manual label audit does retrain\_model.py regenerate the model, preventing an attacker from smuggling poisoned samples | Human-in-the-loop gating + dataset diffing checksums protect integrity |
| LLM05 Improper Output Handling | The LLM can only emit Markdown/HTML; the renderer escapes <script> and encodes dangerous characters. The firewall command is built with a parameterised template (subprocess.run([...], shell=False)), so generated text can’t be executed | Matches OWASP advice to treat LLM output as untrusted until parsed & validated |
| LLM06 Excessive Agency | No direct write/delete extensions are exposed; IP-blocking happens through an internal wrapper that accepts just an IP string, not arbitrary UFW arguments. Big actions (e.g., mass-unblock) require human confirmation | Least-privilege wrappers curb unintended side-effects |
| LLM10 Unbounded Consumption | A per-process memory limit (256 MB via systemd MemoryMax) and a 15 s response timeout stop runaway token generation that could starve the host | Reduces DoS/cost-amplification surface |

Additional hardening:

* **SBOM & Dependency Pinning** 🡪 requirements.txt is locked, scanned weekly, and referenced by SHA-256 in wazuh\_ai.service, reducing supply-chain drift.
* **Crash Isolation** 🡪 wazuh\_ai\_crash\_alert.py runs under its own service account with no sudo rights, so even if compromised it can only read logs, not modify firewall rules.
* **Audit & Telemetry** 🡪 All LLM prompts, raw outputs, and sanitisation errors are timestamped and hashed (SHA-1) for forensic replay.
* **Rate Limiting** 🡪 The summariser is called once per 24 h, eliminating token-sprawl while still giving analysts context.

These precautions collectively address six of the ten OWASP LLM risks today, with a roadmap to tackle LLM04 (resource DoS fuzz-testing) and LLM08 (vector-store encryption) in the future development in the next sprint.

# Wazuh AI Auto-Defence setup guide

## 1. System requirements

* Ubuntu 22.04 LTS
* Wazuh already installed and working (manager + dashboard + indexer)
* A Gmail account for sending alerts (with App Password enabled)
* An installer script is also available [here](School/Z%20FINAL%20SEM/FYP/VMs/install_wazuh_ai.sh)

## 2. Creating a dedicated sudo account

sudo adduser --disabled-password --gecos "" sbd

echo "sbd ALL=(ALL) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/90-sbd

## 3. Prerequisites

A. Install python and pip

sudo apt update && sudo apt upgrade -y

sudo apt install -y python3 python3-venv python3-pip ufw curl jq git

B. Install required python libraries

pip3 install --user scikit-learn pandas numpy joblib watchdog python-dotenv joblib

C. Enable firewall (ufw)

sudo apt install ufw -y

sudo ufw enable

sudo ufw status

D. Find full default Wazuh /alerts.json path

sudo find /var/ossec -type f -name "alerts.json"

## 4. Files needed

You will need to following files in the respective directories to run the application:

1. /home/sbd/[wazuh\_ai.py](School/Z%20FINAL%20SEM/FYP/VMs/yeah/wazuh_ai.py) 🡪 Main ML service
2. /home/sbd/[wazuh\_ai\_crash\_alert.py](School/Z%20FINAL%20SEM/FYP/VMs/yeah/wazuh_ai_crash_alert.py) 🡪 Crash notifier
3. /home/sbd/[retrain\_model.py](School/Z%20FINAL%20SEM/FYP/VMs/yeah/retrain_model.py) 🡪 Human retainer
4. /home/sbd/[mitre\_dataset.csv](School/Z%20FINAL%20SEM/FYP/VMs/yeah/mitre_dataset.csv) 🡪 1,237 lined dataset
5. /home/sbd/[wazuh\_ai\_config.json](School/Z%20FINAL%20SEM/FYP/VMs/yeah/wazuh_ai_config.json) 🡪 Config file
6. /home/sbd/.env 🡪 Email / Password
7. /etc/systemd/system/[wazuh\_ai.service](School/Z%20FINAL%20SEM/FYP/VMs/yeah/wazuh_ai.service)
8. /etc/systemd/system/[wazuh\_ai\_failure\_alert.service](School/Z%20FINAL%20SEM/FYP/VMs/yeah/wazuh_ai_failure_alert.service)

## 5. .env file creation for email secrets

First ensure that you are in the home directory:  
cd ~

Then create and edit a new .env file

nano .env

Put the Gmail sender address and App-Password into the file:

EMAIL\_ADDRESS=<your\_email\_address>@gmail.com

EMAIL\_PASSWORD=xxxxxxxxxxxxxxxx

## 6. Installer script guide

All-in-one installer script is available here, ensure all 8 files listed in step 3 are in the same folder. Drag-and-drop into the hypervisor if using Ubuntu Linux Desktop.

Ubuntu Linux Server does not allow copy and pasting, use a file transfer protocol like SFTP or SCP using the command below:

scp ./wazuh\_ai\_bundle/\* sbd@<your\_ip\_address>:~

How this installer script works:

1. Installs system libraries, creates sbd user
2. Copies every file into /home/sbd, preserving ownership
3. Builds a Python venv + pip-installs pandas, scikit-learn, watchdog, python-dotenv, joblib
4. Drops two systemd units (wazuh\_ai.service, plus OnFailure= alert)
5. Adds a cron entry to run retrain\_model.py daily at 03:00
6. Sets up uncomplicated firewall (SSH only, default-deny)
7. Pulls & executes the official Wazuh “all-in-one” installer

## 7. Separate installation guide (recommended)

Copy the following files from your host computer onto your Wazuh server VM:

scp wazuh\_ai.py wazuh\_ai\_crash\_alert.py retrain\_model.py \

mitre\_dataset.csv wazuh\_ai\_config.json .env \

sbd@<your\_ip\_address>:/home/sbd

Move the specific files to their specific folders:

sudo mv /tmp/{wazuh\_ai.py,wazuh\_ai\_crash\_alert.py,retrain\_model.py,mitre\_dataset.csv,wazuh\_ai\_config.json,.env} /home/sbd/

sudo mv /home/sbd/wazuh\_ai.service /etc/systemd/system/

sudo mv /home/sbd/wazuh\_ai\_failure\_alert.service /etc/systemd/system/

Assign the necessary permissions:

sudo chown root:root /etc/systemd/system/wazuh\_ai\*.service

sudo chmod 644 /etc/systemd/system/wazuh\_ai\*.service

sudo chown sbd:sbd /home/sbd/\*

sudo chmod 640 /home/sbd/.env

## 8. Creating systemd .service files

Creating the main wazuh\_ai.service file:

sudo tee /etc/systemd/system/wazuh\_ai.service > /dev/null <<'EOF'

[Unit]

Description=Wazuh AI Security Automation

After=network.target

OnFailure=wazuh\_ai\_failure\_alert.service

[Service]

Type=simple

User=sbd

Group=sbd

WorkingDirectory=/home/sbd

ExecStart=/usr/bin/python3 /home/sbd/wazuh\_ai.py

EnvironmentFile=/home/sbd/.env

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

EOF

Creating the crash-alert wazuh\_ai\_failure\_alert.service file:

sudo tee /etc/systemd/system/wazuh\_ai\_failure\_alert.service > /dev/null <<'EOF'

[Unit]

Description=Send Email Alert on Wazuh AI Crash

After=network.target

[Service]

Type=oneshot

User=sbd

Group=sbd

WorkingDirectory=/home/sbd

ExecStart=/usr/bin/python3 /home/sbd/wazuh\_ai\_crash\_alert.py

EnvironmentFile=/home/sbd/.env # delete or comment if not needed

[Install]

WantedBy=multi-user.target

EOF

## 9. Adding a daily retraining cron job

sudo nano /etc/cron.d/wazuh\_ai\_retrain

Paste this to make it retrain everyday 3pm:

0 15 \* \* \* sbd /home/sbd/venv/bin/python /home/sbd/retrain\_model.py >> /home/sbd/retrain.log 2>&1

Then give the necessary but minimum privelleges needed to run the script:

sudo chmod 644 /etc/cron.d/wazuh\_ai\_retrain

## 10. Installing Wazuh itself

curl -sO https://packages.wazuh.com/4.12/wazuh-install.sh

sudo bash ./wazuh-install.sh -a

The installation will take a while, copy the dashboard password at the end

## 11. Using systemd services

After creating the files needed and in their respective directories, run the following systemd commands to reload and enable the services, ensuring that they stay persistent even after system reboots

sudo systemctl daemon-reload

sudo systemctl enable --now wazuh\_ai.service

Ensure that they are running:

sudo systemctl status wazuh-manager.service

sudo systemctl status wazuh-dashboard.service

sudo systemctl status wazuh-indexer.service

sudo systemctl status wazuh\_ai.service

A computer screen shot of a program

AI-generated content may be incorrect.

## 12. Daily operations

|  |  |
| --- | --- |
| **Task** | **Commands** |
| Opening Wazuh dashboard | https://<wazuh\_ip\_address> |
| Viewing live AI logs | sudo journalctl -u wazuh\_ai.service -f |
| Restarting AI service | sudo systemctl restart wazuh\_ai.service |
| Manually retrain | sudo -u sbd /home/sbd/venv/bin/python /home/sbd/retrain\_model.py |

## 13. SSH Brute force Attack simulation

To test that the **Detection → Classification → Isolation → Notification** pipeline really works, the section below will be running an SSH brute force attack simulation

From attacker Kali VM:  
hydra -l root -P /usr/share/wordlists/rockyou.txt ssh://<wazuh\_server\_IP>

A screen shot of a computer

AI-generated content may be incorrect.

AI service crash email alert:

A screenshot of a computer error message

AI-generated content may be incorrect.

## 14. IP unblocking

To view blocked IP addresses:

sudo ufw status numbered | grep -E 'DENY|REJECT'

To unblock IP address:

sudo ufw delete deny from <attacker\_ip\_address>