SECURITY INFORMATION AND EVENT MANAGEMENT

Wazuh-Based Home Security Monitoring Lab

Deployment, Detection Engineering and Controlled Attack Simulation

Wazuh · Docker · Ubuntu · Hydra · Nmap

August 22, 2026

Contents

1 Introduction **2**

2 Objectives **2**

3 Lab Architecture **2**

4 Initial Setup and Troubleshooting **3**
4.1 Virtualisation and Network Configuration 3
4.2 Changing the Virtualisation Approach 4

5 Wazuh Deployment **4**
5.1 Central Components 4
5.2 Agent Deployment and Enrolment 6

6 File Monitoring **7**

7 SSH Brute-Force Testing **7**

8 Network Scanning **8**

9 Docker Monitoring **9**

10 Problems Encountered **10**

11 Results **11**

12 Lessons Learned **12**

13 Conclusion **13**

1 Introduction

This project involved building a small home security monitoring environment using **Wazuh** as the central Security Information and Event Management (SIEM) platform. The objective was not only to deploy Wazuh, but also to understand how security events are collected, detected and investigated.

The project developed through several iterations. Initial deployment attempts involved virtualisation and networking issues across two different virtual machine platforms, and were resolved only by abandoning virtualisation and building the lab on physical machines. Wazuh was eventually deployed using Docker, with separate monitored endpoints connected to the central Wazuh environment.

The lab was then used to generate controlled security and system events in order to verify that Wazuh could detect and display them.

> **Scope of the project.** The deliverable is not simply a running Wazuh installation. It is a working end-to-end monitoring pipeline in which activity on a monitored endpoint is collected by an agent, processed centrally, converted into an alert, and finally investigated through a dashboard.

2 Objectives

The main objectives of the project were:

- Deploy a functional Wazuh monitoring environment.
- Understand the relationship between Wazuh managers and agents.
- Monitor activity from separate endpoints.
- Configure file monitoring.
- Generate controlled security events.
- Test SSH authentication monitoring using Hydra.
- Test network visibility using Nmap.
- Monitor Docker-related activity.
- Investigate the alerts and events generated by Wazuh.
- Document problems encountered during deployment and how they were resolved.

3 Lab Architecture

The final architecture consisted of a central Wazuh environment running through Docker and separate endpoints running Wazuh agents. Every host in the diagram below is a **physical machine**: as described in Section 4, virtualisation was removed from the lab after two virtual machine platforms proved unreliable on the network.

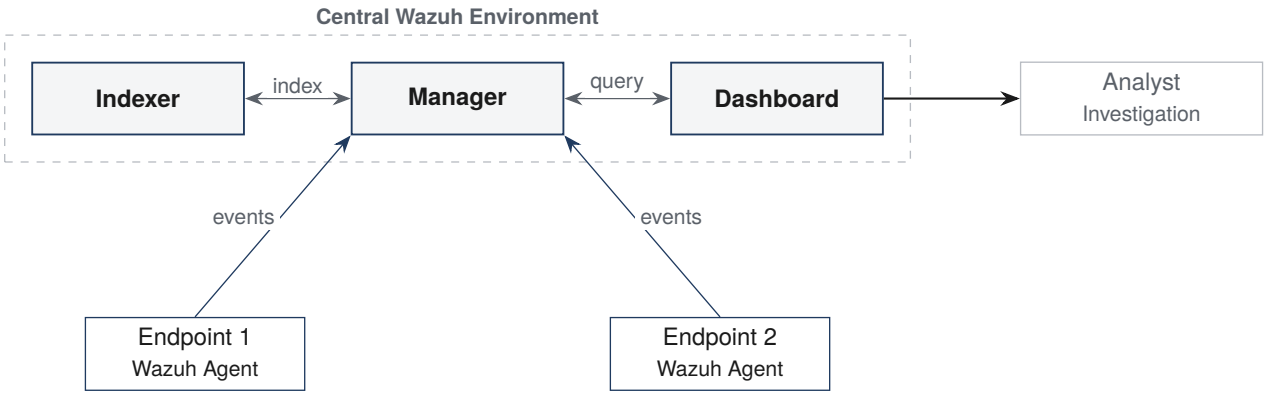


Figure 1. Final lab architecture. The central Wazuh components run as containers on the host, while monitored endpoints run only the Wazuh agent. All hosts are physical machines; the only virtualisation involved is Docker on the central host.

The separation between the central Wazuh components and the monitored endpoints allowed security events to be generated on an endpoint and then observed through the central dashboard.

An additional agent was also configured on another endpoint using documentation for that platform, providing a second example of endpoint monitoring.

Table 1. Role of each component in the completed lab.

Component	Responsibility
Wazuh Manager	Receives agent events, applies decoders and rules, generates alerts.
Wazuh Indexer	Stores and indexes alerts and events so they can be searched.
Wazuh Dashboard	Presents alerts, agent status and search interfaces to the analyst.
Wazuh Agent	Runs on the monitored endpoint; collects logs, file integrity data and Docker events.
Endpoint	The monitored system on which controlled activity is generated.

4 Initial Setup and Troubleshooting

4.1 Virtualisation and Network Configuration

The initial setup used **QEMU** and **virt-manager**. The first major problem occurred while attempting to configure the virtual machine's network connection. The expected default network configuration was not available, preventing the virtual machine from obtaining the intended network connectivity.

System logs were investigated using systemd tools. The logs showed missing dependencies, including `dnsmasq` and `dmidecode`.

The first attempt to resolve the problem by installing `dnsmasq` did not solve the underlying issue. This became an early example of troubleshooting a symptom rather than the actual cause.

The required network configuration was eventually recreated using documentation associated with the virtualisation software. After configuring the network and enabling it to start automatically, the virtual machine was able to use the network correctly, although, as the next subsection describes, this was not the end of the environment's networking problems.

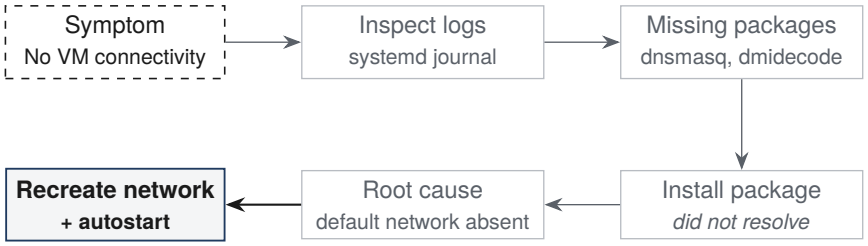


Figure 2. Troubleshooting path for the virtual machine networking failure. The first fix addressed a symptom; the working fix addressed the missing network definition.

4.2 Changing the Virtualisation Approach

Recreating the network definition restored connectivity for that particular virtual machine, but the **virt-manager** environment continued to demand more setup and maintenance effort than the project warranted.

GNOME Boxes was tried next, chosen specifically because it required far less manual configuration to produce a usable virtual machine. It did prove quicker to set up, but the environment then ran into a networking problem of its own: the guest could not reach the internet reliably, which blocked package installation and agent registration. The same class of failure had therefore appeared twice, in two different virtualisation stacks, for two unrelated underlying reasons.

Rather than continue debugging a layer that was not part of the project's subject matter, virtualisation was removed from the lab entirely. The environment was rebuilt on **physical machines**: the central Wazuh components run in Docker on one host, and the Wazuh agents are installed directly on the operating systems of the monitored machines. Networking then behaved exactly as the physical home network did, and no further connectivity problems occurred.

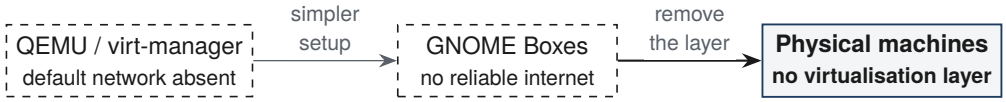


Figure 3. Migration away from virtualisation. Each environment failed on networking for a different reason; removing the virtualisation layer altogether resolved the problem rather than relocating it.

The lab environment itself therefore became a significant source of problems before any security tooling was installed. It also showed that persisting with a troublesome component is not always the right decision: two attempts were spent working around the virtualisation layer before it became clear that the layer was optional for this project and could simply be dropped.

5 Wazuh Deployment

5.1 Central Components

Wazuh was initially installed as a central SIEM environment. The dashboard was successfully accessed after resolving issues with how the environment was being reached.

A problem was then discovered with running both the Wazuh manager and the Wazuh agent on the same hosting environment, because the packages conflicted. This required changing the architecture.

The central Wazuh components were subsequently moved into Docker. This allowed the Wazuh environment to run on the existing host while separate endpoints could run the Wazuh agent.

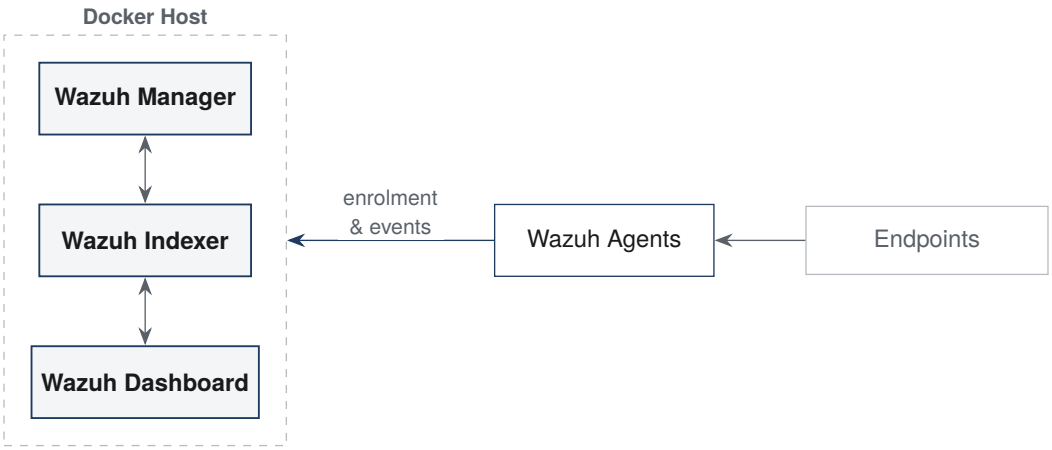


Figure 4. Containerised deployment. Manager, indexer and dashboard run as containers on the Docker host; agents run natively on the monitored endpoints.

The Docker deployment initially appeared to work, but the dashboard did not display the expected data (Figure 5). Investigation eventually identified a **credentials mismatch** between components in the Docker configuration. After resolving this issue, the dashboard began displaying the collected events.

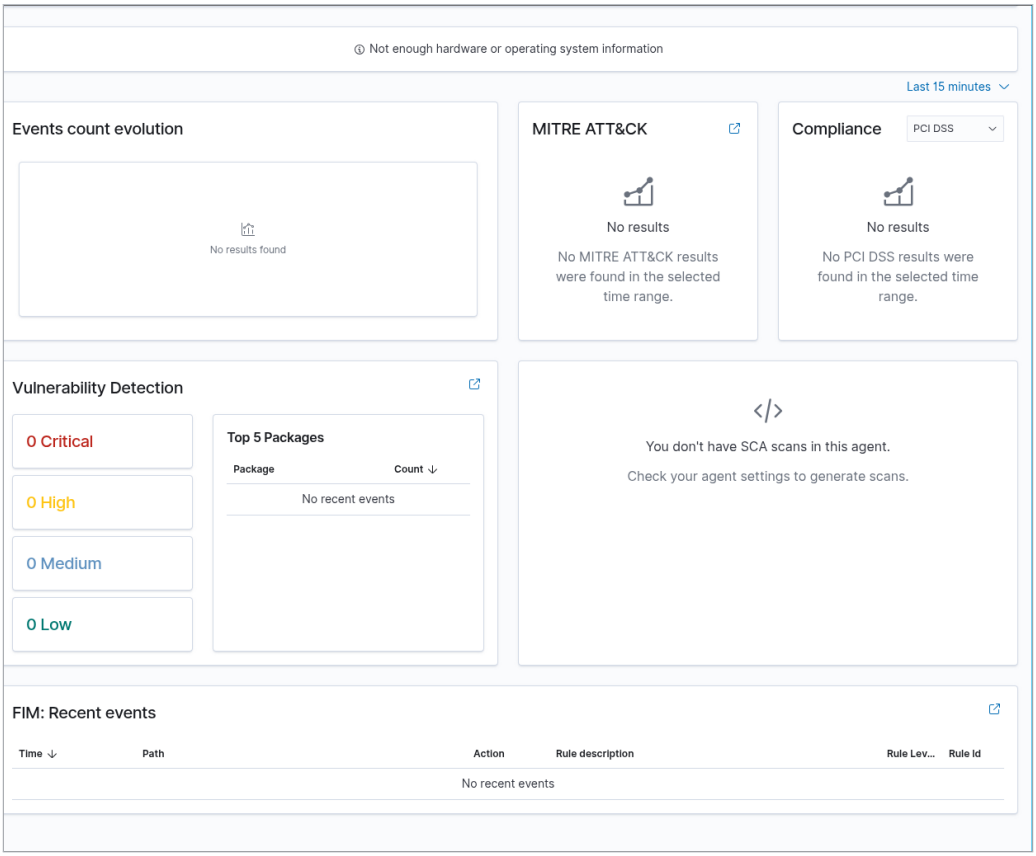


Figure 5. The dashboard before the credentials mismatch was resolved. Every panel reports an empty result set: no events, no MITRE ATT&CK or PCI DSS matches, no vulnerability or FIM records, despite the agent showing as connected.

> **Why the dashboard was empty**
>
> The agent reported as connected and the containers were running, so the failure was not visible from the component status alone. Because the dashboard could not authenticate to the indexer, alerts were being generated and stored but never returned to the interface. Aligning the credentials across the Docker configuration restored the data flow.

5.2 Agent Deployment and Enrolment

With the central environment working, endpoints were added through the dashboard's guided **Deploy new agent** flow. The flow builds the installation and enrolment command from three inputs: the package matching the endpoint's operating system and CPU architecture, the address of the Wazuh server, and an optional agent name.

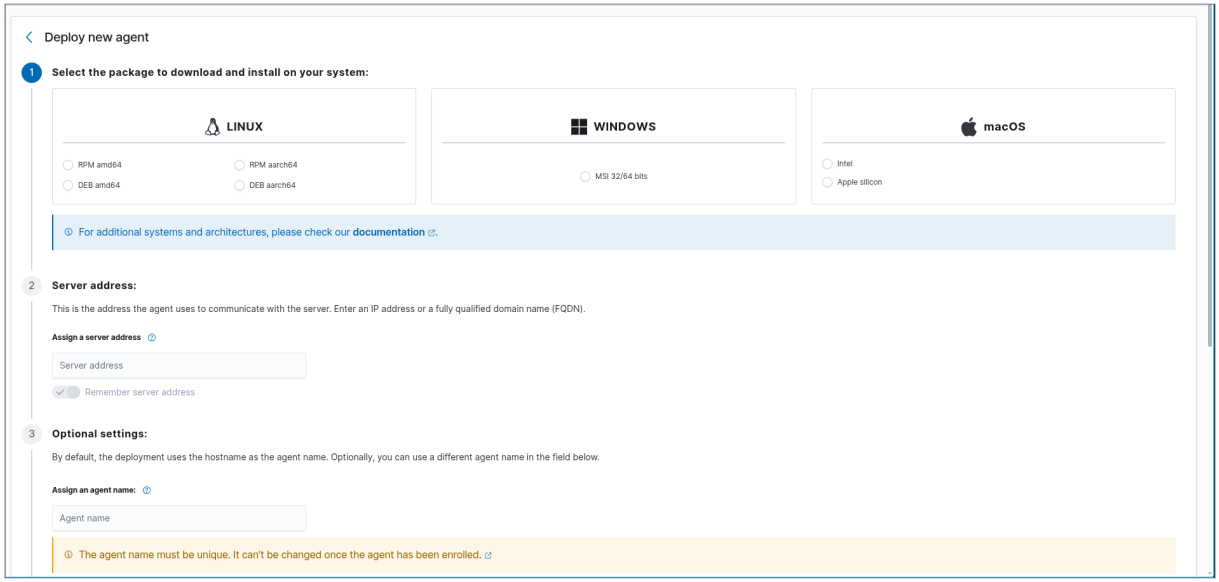


Figure 6. The agent deployment flow in the Wazuh dashboard. The package list covers Linux (RPM and DEB, amd64 and aarch64), Windows and macOS; the server address and agent name determine where the agent reports and how it is identified once enrolled.

Three details of this screen turned out to matter in practice:

- **Package selection.** The correct package depends on both the distribution's package format and the CPU architecture. This is why the second endpoint could not simply reuse the process followed for the first, because a different platform required a different package and a different installation procedure.
- **Server address.** This is the address the agent uses to reach the manager, not the address of the endpoint itself. Misreading this field was the cause of the agent configuration problem described in Section 10: the agent was pointed at its own host and therefore never reached the manager.
- **Agent name.** The name defaults to the endpoint's hostname and must be unique. It cannot be changed after enrolment, which is why the monitored endpoint appears throughout this report under the fixed name `watchman-protocol`.

Once the generated command was run on the endpoint, the agent registered with the manager and began forwarding events to the central environment.

6 File Monitoring

File monitoring was configured as one of the first tests of the completed Wazuh environment. A dedicated directory was created for testing, and changes within that monitored location were then used to verify that the Wazuh agent could detect and report file activity.

This provided the first confirmation that the complete collection pipeline was functioning.

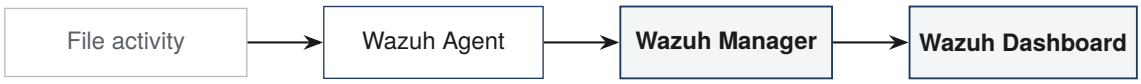


Figure 7. File integrity monitoring pipeline verified during the first test.

The successful appearance of these events in the dashboard confirmed that the monitoring configuration was functioning correctly.

7 SSH Brute-Force Testing

One of the main security tests involved simulating an SSH brute-force attack using **Hydra**. The test generated multiple authentication attempts against the monitored endpoint, and Wazuh successfully detected the resulting authentication failures.

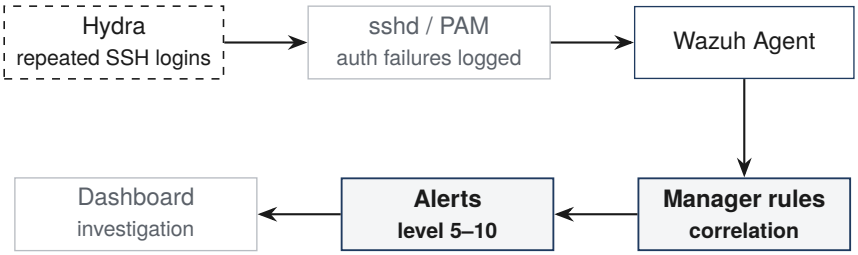


Figure 8. Path taken by a simulated brute-force attempt, from Hydra through to analyst investigation.

The dashboard displayed events including:

Table 2. Representative rule descriptions observed during the Hydra test.

Rule description	Level	Interpretation
`sshd`: authentication failed	5	Single failed SSH login attempt
`unix_chkpwd`: Password check failed	5	PAM password verification failure
`syslog`: User authentication failure	5	Generic authentication failure
Maximum authentication attempts exceeded	8	Session exceeded the permitted retries
`syslog`: User missed the password more than one time	10	Correlated repeated failures

Several alerts were generated with different severity levels, demonstrating that Wazuh was able to identify repeated authentication activity rather than only isolated failures.

watchman-protocol	unix_chkpwd: Password check failed.	5
watchman-protocol	unix_chkpwd: Password check failed.	5
watchman-protocol	unix_chkpwd: Password check failed.	5
watchman-protocol	sshd: authentication failed.	5
watchman-protocol	sshd: authentication failed.	5
watchman-protocol	sshd: authentication failed.	5
watchman-protocol	syslog: User missed the password more than one time	10
watchman-protocol	syslog: User missed the password more than one time	10
watchman-protocol	syslog: User authentication failure.	5
watchman-protocol	syslog: User authentication failure.	5
watchman-protocol	Maximum authentication attempts exceeded.	8
watchman-protocol	Maximum authentication attempts exceeded.	8
watchman-protocol	sshd: authentication failed.	5
watchman-protocol	sshd: authentication failed.	5
watchman-protocol	unix_chkpwd: Password check failed.	5

Figure 9. SSH brute-force detection in Wazuh. The agent `watchman-protocol` reports individual `sshd` and `unix_chkpwd` failures at level 5, alongside higher-severity correlated rules at levels 8 and 10.

The screenshot provides evidence that the attack generated multiple authentication events and that Wazuh successfully collected and classified them.

> **Why this matters.** This test demonstrated the practical value of centralised security monitoring: individual failed authentication attempts can be examined collectively rather than being treated as isolated events. The level 10 rule only exists because the manager correlated several level 5 events from the same source.

The Hydra test and the resulting logs were documented during the lab process.

8 Network Scanning

Nmap was also used to perform network scanning against the monitored environment.

The initial scan did not behave as expected. Investigation showed that the scanning method needed to be changed for the network environment being used. A **SYN scan** was subsequently used, allowing the scan to proceed successfully.

Figure 10. Adjusting the scanning technique after the default host-discovery method failed in this network environment.

This was another example of the lab requiring investigation rather than simply executing a command from a tutorial. The test demonstrated the importance of understanding how network tools determine whether a target is reachable, and how different scanning techniques can affect the result.

9 Docker Monitoring

Docker activity was also monitored through Wazuh. The test generated several container lifecycle and networking events.

Table 3. Docker events collected by the Wazuh agent.

Resource	Action	Reported event
Container	create	Container created
Container	start	Container started
Container	stop	Container stopped
Container	kill / die	Container received a termination action
Container	destroy	Container destroyed
Network	create	Docker network created
Network	connect	Container attached to a Docker network
Network	disconnect	Container detached from a Docker network
Network	destroy	Docker network deleted

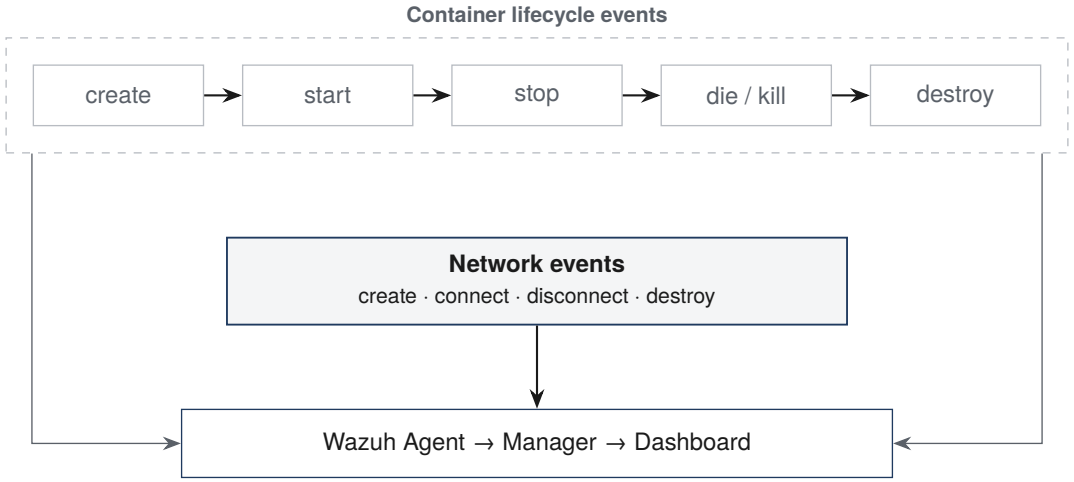
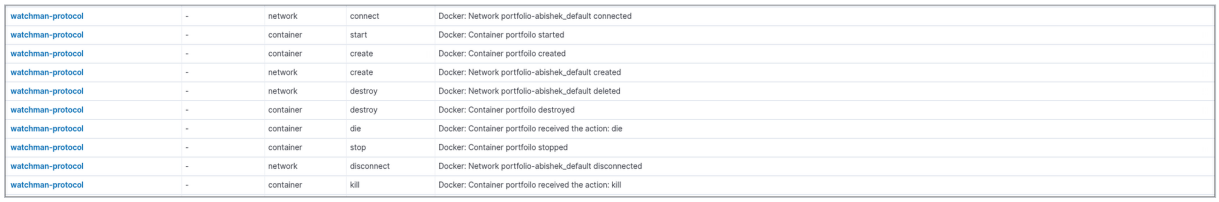


Figure 11. Container lifecycle and network events captured through Docker monitoring.



watchman-protocol	-	network	connect	Docker: Network portfolio-abishek_default connected
watchman-protocol	-	container	start	Docker: Container portfolio started
watchman-protocol	-	container	create	Docker: Container portfolio created
watchman-protocol	-	network	create	Docker: Network portfolio-abishek_default created
watchman-protocol	-	network	destroy	Docker: Network portfolio-abishek_default deleted
watchman-protocol	-	container	destroy	Docker: Container portfolio destroyed
watchman-protocol	-	container	die	Docker: Container portfolio received the action: die
watchman-protocol	-	container	stop	Docker: Container portfolio stopped
watchman-protocol	-	network	disconnect	Docker: Network portfolio-abishek_default disconnected
watchman-protocol	-	container	kill	Docker: Container portfolio received the action: kill

Figure 12. Docker activity detected by Wazuh, showing container and network actions for the monitored deployment on the `watchman-protocol` agent.

The screenshot demonstrates that Wazuh was able to provide visibility into container activity rather than only traditional operating-system events.

This is particularly useful in environments where applications and services are deployed through containers, because unexpected container creation, termination or network changes could potentially represent important security events.

10 Problems Encountered

The project involved several problems that required investigation.

Virtual machine networking

The initial virtual machine could not obtain the expected network connectivity. Investigation of system logs revealed missing dependencies and an absent network configuration, which was recreated from the available documentation. The virtualisation layer kept causing problems regardless: a second attempt using GNOME Boxes was much faster to set up but could not reach the internet reliably. The lab was ultimately rebuilt on physical machines, which removed the virtualisation layer and with it this entire class of problem.

Wazuh deployment architecture

The initial architecture attempted to run the Wazuh manager and agent together. This was changed after discovering that the packages conflicted. The architecture was redesigned so that the central Wazuh components and the monitored endpoints were separated, as sketched in Figure 13.

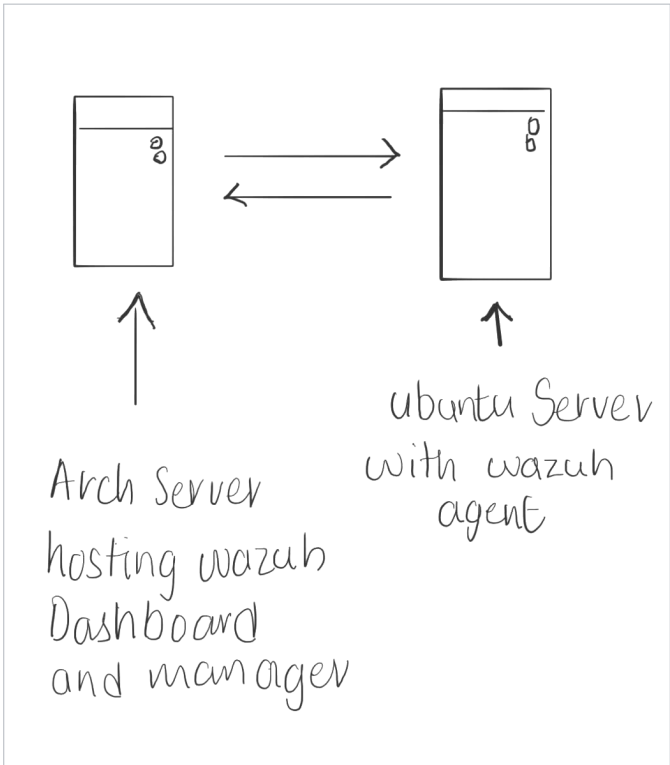


Figure 13. Working sketch of the redesigned deployment: an Arch Linux host running the Wazuh manager and dashboard, and a separate Ubuntu server running only the Wazuh agent. Splitting the roles across two hosts removed the package conflict that made the original single-host design unworkable.

Agent configuration

The first agent configuration used the wrong destination for the Wazuh manager. This was corrected after understanding that the agent must communicate with the machine hosting the Wazuh manager rather than using its own address.

Dashboard data

The Wazuh dashboard initially showed no monitoring data even though the agent appeared to be connected. Investigation eventually identified a credentials mismatch between components in the Docker deployment. Correcting the configuration restored the expected data flow.

Endpoint agent installation

An additional agent was later installed using a different installation method. This required following the platform documentation rather than reusing the installation process used for Ubuntu.

Table 4. Summary of problems, root causes and resolutions.

Problem	Root cause	Resolution
VM networking	Default virtual network definition missing; virtualisation layer unreliable in general	Recreated the network, then tried GNOME Boxes, then moved to physical machines
Manager + agent conflict	Packages not intended to coexist on one host	Separated central components from endpoints
Agent points to itself	Misunderstanding of manager address	Configured the agent to target the manager host
Empty dashboard	Credentials mismatch between Docker components	Aligned credentials across the deployment
Second endpoint	Different platform, different installer	Followed the platform-specific documentation

11 Results

The completed lab successfully demonstrated several capabilities of Wazuh.

Table 5. Test results.

Test	Result
File monitoring	Successful
SSH brute-force simulation	Successful
Authentication failure detection	Successful
Network scanning	Successfully performed
Docker event monitoring	Successful
Endpoint agent monitoring	Successful
Central dashboard visibility	Successful

The most significant result was that the complete monitoring pipeline could be demonstrated using controlled activity.

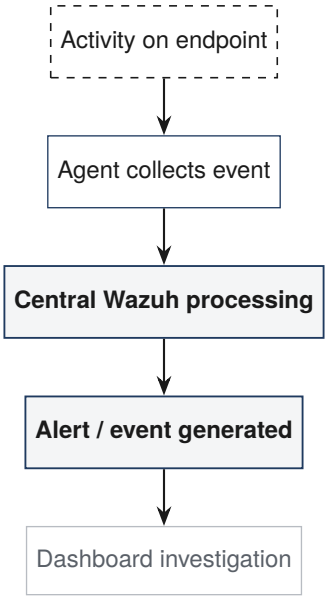


Figure 14. The verified end-to-end monitoring pipeline.

12 Lessons Learned

The project demonstrated that deploying a security monitoring platform involves considerably more than installing the software itself.

Troubleshooting requires understanding the underlying system

Installing a package or dependency based only on an error message does not necessarily resolve the actual problem.

Architecture matters

The relationship between the manager, agents, containers and endpoints needs to be understood before configuring the system.

Logs are essential during troubleshooting

System and application logs provided information that was not visible from the original symptoms.

Documentation is more useful when understood rather than copied

Several problems were resolved only after investigating the relevant documentation and understanding what each component was expected to do.

Security monitoring needs real events to be meaningful

A functioning dashboard alone does not demonstrate a useful security system. Controlled testing was necessary to verify that Wazuh could actually detect relevant activity.

Failed attempts are part of the learning process

The project required several changes to the original plan before reaching a functional architecture.

A problematic component can sometimes be removed rather than fixed

Two virtualisation platforms were tried before it became clear that the virtualisation layer was not required for this project at all. Rebuilding on physical machines removed the problem instead of relocating it.

13 Conclusion

This project resulted in a functional home security monitoring environment using Wazuh.

The project progressed from virtual machine and networking problems through Wazuh deployment, agent configuration, troubleshooting and finally security testing. The networking problems were resolved only by dropping virtualisation in favour of physical machines. The completed environment successfully collected and displayed events generated through file activity, SSH brute-force testing, network scanning and Docker activity.

The most valuable part of the project was not simply getting Wazuh running. The troubleshooting process demonstrated how different components interact and how security events move from an endpoint into a central monitoring platform.

The project also provided practical experience with security monitoring, event analysis, Linux troubleshooting, Docker, networking, endpoint monitoring and controlled attack simulation.

> The final result is therefore more than a Wazuh installation. It is a small, functional security monitoring lab that can be used as a foundation for future security experiments.