Linux Intrusion Detection with OSSEC & Wazuh

Deploying, Configuring, and Managing Host-Based Intrusion Detection Systems

Preface

Every Linux server connected to the internet is a target. Whether it hosts a simple web application or underpins critical enterprise infrastructure, the moment it goes live, it faces a relentless barrage of automated scans, brute-force attempts, and increasingly sophisticated intrusion campaigns. The question is never *if* your Linux systems will be probed—it's whether you'll know when it happens, and whether you'll be ready to respond.

This book exists to make sure the answer is *yes*.

Why This Book

Linux Intrusion Detection with OSSEC & Wazuh is a practical, hands-on guide to deploying, configuring, and managing host-based intrusion detection systems (HIDS) on Linux. It is written for system administrators, security engineers, and DevOps professionals who are responsible for protecting Linux environments and who want to move beyond reactive firefighting toward proactive, intelligence-driven defense.

OSSEC and Wazuh are among the most widely deployed open-source HIDS platforms in the world, and for good reason. They are powerful, flexible, and purpose-built for the kinds of threats that Linux systems face daily—unauthorized file modifications, privilege escalation attempts, rootkit installations, log tampering, and far more. Yet despite their capabilities, many organizations deploy these tools with default configurations and never unlock their full potential. This book aims to change that.

What You Will Learn

The journey begins with foundational concepts—what intrusion detection is, how OSSEC and Wazuh are architected, and why host-based detection is an essential layer in any Linux security strategy. From there, we move into *doing*: installing OSSEC and Wazuh on Linux, configuring agents and log collection, setting up file integrity monitoring, and tuning rules and alerts to match the realities of your environment.

But detection without action is just noise. That's why significant attention is devoted to **active response**—automatically blocking threats at the Linux firewall level—as well as integration with SIEM platforms, custom rule development, and threat intelligence feeds. You'll learn not only how to detect suspicious activity on your Linux hosts but how to build workflows that turn detection into decisive incident response.

The later chapters address the challenges that come with maturity and scale: hardening your IDS infrastructure itself, scaling Wazuh across production Linux environments, and building repeatable incident response processes. The final chapter offers something more personal—a roadmap for evolving from a Linux system administrator into a security engineer, with the skills and mindset that transformation requires.

How This Book Is Structured

Chapters 1 through 4 lay the groundwork with IDS fundamentals and OSSEC essentials. Chapters 5 through 9 dive deep into Wazuh deployment, monitoring, and active response on Linux. Chapters 10 through 12 expand your detection capabilities through integrations, custom rules, and threat intelligence. Chapters 13 through 16 focus on production readiness, security hardening, incident response, and career growth. The appendices provide quick-reference cheat sheets, rule examples, checklists, and templates that you'll return to long after your first read.

Each chapter is designed to be *actionable*. Concepts are illustrated with real configurations, real rules, and real Linux command-line examples. You can read the book cover to cover or jump directly to the chapters most relevant to your current needs.

Acknowledgments

This book would not exist without the extraordinary open-source communities behind OSSEC and Wazuh. Their commitment to building freely available, enterprise-grade security tools for Linux and beyond has made the world measurably safer. I am also deeply grateful to the security researchers, Linux administrators, and incident responders whose published work, blog posts, and conference talks have shaped my understanding of host-based detection. Finally, my thanks go to every reader who picks up this book with the intention of making their Linux systems harder to compromise—you are the front line, and your work matters.

Let's get started.

Bas van den Berg

Table of Contents

Chapter	Title	Page
1	Understanding Intrusion Detection Systems	6
2	OSSEC and Wazuh Architecture	18
3	OSSEC Installation and Initial Setup	34
4	OSSEC Rules and Alerts	49
5	Installing Wazuh Server	65
6	Wazuh Agents and Log Collection	79
7	File Integrity Monitoring (FIM)	98
8	Detecting Suspicious Activity	115
9	Configuring Active Response	132
10	Integrating with Firewalls and SIEM	147
11	Custom Rule Development	164
12	Threat Intelligence Integration	182
13	Scaling Wazuh in Production	201
14	Securing the IDS Infrastructure	217
15	Building an Incident Response Workflow	234
16	From System Administrator to Security Engineer	250
App	OSSEC Configuration Cheat Sheet	267
App	Wazuh Rule Examples	286
App	File Integrity Monitoring Checklist	300
App	Incident Response Template	318
App	Linux Security Learning Path	330

Chapter 1: Understanding Intrusion Detection Systems

The security of a Linux system is not something that can be achieved through a single tool or a one-time configuration. It is an ongoing process, a continuous vigil that demands awareness, intelligence, and the right set of tools. Among the most critical components of any serious Linux security infrastructure is the Intrusion Detection System, commonly referred to as an IDS. Before we dive into the specifics of OSSEC and Wazuh in later chapters, it is essential that we build a thorough and grounded understanding of what intrusion detection systems are, why they matter in a Linux environment, how they function, and what types exist. This foundational knowledge will serve as the bedrock upon which every subsequent chapter of this book is built.

Linux, by its very nature, is the backbone of modern server infrastructure. From web servers running Apache or Nginx, to database clusters powered by PostgreSQL or MariaDB, to containerized microservices orchestrated by Kubernetes, Linux is everywhere. Its prevalence makes it both a powerful platform and an attractive target. Attackers know that compromising a single Linux server can yield access to sensitive data, lateral movement across networks, and persistent footholds in enterprise environments. An intrusion detection system is, in many ways, the watchful eye that never sleeps, monitoring every corner of your Linux infrastructure for signs of unauthorized access, policy violations, and malicious activity.

What Is an Intrusion Detection System

An Intrusion Detection System is a software application or, in some cases, a hardware appliance that monitors a system or network for malicious activities or policy violations. When suspicious behavior is detected, the IDS generates alerts that are sent to administrators or collected by a centralized security information and event management (SIEM) system. The fundamental purpose of an IDS is detection, not prevention. It observes, analyzes, and reports. Think of it as a sophisticated alarm system for your Linux infrastructure. It does not lock the doors or block the windows. Instead, it tells you when someone has tried to open them, when a window has been broken, or when someone is moving through your house who should not be there.

In a Linux context, an IDS might monitor system log files such as `/var/log/auth.log` or `/var/log/secure` for repeated failed login attempts. It might watch configuration files like `/etc/passwd`, `/etc/shadow`, or `/etc/sudoers` for unauthorized modifications. It might analyze network traffic arriving at a Linux firewall for patterns that match known attack signatures. The scope of what an IDS can monitor is broad, and the sophistication of modern IDS tools like OSSEC and Wazuh makes them indispensable for any Linux administrator who takes security seriously.

It is important to distinguish between an IDS and an Intrusion Prevention System (IPS). While an IDS passively monitors and alerts, an IPS takes active measures to block or prevent detected threats. Some tools, including Wazuh, can function in both capacities, but the conceptual distinction remains important. An IDS tells you that something bad is happening. An IPS tries to stop it from happening. In practice, many Linux security architectures employ both, often within the same toolset.

Why Linux Systems Need Intrusion Detection

There is a persistent and dangerous myth that Linux systems are inherently secure and do not require the same level of monitoring as other operating systems. While it is true that Linux benefits from a robust permissions model, a strong community of security-minded developers, and a transparent open-source codebase, none of these qualities make a Linux system immune to attack. Misconfigurations, unpatched vulnerabilities, weak passwords, exposed services, and insider threats are all realities that affect Linux systems just as they affect any other platform.

Consider the following scenarios that are common in real-world Linux environments:

A developer accidentally sets the permissions on a sensitive configuration file to `777`, making it world-readable and world-writable. Without an IDS monitoring file integrity, this misconfiguration could go unnoticed for weeks or months, giving any user on the system or any attacker who gains even limited access the ability to read or modify critical settings.

An attacker exploits a known vulnerability in an outdated version of OpenSSH running on a Linux server. They gain shell access and begin exfiltrating data. Without an IDS analyzing authentication logs and detecting anomalous session activity, the breach might not be discovered until the damage is done.

A disgruntled employee with legitimate access to a Linux database server begins copying customer records to an external storage device. Without behavioral analysis and log monitoring provided by an IDS, this insider threat could go completely undetected.

These are not hypothetical situations. They happen every day in organizations of all sizes. An intrusion detection system provides the visibility necessary to detect these events early, enabling administrators to respond quickly and minimize damage.

Types of Intrusion Detection Systems

Intrusion detection systems are broadly categorized into two primary types, each with its own strengths, limitations, and ideal use cases. Understanding both types is critical because tools like OSSEC and Wazuh primarily fall into one category while also incorporating elements of the other.

The following table provides a comprehensive comparison of the two primary types of intrusion detection systems:

Characteristic	**Network-Based IDS (NIDS)**	**Host-Based IDS (HIDS)**
Monitoring Scope	Monitors network traffic across a segment or entire network	Monitors activity on a single host or endpoint
Data Source	Network packets, traffic flows, protocol analysis	System logs, file integrity, process activity, system calls
Deployment Location	Positioned at network chokepoints such as routers, switches, or firewalls	Installed directly on the Linux host being monitored
Encrypted Traffic	Cannot inspect encrypted traffic without decryption capabilities	Can monitor activity regardless of encryption since it operates at the host level
Examples	Snort, Suricata, Zeek (formerly Bro)	OSSEC, Wazuh, AIDE, Tripwire
Strengths	Broad visibility across network, can detect scanning and network-level attacks	Deep visibility into host activity, file changes, rootkit detection

Limitations	Blind to encrypted traffic, cannot see host-level activity	Limited to the host where it is installed, requires agent deployment
Ideal Use Case	Monitoring perimeter traffic, detecting port scans, DDoS patterns	Monitoring critical Linux servers, detecting unauthorized file changes, log analysis
Resource Impact	Requires dedicated hardware or VM for traffic capture	Uses resources on the monitored host, though typically minimal
Linux Relevance	Often runs on Linux (Snort on Ubuntu, Suricata on CentOS)	Designed specifically for host-level Linux monitoring

A Network-Based Intrusion Detection System, or NIDS, monitors traffic flowing across a network. It captures and analyzes packets in real time, comparing them against a database of known attack signatures or analyzing them for anomalous patterns. A NIDS is typically deployed at strategic points in the network, such as behind a firewall or at the boundary between internal network segments. On Linux, tools like Snort and Suricata are popular NIDS solutions. They can be installed on a dedicated Linux machine that has its network interface configured in promiscuous mode, allowing it to capture all traffic on the network segment rather than just traffic destined for its own IP address.

To configure a network interface in promiscuous mode on a Linux system, you would use the following command:

```
sudo ip link set eth0 promisc on
```

You can verify the configuration with:

```
ip link show eth0
```

The output should include the word `PROMISC` in the flags, indicating that the interface is now capturing all network traffic on the segment.

A Host-Based Intrusion Detection System, or HIDS, operates at the individual host level. It is installed directly on the Linux machine it is tasked with protecting, and it monitors internal activities such as file system changes, log entries, running processes, user activity, and system call behavior. OSSEC and Wazuh are both host-based intrusion detection systems, and they are the primary focus of this book. A HIDS provides a depth of visibility that a NIDS simply cannot achieve. While a NIDS can tell you that suspicious traffic was directed at a particular server, a HIDS can tell you exactly what happened on that server as a result, whether files were modified, whether new user accounts were created, whether privilege escalation occurred, and much more.

In a well-architected Linux security environment, both NIDS and HIDS are deployed together to provide comprehensive coverage. The NIDS watches the roads leading to the castle, while the HIDS watches every room inside it.

Core Detection Methods

Regardless of whether an IDS is network-based or host-based, it employs one or more detection methods to identify malicious activity. The two primary detection methods are signature-based detection and anomaly-based detection.

Signature-Based Detection works by comparing observed activity against a database of known attack patterns, called signatures or rules. Each signature describes a specific attack or malicious behavior. When the IDS observes activity that matches a signature, it generates an alert. This method is highly effective at detecting known threats and produces relatively few false positives when the signature database is well maintained. However, it is completely blind to novel attacks that do not match any existing signature. On a Linux system running OSSEC, signatures are defined as rules in XML format and are stored in the rules directory, typically located at /var/ossec/rules/. A simple OSSEC rule that detects multiple failed SSH login attempts might look like this:

```
<rule id="5720" level="10" frequency="6" timeframe="120">
  <if_matched_sid>5716</if_matched_sid>
  <description>Multiple SSH authentication failures.</
description>
  <group>authentication_failures,</group>
</rule>
```

This rule triggers when rule 5716 (which detects a single SSH authentication failure) is matched six times within a 120-second window. The alert level is set to 10, indicating a high-severity event.

Anomaly-Based Detection takes a different approach. Instead of looking for known patterns, it establishes a baseline of normal behavior and then flags deviations from that baseline as potentially suspicious. For example, if a Linux web server typically handles 500 requests per minute and suddenly begins receiving 50,000 requests per minute, an anomaly-based system would flag this as abnormal. Similarly, if a user account that normally logs in during business hours from a specific IP range suddenly logs in at 3:00 AM from a foreign IP address, this deviation from the established baseline would trigger an alert. Anomaly-based detection can identify novel attacks that have never been seen before, but it is also prone to higher rates of false positives, especially during the initial learning period when the baseline is being established.

The following table summarizes the key differences between these detection methods:

Aspect	Signature-Based Detection	Anomaly-Based Detection
How It Works	Compares activity against known attack patterns	Compares activity against a learned baseline of normal behavior

Strengths	Accurate for known threats, low false positive rate	Can detect unknown and zero-day attacks
Weaknesses	Cannot detect unknown attacks, requires regular updates	Higher false positive rate, requires training period
Maintenance	Signature database must be regularly updated	Baseline must be periodically recalibrated
Linux Example	OSSEC rule matching failed SSH attempts in /var/log/secure	Wazuh detecting unusual process execution patterns
Best For	Environments with well-defined threat profiles	Environments concerned about advanced persistent threats

Modern IDS solutions like Wazuh combine both methods, using signature-based rules for known threats while also incorporating anomaly detection capabilities for identifying unusual system behavior that might indicate a previously unknown attack.

Key Components of a Linux IDS Architecture

A complete IDS deployment on Linux involves several interconnected components that work together to provide comprehensive monitoring and alerting. Understanding these components is essential for planning and implementing an effective intrusion detection strategy.

The Agent is the software component installed on each Linux host that needs to be monitored. In the context of OSSEC and Wazuh, the agent collects data from the host, including log files, file integrity information, running processes, and open ports. It then forwards this data to a central server for analysis. Installing a Wazuh agent on a Linux system involves adding the Wazuh repository and installing the agent package:

```
curl -s https://packages.wazuh.com/key/GPG-KEY-WAZUH | gpg --no-
default-keyring --keyring gnupg-ring:/usr/share/keyrings/
wazuh.gpg --import && chmod 644 /usr/share/keyrings/wazuh.gpg
echo "deb [signed-by=/usr/share/keyrings/wazuh.gpg] https://
packages.wazuh.com/4.x/apt/ stable main" | tee /etc/apt/
sources.list.d/wazuh.list
apt-get update
apt-get install wazuh-agent
```

The Manager or Server is the central component that receives data from all agents, processes it against the rule set, correlates events across multiple hosts, and generates alerts. In a typical Linux deployment, the Wazuh manager runs on a dedicated Linux server, often Ubuntu Server or CentOS/Rocky Linux, and is configured to listen for agent connections on port 1514.

The Rule Engine is the brain of the IDS. It contains the logic that determines what constitutes suspicious or malicious activity. Rules are typically defined in XML files and can be customized to match the specific needs of your environment. Both OSSEC and Wazuh ship with extensive default rule sets that cover common Linux security events, including SSH brute force attempts, file integrity violations, rootkit detection, and privilege escalation.

The Decoder is responsible for parsing raw log data into structured fields that the rule engine can evaluate. When a log entry arrives from a Linux system, the decoder extracts relevant information such as the source IP address, the username, the action performed, and the result. Without proper decoders, the rule engine would be unable to interpret the raw log data.

The Alert and Reporting System is the output mechanism that communicates detected threats to administrators. Alerts can be delivered via email, written to log files, sent to a SIEM system, or displayed in a web-based dashboard. Wazuh integrates with the Elastic Stack (now OpenSearch) to provide a rich, visual dashboard for exploring and analyzing security events across your entire Linux infrastructure.

Practical Exercise: Examining Linux Logs for Security Events

Before we begin working with OSSEC and Wazuh in the coming chapters, it is valuable to understand the raw data that these tools analyze. The following exercise walks you through examining key Linux log files for common security events.

First, examine the authentication log on a Debian or Ubuntu system:

```
sudo tail -50 /var/log/auth.log
```

On Red Hat, CentOS, Rocky Linux, or Fedora systems, the equivalent file is:

```
sudo tail -50 /var/log/secure
```

Look for entries that contain phrases like "Failed password," "Invalid user," or "session opened." These entries represent the raw data that an IDS like OSSEC or Wazuh would analyze.

Next, check for recent sudo activity:

```
sudo grep "sudo" /var/log/auth.log | tail -20
```

This command filters the authentication log for entries related to sudo usage, which is a critical area for monitoring privilege escalation.

To examine system messages for potential issues:

```
sudo tail -100 /var/log/syslog
```

Or on Red Hat-based systems:

```
sudo journalctl -n 100 --no-pager
```

Finally, check the current state of critical system files that an IDS would monitor for integrity:

```
ls -la /etc/passwd /etc/shadow /etc/sudoers /etc/ssh/sshd_config
```

Note the permissions, ownership, and modification times of these files. A host-based IDS continuously monitors these attributes and alerts you when any of them change unexpectedly.

```
Note: The exercises in this chapter are designed to build
familiarity with the types
of data that OSSEC and Wazuh analyze. In subsequent chapters, you
will learn how
to automate this monitoring and configure intelligent alerting
based on the events
found in these log files. Keep a record of the output from these
exercises, as it
will serve as a useful reference when we begin configuring IDS
rules.
```

Preparing for OSSEC and Wazuh

With this foundational understanding of intrusion detection systems firmly established, you are now prepared to move into the practical world of OSSEC and Wazuh. In the chapters that follow, we will install, configure, and tune these powerful host-based intrusion detection systems on Linux platforms. We will write custom rules, configure file integrity monitoring, detect rootkits, integrate with centralized logging and dashboarding solutions, and build a comprehensive security monitoring infrastructure that can scale from a single Linux server to an enterprise fleet of thousands of machines.

The key takeaway from this chapter is that intrusion detection is not optional for any Linux system that handles sensitive data, serves production traffic, or operates within a regulated environment. The question is not whether your Linux systems will be targeted, but when, and whether you will have the visibility to detect and respond to that targeting when it occurs. OSSEC and Wazuh provide that visibility, and this book will teach you how to deploy and master them.

Chapter 2: OSSEC and Wazuh Architecture

Understanding the internal architecture of intrusion detection systems is not merely an academic exercise. It is the foundation upon which every configuration decision, every deployment strategy, and every troubleshooting session rests. When you manage Linux servers in production environments, the difference between a well-understood security tool and a black box can mean the difference between catching an intrusion in real time and discovering a breach weeks after it has occurred. This chapter takes you deep into the architectural bones of both OSSEC and Wazuh, explaining how each component functions, how data flows through the system, and how these tools have been purpose-built to protect Linux infrastructure at scale.

The Foundation: Understanding Host-Based Intrusion Detection on Linux

Before dissecting the specific architectures of OSSEC and Wazuh, it is essential to understand what a host-based intrusion detection system (HIDS) actually does on a Linux machine. Unlike network-based intrusion detection systems that monitor traffic flowing across network segments, a HIDS operates directly on the host itself. It reads log files generated by the Linux kernel, system services, and applications. It monitors the integrity of critical files stored on Linux filesystems. It watches for changes in system configuration, examines running processes, and detects rootkits that attempt to hide their presence within the operating system.

On a Linux system, this means the HIDS interacts intimately with components such as `/var/log/syslog`, `/var/log/auth.log`, `/etc/passwd`, `/etc/shadow`, the `/proc` filesystem, and many other system resources. The architecture of the HIDS determines how efficiently and reliably it can monitor these resources without degrading system performance or missing critical events.

Both OSSEC and Wazuh follow a manager-agent architecture model, though they can also operate in standalone or agentless modes. The manager acts as the central brain, collecting data from agents deployed on individual Linux hosts, analyzing that data against rulesets, correlating events across multiple systems, and triggering alerts or active responses when threats are detected.

OSSEC Architecture in Detail

OSSEC was designed with modularity in mind. Each major function of the intrusion detection system is handled by a separate daemon or process, and these processes communicate with each other through internal Unix sockets and message queues on the Linux system. This modular design means that if one component encounters a problem, it does not necessarily bring down the entire system.

The Core Daemons and Their Roles

The OSSEC architecture comprises several key daemons, each responsible for a specific aspect of the intrusion detection pipeline. The following table provides a comprehensive overview of each daemon and its function within the system.

Daemon	Process Name	Function	Location
Manager Daemon	ossec-maild	Handles email alert notifications and delivers formatted alert messages to configured recipients	Manager only
Analysis Daemon	ossec-analysisd	The central analysis engine that decodes, matches rules, and correlates events from all sources	Manager only
Remote Daemon	ossec-remoted	Listens for incoming connections from agents, receives encrypted event data, and passes it to the analysis engine	Manager only
Syscheck Daemon	ossec-syscheckd	Performs file integrity monitoring by scanning configured directories and comparing file checksums against stored baselines	Manager and Agent
Log Collector	ossec-logcollector	Reads local log files and command outputs, then forwards them to the analysis engine or to the manager	Manager and Agent
Agent Daemon	ossec-agentd	Runs on monitored hosts, collects local data, and transmits it securely to the manager	Agent only

Execution Daemon	ossec-execd	Executes active response commands when triggered by the analysis engine, such as blocking an IP address with iptables	Manager and Agent
Monitor Daemon	ossec-monitord	Monitors the overall health of the OSSEC processes and compresses old log files	Manager only
Database Daemon	ossec-dbd	Writes alert data to a database backend such as MySQL or PostgreSQL for long-term storage and querying	Manager only (optional)

Data Flow Within OSSEC

Understanding how data moves through the OSSEC system is critical for both configuration and troubleshooting. The data flow begins at the point of collection and ends with an alert, a log entry, or an active response action.

On a monitored Linux agent, the `ossec-logcollector` daemon continuously reads from configured log files. For example, it might be configured to monitor `/var/log/auth.log` for authentication events, `/var/log/apache2/access.log` for web server activity, and `/var/log/syslog` for general system messages. The log collector reads new entries as they are written to these files, using inode tracking to handle log rotation gracefully.

Simultaneously, the `ossec-syscheckd` daemon performs periodic scans of configured directories. On a typical Linux server, this might include `/etc`, `/usr/bin`, `/usr/sbin`, and `/boot`. During each scan, syscheck calculates cryptographic hashes (MD5, SHA1, or SHA256) of each file and compares them against previously stored values. Any changes in file size, permissions, ownership, or content are flagged as events.

Both the log collector and syscheck send their collected data to the `ossec-agentd` daemon, which encrypts the data using a pre-shared key and transmits it over UDP port 1514 to the OSSEC manager. This encryption is critical because the event data may contain sensitive information about the Linux host, including usernames, IP addresses, and system configurations.

On the manager side, `ossec-remoted` receives the encrypted data, decrypts it, and places it into an internal message queue. The `ossec-analysisd` daemon then picks up each event from the queue and processes it through a multi-stage pipeline.

The analysis pipeline works as follows. First, the event is passed through a predecoding phase where common fields such as timestamp, hostname, and program name are extracted. Then the event enters the decoding phase, where OSSEC matches the log format against its library of decoders. These decoders use regular expressions to extract specific fields from the log entry. For example, a decoder for SSH logs might extract the source IP address, the username, and whether the authentication attempt succeeded or failed.

After decoding, the extracted fields are matched against the ruleset. OSSEC rules are organized in a hierarchical structure with levels ranging from 0 (ignored) to 16 (most severe). Rules can be simple pattern matches or complex composite rules that require multiple conditions to be met within a specified time window. This is where event correlation happens. For example, a single failed SSH login might generate a level 5 alert, but ten failed SSH logins from the same IP address within two minutes might trigger a level 10 alert indicating a brute force attack.

If a rule match results in an alert at or above the configured threshold, the alert is written to `/var/ossec/logs/alerts/alerts.log` and optionally sent via email through `ossec-maild`, written to a database through `ossec-dbd`, or used to trigger an active response through `ossec-execd`.

The active response system deserves special attention because it transforms OSSEC from a passive detection tool into an active defense mechanism. When `os-sec-execd` receives an active response command, it executes a predefined script on the Linux system. Common active response actions include adding firewall rules using `iptables` or `nftables` to block an attacking IP address, disabling a user account by modifying `/etc/shadow`, or running a custom script that performs any action the administrator has defined.

```
# Example of how OSSEC active response blocks an IP using
iptables
# This is executed automatically by ossec-execd when triggered
iptables -I INPUT -s 192.168.1.100 -j DROP

# The block is typically temporary, with a timeout configured in
ossec.conf
# After the timeout, the reverse command is executed
iptables -D INPUT -s 192.168.1.100 -j DROP
```

OSSEC Directory Structure on Linux

The installation directory of OSSEC on a Linux system follows a well-organized structure that reflects its modular architecture.

Directory Path	**Contents**	**Purpose**
/var/ossec/bin	Executable binaries	Contains all OSSEC daemon binaries and management utilities
/var/ossec/etc	Configuration files	Houses ossec.conf, the main configuration file, and shared agent configuration

/var/ossec/rules	Rule files	Contains XML rule definitions used by the analysis engine
/var/ossec/decoders	Decoder files	Stores decoder definitions that parse log formats
/var/ossec/logs	Log files and alerts	Stores OSSEC's own logs, alert files, and archived events
/var/ossec/queue	Internal queues	Used for inter-process communication between OSSEC daemons
/var/ossec/stats	Statistical data	Contains hourly and weekly event statistics
/var/ossec/tmp	Temporary files	Used during operations such as agent key exchange
/var/ossec/agentless	Agentless scripts	Scripts for monitoring systems where agents cannot be installed
/var/ossec/active-response	Response scripts	Contains scripts executed during active response actions

Wazuh Architecture: The Evolution

Wazuh began as a fork of OSSEC and has since evolved into a comprehensive security platform that retains the core HIDS functionality while adding significant new capabilities. Understanding the Wazuh architecture requires recognizing both what it inherited from OSSEC and what it has fundamentally changed or added.

Wazuh Component Overview

Wazuh's architecture is divided into three primary components that work together to form a complete security monitoring solution. These components are the Wazuh agent, the Wazuh manager (also called the Wazuh server), and the Wazuh indexer along with the Wazuh dashboard.

The Wazuh agent runs on each monitored Linux host and performs the same fundamental collection tasks as the OSSEC agent, but with significant enhancements. The agent collects log data, performs file integrity monitoring, detects rootkits, monitors system inventory, assesses security configuration compliance, and scans for known vulnerabilities. Each of these functions is handled by a dedicated module within the agent.

Wazuh Agent Module	**Function**	**Linux-Specific Details**
Log Data Collection	Reads and forwards log files and journal entries	Supports both traditional syslog files and systemd journal via journalctl integration
File Integrity Monitoring (FIM)	Monitors files and directories for changes in real time or on schedule	Uses Linux inotify subsystem for real-time monitoring of filesystem events
Rootcheck	Scans for rootkits and system anomalies	Checks against known Linux rootkit signatures, examines /dev, hidden processes, and kernel modules
Syscollector	Gathers system inventory information	Collects installed packages (dpkg, rpm), running processes, network interfaces, ports, and hardware information
SCA (Security Configuration Assessment)	Evaluates system configuration against security benchmarks	Supports CIS benchmarks for various Linux distributions including Ubuntu, CentOS, Red Hat, and Debian