THOMAS MORE

Central Logging met SIEM / SOAR

Realisatiedocument

Alexander Conserva
Student Bachelor in de Elektronica-ICT – Cloud & Cyber Security

Stagebedrijf: Intersoft Electronics

2025 – 2026

Inhoudsopgave

1. INLEIDING 6

2. ANALYSE 7

2.1. Selectiecriteria en marktonderzoek 7

2.2. Log-extractie en datatransformatie 7

2.3. Evaluatie van SIEM- en logmanagementplatformen 7

2.4. Threat Intelligence en SOAR-alternatieven 8

3. REALISATIE 9

3.1. Architectuur en Log-ingestie 9

3.2. Threat Intelligence Integratie (MISP) 10

3.3. Ontwikkeling van de maatwerkapplicatie 10

3.4. Logretentie en Compliance-archivering via Graylog 16

3.5. Systeemhardening van de IT-infrastructuur 17

4. BESLUIT 18

4.1 Evaluatie van de Proof of Concept en projectdoelstellingen 18

4.2 Realisatie van leerdoelen 18

4.3 Advies en aanbevelingen voor de productieomgeving 18

LITERATUURLIJST 20

BIJLAGEN 21

Lijst van tabellen

Tabel 1: Vergelijking SIEM Platformen

Lijst van figuren

Figuur 1. Centrale gebruikersinterface van het dashboard. 11
Figuur 2. Overzicht van de geautomatiseerde case management. 12
Figuur 3. Detailweergave van alert-correlatie binnen een incidentdossier. 12
Figuur 4. Integratie van de lokaal gehoste AI-assistent. 13
Figuur 5. Interface van de archieffunctie. 14
Figuur 6. Dashboard voor gebruikersbeheer. 14
Figuur 7. Systeeminstellingen en accountconfiguratie. 15
Figuur 8. Beveiligde initiële configuratiewizard. 15

Begrippenlijst

AI (Artificial Intelligence): Kunstmatige intelligentie; in dit project toegepast om complexe loggegevens en waarschuwingen automatisch te analyseren en interpreteren ter ondersteuning van beveiligingsanalisten.

JSON (JavaScript Object Notation): Een gestandaardiseerd, lichtgewicht formaat voor de uitwisseling van data. In dit project gebruikt om ongestructureerde loggegevens leesbaar te maken voor het SIEM-systeem.

Legacy Systeem: Verouderde hard- of software (zoals Windows Server 2008 of Windows XP) die geen beveiligingsupdates meer ontvangt (End-of-Life), maar nog wel essentieel is voor de bedrijfsvoering.

MISP (Malware Information Sharing Platform): Een opensource platform voor Threat Intelligence, gebruikt voor het verzamelen, opslaan en delen van indicatoren van compromittatie (zoals kwaadaardige IP-adressen).

PoC (Proof of Concept): Een praktische, afgebakende proefopstelling om aan te tonen dat een bedacht concept (zoals de gekozen SOC-stack) in de realiteit daadwerkelijk functioneert.

SBOM (Software Bill of Materials): Een gedetailleerde inventarislijst van alle softwarecomponenten en afhankelijkheden binnen een applicatie, gebruikt om kwetsbaarheden in de 'supply chain' te identificeren.

SIEM (Security Information and Event Management): Een centraal systeem dat loggegevens uit het hele netwerk verzamelt, normaliseert, correleert en analyseert om beveiligingsincidenten te detecteren en hier waarschuwingen (alerts) voor te genereren.

SIRP (Security Incident Response Platform): Een applicatie die fungeert als 'case management' of ticketsysteem, speciaal gebouwd voor beveiligingsanalisten om incidenten gestructureerd te documenteren en af te handelen.

SOAR (Security Orchestration, Automation and Response): Een systeem dat beveiligingsprocessen en workflows automatiseert. Het koppelt verschillende tools aan elkaar en kan bijvoorbeeld automatisch data verrijken of incidenten aanmaken.

SOC (Security Operations Center): Het team, de processen en de technologieën binnen een organisatie die verantwoordelijk zijn voor het continu monitoren, detecteren en afhandelen van cyberdreigingen.

TI (Threat Intelligence): Dreigingsinformatie; verzamelde data over (potentiële) cyberaanvallen en kwaadaardige actoren die gebruikt wordt om proactief te kunnen verdedigen.

VM (Virtual Machine): Een virtuele computer/server die als software draait op een fysieke server. Binnen dit project gebruikt om de verschillende applicaties veilig op te hosten.

1. Inleiding

Dit realisatiedocument is opgesteld in het kader van de afstudeerstage binnen de opleiding Bachelor in de Elektronica-ICT – Cloud & Cyber Security en behandelt het project "Central Logging met SIEM / SOAR". Het initiële projectplan (Conserva, 2026) stelde als hoofddoel om de compatibiliteit van diverse SIEM- en SOAR-oplossingen te onderzoeken in relatie tot verouderde, out-of-support IT-omgevingen.

De centrale probleemstelling van dit onderzoek is: hoe kunnen loggegevens veilig en efficiënt uit legacy-systemen worden geëxtraheerd, zonder de stabiliteit van deze fragiele systemen in gevaar te brengen? Het wegnemen van deze netwerkblinde vlekken is van cruciaal belang voor de organisatie, niet alleen om potentiële aanvalsvectoren te minimaliseren, maar ook om te kunnen voldoen aan actuele compliance-richtlijnen. Hiervoor is onder andere gekeken naar methoden zoals agentless logging, legacy-compatible agents en rsyslog.

Waar het projectplan de theoretische doelstellingen en risico's in kaart bracht, beschrijft dit document de concrete uitwerking en de succesvolle opbouw van de Proof of Concept (PoC). In de volgende hoofdstukken worden de verschillende stappen van deze realisatie in detail besproken:

2. Analyse: Hierin wordt het vooronderzoek behandeld. Er wordt dieper ingegaan op de onderzochte log-extractiemethoden en de vergeleken SIEM/SOAR-platformen, inclusief een objectieve verantwoording van de gemaakte technologische keuzes ter voorbereiding van de PoC.

3. Realisatie: Dit hoofdstuk bevat de beschrijving van het uiteindelijke resultaat. De opzet van de werkende testomgeving, de veilige configuratie van de logverzameling voor de geselecteerde legacy systemen, de inrichting van het SIEM-dashboard en de ontwikkelde detectieregels en waarschuwingen (alerts) komen hier aan bod.

4. Besluit: Het document wordt afgerond met een evaluatie van de PoC en een terugkoppeling naar de vereisten uit het projectplan (Conserva, 2026). Tot slot wordt een gedetailleerd advies en een concrete aanbeveling geformuleerd voor de definitieve inrichting van de 'SOC stack' voor productief gebruik. Voor lezers die behoefte hebben aan verdere technische verdieping en de exacte configuratiedetails van de gerealiseerde systemen, is er uitgebreide documentatie opgenomen in de bijlagen.

2. Analyse

Dit hoofdstuk biedt een overzicht van het vooronderzoek naar de verschillende componenten die noodzakelijk zijn voor de opzet van een centrale logverzameling en monitoringomgeving ter ondersteuning van een Security Operations Center (SOC). Ter voorbereiding van de Proof of Concept (PoC) zijn diverse tools en technologieën geanalyseerd en met elkaar vergeleken. De focus lag hierbij op kostenefficiëntie, effectiviteit binnen verouderde IT-omgevingen en compatibiliteit met moderne beveiligingsstandaarden.

2.1. Selectiecriteria en marktonderzoek

Bij de selectie van een geschikte SIEM- (Security Information and Event Management) en SOAR-oplossing (Security Orchestration, Automation, and Response) lag de nadruk op opensourcesoftware en kosteloze licentiemodellen. Om een realistisch beeld te vormen van de huidige industriestandaarden, is er tijdens een jobbeurs actief navraag gedaan bij diverse IT-bedrijven naar de beveiligingssystemen die zij in de praktijk inzetten. De inzichten uit deze gesprekken vormden de basis voor de verdere selectie van de te onderzoeken platformen.

2.2. Log-extractie en datatransformatie

Een kernuitdaging binnen legacy-omgevingen is het veilig en gestructureerd verzamelen van logs. Voor de routering en transformatie van deze data is in eerste instantie de applicatie NXlog geëvalueerd. Deze tool bleek echter ongeschikt voor de testomgeving: de opensource community-versie bood geen ondersteuning voor het gekozen besturingssysteem (Ubuntu 24.04) en de installatie vereiste minimaal 19 GB aan werkgeheugen, wat te zwaar is voor de beoogde infrastructuur (NXlog, z.d.).

Als efficiënt alternatief is Vector geselecteerd. Vector functioneert als een lichtgewicht log-router die in staat is om complexe, ongestructureerde logs uit verouderde systemen te ontleden (parsen) en om te zetten naar een gestandaardiseerd JSON-formaat (JavaScript Object Notation). Hierdoor kan de onderliggende SIEM-oplossing deze data correct interpreteren en verwerken (Vector, z.d.).

2.3. Evaluatie van SIEM- en logmanagementplatformen

Tijdens de analysefase zijn verschillende logmanagementplatformen geëvalueerd, waaronder de Elastic Stack (Elastic, z.d.) en Wazuh (Wazuh, z.d.). Om de besluitvorming transparant te maken, zijn de belangrijkste oplossingen vergeleken op basis van vooraf vastgestelde criteria.

Tabel 1: Vergelijking SIEM Platformen

Criterium	Elastic Stack (ELK)	Wazuh
Licentiekosten	Deels open-source, deels betaald (premium features)	Volledig open-source en kosteloos
Dreigingsdetectie	Vereist aanzienlijke handmatige configuratie	Out-of-the-box regelspecifieke detecties
Legacy Compatibiliteit	Zware agent, minder geschikt voor oude systemen	Brede ondersteuning via agent en agentless (syslog)
Resourceverbruik	Hoog	Gemiddeld tot Laag
Conclusie	Geschikt voor pure log-aggregatie	Geselecteerd als primaire SIEM-oplossing

Uit deze vergelijking bleek Wazuh de meest geschikte primaire SIEM-oplossing voor het hoofdproject, mede dankzij de uitgebreide ingebouwde functionaliteiten voor dreigingsdetectie op fragiele legacy-systemen (Wazuh, z.d.)

Alternatief traject: Pure logretentie via Graylog

Naast de primaire SIEM-oplossing is er een afzonderlijk traject onderzocht voor systemen waarbij actieve dreigingsdetectie (Wazuh) niet strikt noodzakelijk is. Voor systemen waarop het wel toegestaan en mogelijk is om (betaalde) agent-software van het bedrijf te installeren, volstaat soms puur de langdurige opslag van loggegevens om aan compliance-richtlijnen te voldoen. Voor deze specifieke, afgescheiden use-case is Graylog geëvalueerd als een onafhankelijk logmanagementsysteem (Graylog, z.d.). Omdat de focus van dit afstudeerproject echter specifiek ligt op het beveiligen en monitoren van legacy-systemen zonder deze moderne afhankelijkheden, vormt de Wazuh-stack de kern van de gerealiseerde Proof of Concept.

2.4. Threat Intelligence en SOAR-alternatieven

Om de detectiecapaciteiten van de SIEM-oplossing te verrijken, is de integratie van Threat Intelligence (TI) geanalyseerd. Hiervoor is gekozen voor een MISP-server (Malware Information Sharing Platform). Wazuh gebruikt deze MISP-server om binnenkomende netwerkstromen te verifiëren tegen een continu bijgewerkte database van bekende kwaadaardige IP-adressen en domeinnamen (MISP, z.d.).

Tot slot is er gekeken naar oplossingen voor SOAR en incidentafhandeling (case management). Bestaande tools zoals Shuffle en TheHive zijn onderzocht, maar het standaard dashboard van Wazuh en de integratie met deze externe SOAR-tools boden onvoldoende overzicht voor de specifieke vereisten van de legacy-omgeving (Shuffle, z.d.; TheHive, z.d.). Dit inzicht leidde tot de strategische beslissing om een eigen maatwerkapplicatie te ontwikkelen voor efficiëntere incidentrespons.

3. Realisatie

Dit hoofdstuk beschrijft de concrete uitwerking van de geselecteerde technologieën tot een werkende Proof of Concept (PoC). De nadruk ligt op de architectuur van de SOC-stack (Security Operations Center), de data-integratie, de ontwikkeling van een op maat gemaakte beheerapplicatie met AI-ondersteuning en de implementatie van beveiligingsmaatregelen op het besturingssysteem.

3.1. Architectuur en Log-ingestie

Binnen de testomgeving is de Wazuh-infrastructuur ingezet als het centrale component voor dreigingsdetectie en logverzameling (zie Bijlage A voor de uitgebreide Wazuh-documentatie en configuratiedetails). Omdat het IT-landschap van de organisatie bestaat uit een complexe mix van systemen waaronder uiterst fragiele legacy-omgevingen is er een gelaagde aanpak toegepast voor log-ingestie. Afhankelijk van de technische beperkingen en risicoprofielen van de individuele eindpunten, zijn de volgende methoden succesvol geïmplementeerd:

Wazuh Agents: Voor systemen waarbij het besturingssysteem dit toeliet en de stabiliteit niet in het geding kwam, is de native Wazuh-agent ingezet. Om het beheer te vereenvoudigen, zijn deze agents ingedeeld in specifieke groepen (agent groups), waardoor configuraties centraal en uniform gepusht kunnen worden. Om te voorkomen dat ongeautoriseerde systemen zich als agent registreren, is de registratieprocedure extra beveiligd door middel van wachtwoordauthenticatie.

Directe Syslog-integratie: Netwerkcomponenten en systemen waarop geen agent geïnstalleerd kon worden (zoals de Kali-testomgeving), zijn geconfigureerd om hun loggegevens rechtstreeks via het Syslog-protocol naar de Wazuh-manager te sturen. Tijdens de realisatie is onderzocht of deze datastroom geëncrypteerd kon worden verstuurd. Aangezien Wazuh native geen geëncrypteerde syslog ondersteunt, is er gewerkt met de standaard onversleutelde syslog-overdracht binnen het afgeschermde netwerk.

Agentless Monitoring: Voor sterk verouderde systemen waarbij zowel het installeren van software als het configureren van actieve log-forwarding onmogelijk of een te groot veiligheidsrisico was, is gebruikgemaakt van Wazuh's agentless monitoring functionaliteit. Hierbij logt de Wazuh-server periodiek via SSH in op het doelsysteem om integriteitscontroles (bestandsveranderingen) uit te voeren.

Vector voor uitzonderingsgevallen: Oudere netwerkapparatuur, zoals legacy switches, genereren vaak verouderde logformaten die standaard niet door de Wazuh-manager worden begrepen. Voor deze specifieke uitzonderingen is Vector ingericht als log-router. Vector ontvangt de ongestructureerde syslogs van de switch en gebruikt specifieke transformatiescripts om deze data om te zetten naar een gestandaardiseerd JSON-formaat, waarna het door Wazuh wel correct verwerkt kan worden.

Datatransformatie en Normalisatie
Aangezien er logs uit sterk uiteenlopende bronnen in de centrale SIEM binnenkwamen, wees de praktijktest uit dat niet alle data standaard werd geclassificeerd. Om deze blinde vlekken te verhelpen, zijn er handmatig specifieke decoders en rules (detectieregels) geschreven in Wazuh. Hierdoor wordt nu elke binnenkomende, voorheen onbekende log correct genormaliseerd en kan er adequaat een waarschuwing (alert) aan gekoppeld worden.

Realtime Alarmering via Discord-integratie
Om de reactietijd van het incidentrespons-team te optimaliseren, is er een externe integratie gerealiseerd met Discord. Via deze koppeling stuurt de Wazuh kritieke beveiligingswaarschuwingen (alerts met een hoge prioriteit) automatisch door naar een afgeschermd communicatiekanaal. Hierdoor ontvangen analisten direct een pushmelding bij ernstige incidenten. Dit garandeert dat urgente dreigingen onmiddellijk worden opgemerkt en opgevolgd, zelfs wanneer het primaire dashboard op dat moment niet actief gemonitord wordt.

3.2. Threat Intelligence Integratie (MISP)

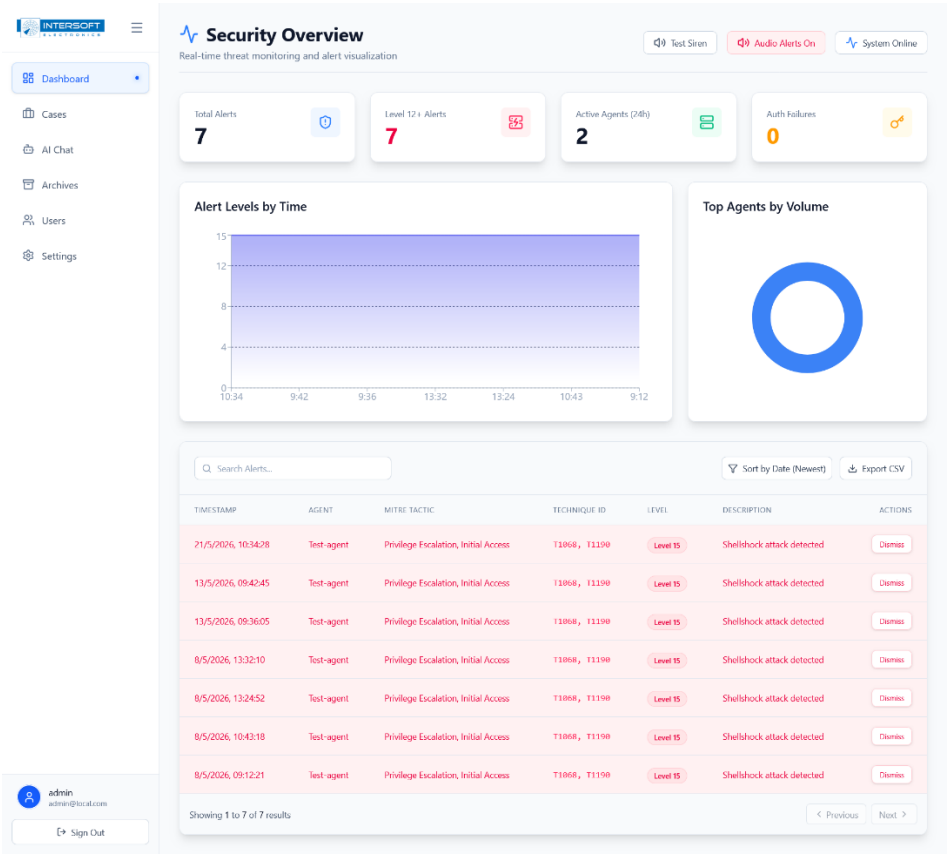
Om het aantal 'false positives' te verminderen en accuraat te reageren op daadwerkelijke dreigingen, is een integratie opgezet met een lokaal gehoste MISP-server (Malware Information Sharing Platform) (zie Bijlage B voor de uitgebreide MISP-documentatie en configuratiedetails). Binnen MISP zijn diverse threat feeds en 'warninglists' geactiveerd die dagelijks via een geautomatiseerd proces worden gesynchroniseerd met actuele dreigingsinformatie.

Via een op maat gemaakt integratiescript controleert Wazuh automatisch of IP-adressen en bestands-hashes uit de binnenkomende logs overeenkomen met de indicatoren uit de MISP-database. Wanneer er een kwaadaardig IP-adres wordt gedetecteerd, triggert het systeem een 'active response' waarbij het kwaadaardige IP-adres direct wordt geblokkeerd. Voor authenticatie met de MISP-API is gebruikgemaakt van een 'least privileged' serviceaccount, in combinatie met veilig geconfigureerde certificaten.

3.3. Ontwikkeling van de maatwerkapplicatie

Tijdens de configuratiefase bleek dat de standaard dashboards van Wazuh, evenals de ingebouwde mogelijkheden om custom dashboards te bouwen, te veel beperkingen oplegden qua ontwerpvrijheid en functionaliteit. Om deze reden is er besloten om een volledig eigen webapplicatie te ontwikkelen die fungeert als het centrale zenuwstelsel voor incidentrespons.

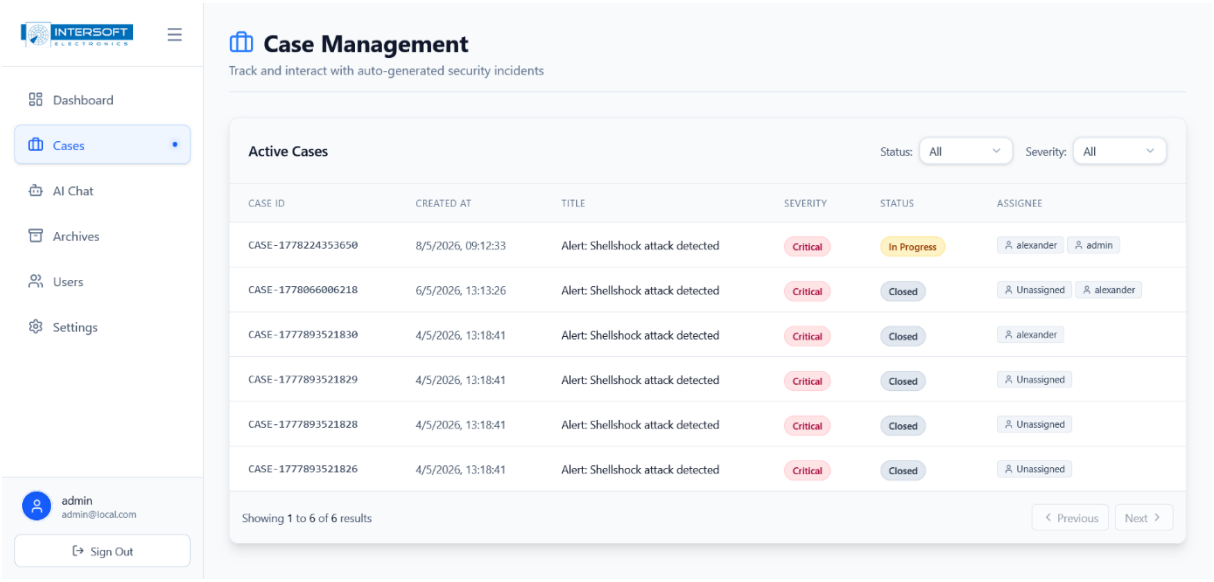
Dashboard en Auditieve Alarmering
De applicatie is voorzien van een overzichtelijke en moderne gebruikersinterface (met ondersteuning voor light en dark mode), ontworpen om de werkdruk van analisten te verlichten (zie Figuur 1). Een cruciale toevoeging ten opzichte van het standaard Wazuh-systeem is de implementatie van auditieve alarmen. Wanneer er een kritieke waarschuwing (Wazuh alert level 12 of hoger) binnenkomt, genereert het dashboard direct een duidelijk hoorbaar geluidssignaal, wat de reactietijd bij ernstige incidenten aanzienlijk verkort.



Figuur 1. Centrale gebruikersinterface van het dashboard.

Geautomatiseerd Case Management
Het hart van de applicatie is de geïntegreerde case management. Het afhandelen van incidenten is vergaand geautomatiseerd door middel van de volgende functionaliteiten:

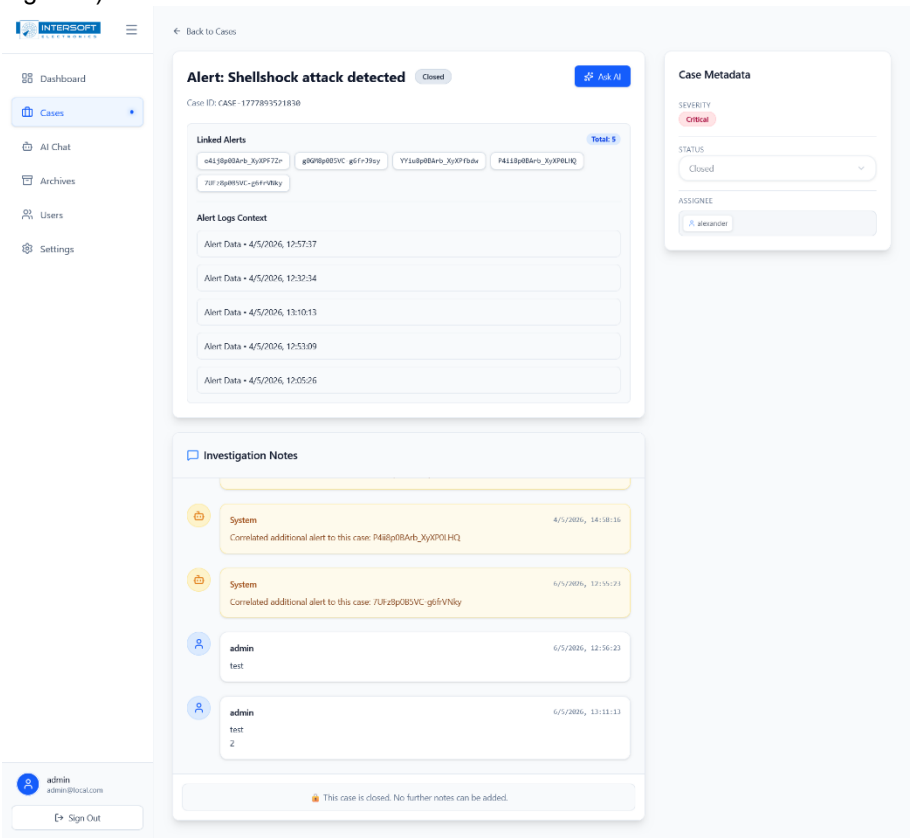
Automatische Incidentcreatie
Bij het detecteren van een level 12+ alert wordt er automatisch een nieuwe case (incidentdossier) aangemaakt in het systeem, zoals geïllustreerd in Figuur 2.



Figuur 2. Overzicht van de geautomatiseerde case management.

Alert Correlatie

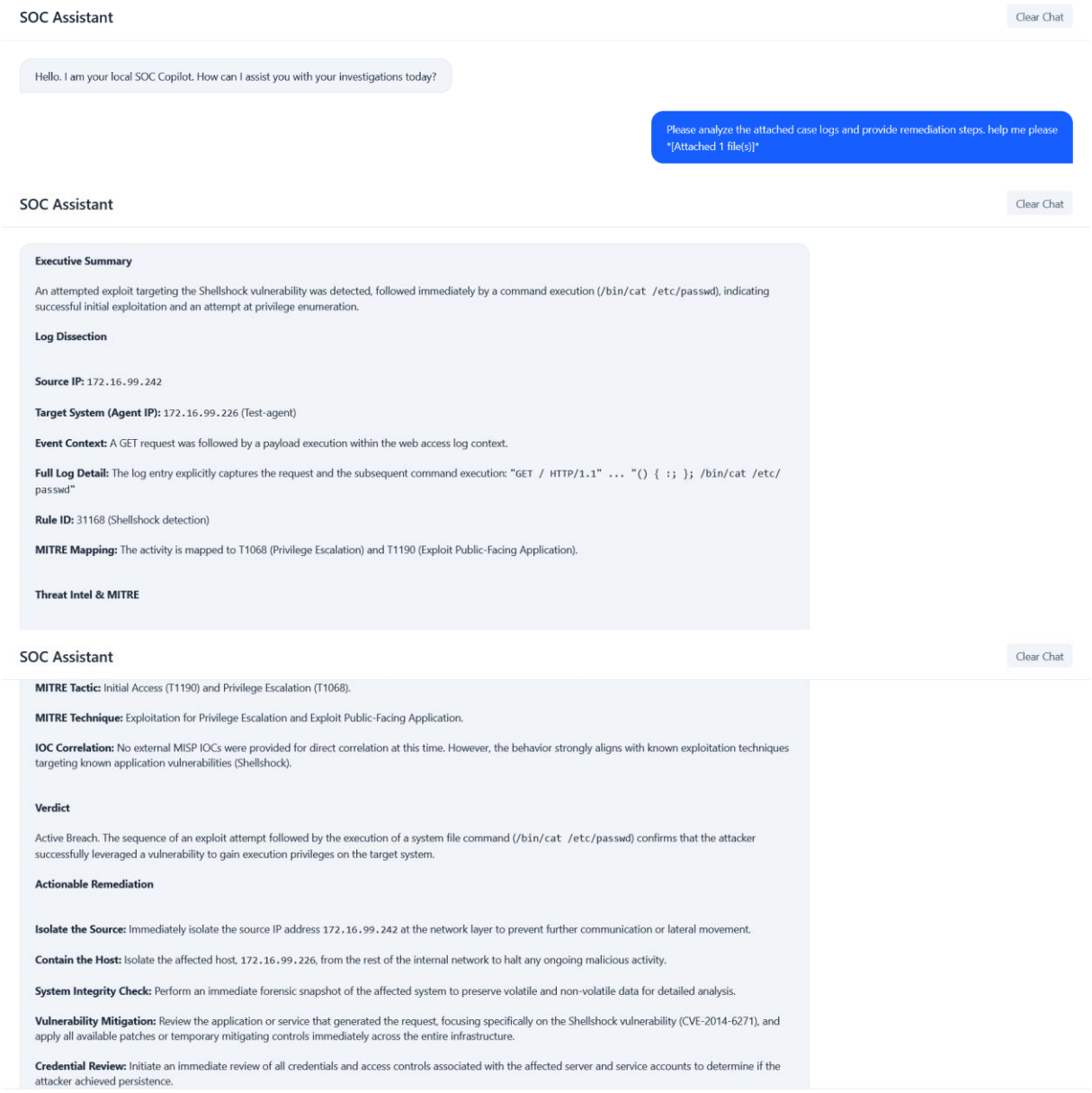
Om alert-moeheid ('alert fatigue') te voorkomen, controleert de applicatie of er al een openstaande case is voor de desbetreffende agent of het specifieke endpoint. Nieuwe, gerelateerde waarschuwingen worden automatisch aan dit bestaande dossier toegevoegd, in plaats van telkens nieuwe cases te genereren (zie Figuur 3).



Figuur 3. Detailweergave van alert-correlatie binnen een incidentdossier.

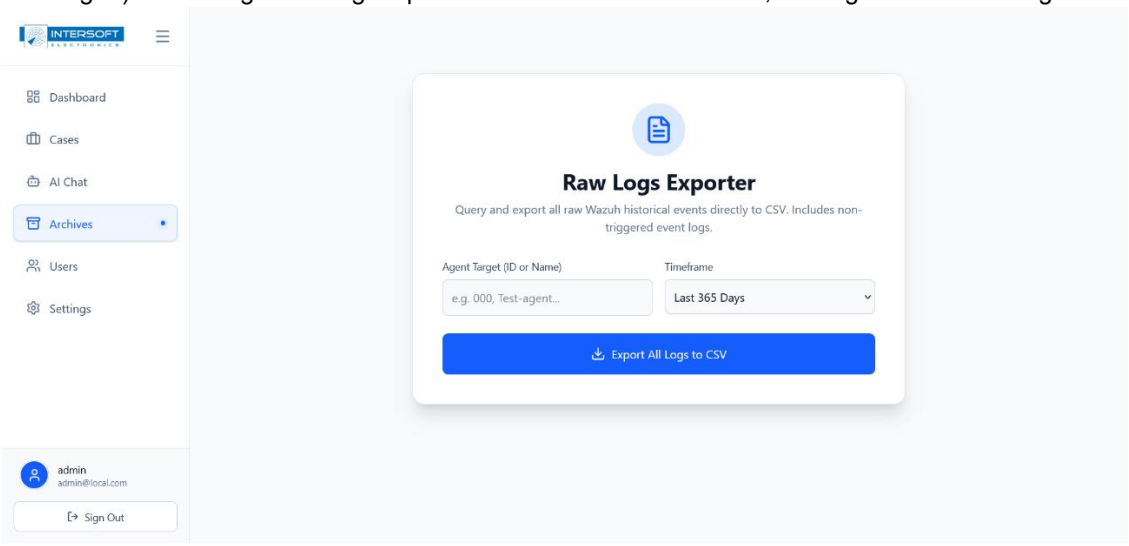
Dossierbeheer
Binnen een case hebben analisten direct toegang tot de bijbehorende ruwe loggegevens. Incidenten kunnen worden toegewezen aan specifieke gebruikers, en de status van het onderzoek wordt bijgehouden. Tevens kunnen analisten opmerkingen toevoegen om de bevindingen gestructureerd te documenteren.

AI-ondersteuning
Als innovatieve toevoeging is Artificial Intelligence (AI) diep in de incident-workflow geïntegreerd. Binnen een case kan de analist gebruikmaken van de "Ask AI"-functionaliteit. Het systeem verzamelt hierbij automatisch alle relevante loggegevens van het incident en stuurt deze mee als context naar de lokaal gehoste AI-assistent. De AI helpt vervolgens bij het interpreteren van de complexe logdata, wat de analyse aanzienlijk versnelt en helpt tot een oplossing (zie Figuur 4).



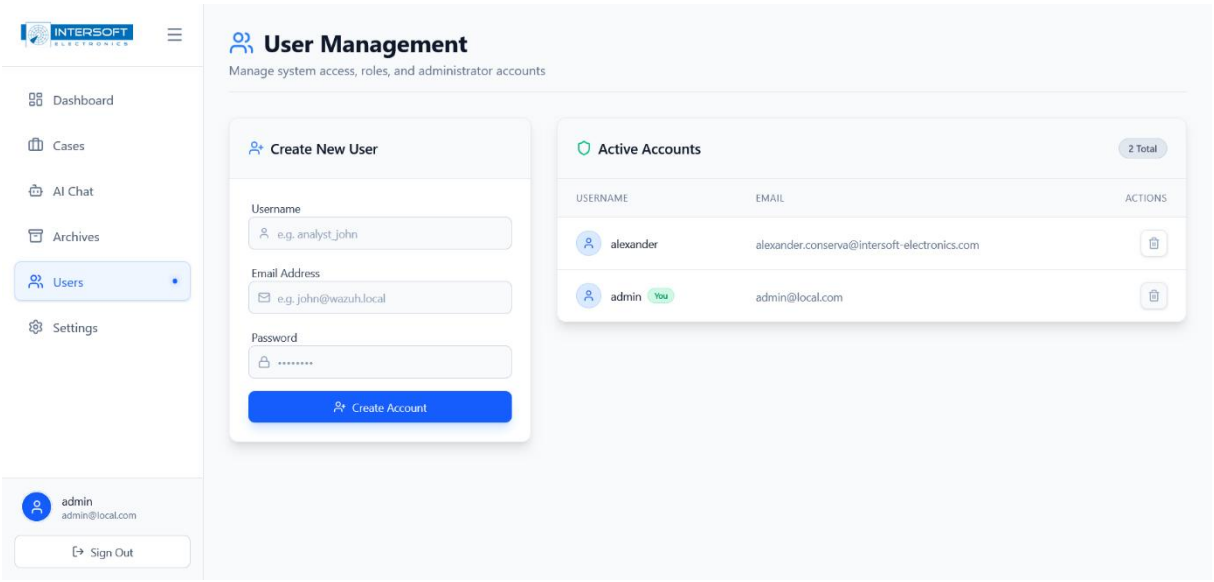
Figuur 4. Integratie van de lokaal gehoste AI-assistent.

Archivering en Systeembeheer
Om te voldoen aan administratieve en compliance-eisen, bevat de applicatie een uitgebreide archieffunctie. Hiermee kunnen alle opgeslagen loggegevens van geselecteerde agents over specifieke tijdsperiodes (tot 365 dagen) eenvoudig worden geëxporteerd naar een CSV-bestand, zoals geïllustreerd in Figuur 5.

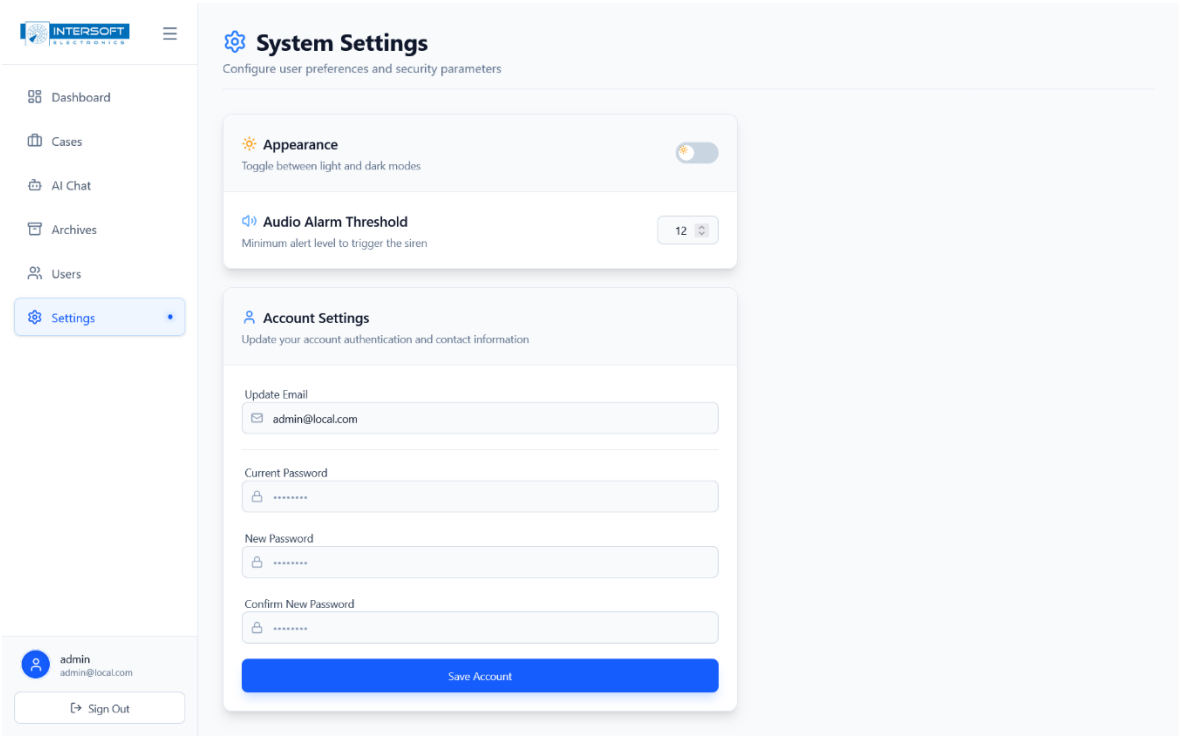


Figuur 5. Interface van de archieffunctie.

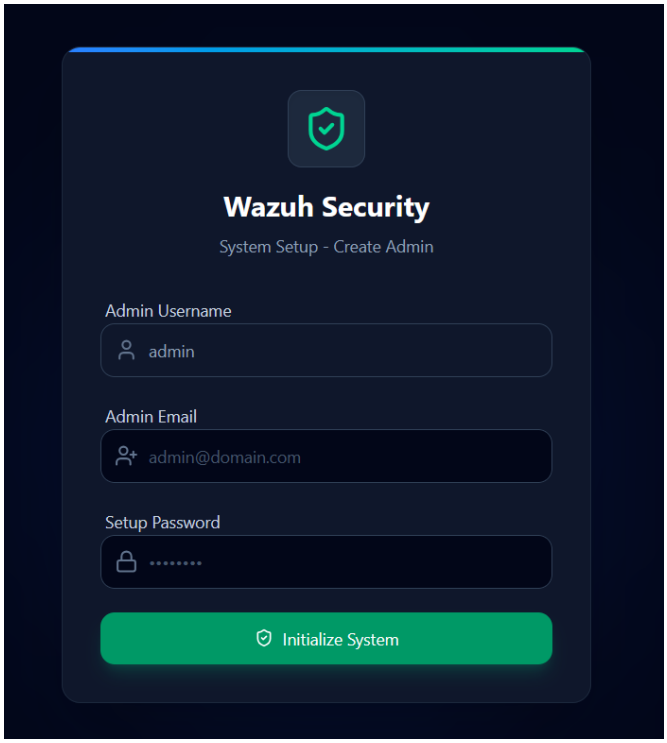
.
Daarnaast is er een robuuste beheermodule ingebouwd (User management en Settings) (zie Figuur 6 en Figuur 7). Om een veilige ingebruikname te garanderen, is er een logica geïmplementeerd voor de initiële installatie: wanneer de applicatie op een nieuwe server wordt opgestart en detecteert dat er nog geen beheerder bestaat, wordt de gebruiker verplicht om eerst een veilig admin-account aan te maken, zoals geïllustreerd in Figuur 8.



Figuur 6. Dashboard voor gebruikersbeheer.



Figuur 7. Systeeminstellingen en accountconfiguratie.



Figuur 8. Beveiligde initiële configuratiewizard.

Applicatiebeveiliging en Systeemarchitectuur
Aangezien de maatwerkapplicatie uiterst gevoelige incidentgegevens verwerkt, is er een gelaagde 'defense-in-depth' beveiligingsstrategie toegepast over de gehele architectuur. Hierbij zijn de volgende maatregelen geïmplementeerd:

Netwerk- en Infrastructuurbeveiliging
De applicatie is opgebouwd volgens een Zero-Trust microservices-architectuur. Met behulp van Docker Compose (Docker, z.d.) zijn alle componenten (de React frontend, Node.js backend, PostgreSQL database en de lokale Ollama AI) ondergebracht in afzonderlijke, geïsoleerde containers. De applicatie wordt uitsluitend gehost binnen het afgeschermde, lokale bedrijfsnetwerk. Door het gebruik van gescheiden Docker-netwerken is de interne routering strikt afgebakend; uitsluitend de Nginx-webserver fungeert als het interne toegangspunt voor gebruikers. Hierdoor zijn de onderliggende database en API's volledig afgeschermd en kunnen zij niet rechtstreeks worden benaderd door andere systemen of gebruikers op het lokale netwerk. Verder is alle communicatie voorzien van TLS/SSL-encryptie om veilige HTTPS-verbindingen te garanderen en man-in-the-middle-aanvallen uit te sluiten.

API-beveiliging en Hardening
Om de API te beschermen tegen brute-force inlogpogingen en Denial of Service (DoS) aanvallen, is rate limiting geconfigureerd. Daarnaast worden er automatisch HTTP-security headers geïnjecteerd om de applicatie te weren tegen kwetsbaarheden zoals Cross-Site Scripting (XSS) en clickjacking. Een strikte CORS-configuratie (Cross-Origin Resource Sharing) garandeert tevens dat de backend uitsluitend verzoeken accepteert die afkomstig zijn van het legitieme frontend-domein.

Identiteits- en Toegangsbeheer (IAM)
Gebruikerswachtwoorden worden veilig opgeslagen door middel van cryptografische hashing en salting via bcrypt. Sessiebeheer verloopt stateless en fraudebestendig via JSON Web Tokens (JWT). Om autorisatie te handhaven, is een strikt Role-Based Access Control (RBAC) model uitgewerkt. Aan de backendzijde weigert een specifieke toegangscontrole (middleware) elk ongeautoriseerd API-verzoek, waardoor kwaadwillige manipulatie via tools als Postman of cURL onmogelijk is. Aan de frontendzijde voorkomen Route Guards dat standaardgebruikers via de URL toegang forceren tot beheerderssecties.

Software Supply Chain Security
Om de integriteit van de codebase te kunnen auditen is een Software Bill of Materials (SBOM) gemaakt voor zowel de backend als de frontend. Op basis van deze documentatie is de infrastructuur ontdaan van overbodige afhankelijkheden wat de totale attack surface van de server aanzienlijk heeft gereduceerd.

Integratie volgens het Principle of Least Privilege
De koppeling tussen de applicatie en de externe Wazuh-infrastructuur is opgezet met minimale rechten. In plaats van een beheerder-account te gebruiken, raadpleegt de applicatie Wazuh via een specifiek aangemaakt serviceaccount met uitsluitend leesrechten. Om de complexiteit van de frequent verlopende Wazuh API-tokens te omzeilen, is de backend bovendien geconfigureerd om specifieke data rechtstreeks via de afgeschermde OpenSearch-database te bevragen.

3.4. Logretentie en Compliance-archivering via Graylog

Naast de primaire SIEM-infrastructuur voor actieve monitoring, is er binnen de Proof of Concept een afzonderlijk traject uitgewerkt voor moderne en stabiele IT-omgevingen. Voor systemen waarop het probleemloos mogelijk en toegestaan is om de actieve, betaalde beveiligingssoftware (agents) van het bedrijf te installeren, is de inzet van Wazuh voor actieve dreigingsdetectie overbodig. Echter vereisen actuele compliance-richtlijnen dat de loggegevens van deze systemen veilig en langdurig (tot 365 dagen) worden bewaard. Voor deze specifieke, use-case is Graylog geïmplementeerd als een onafhankelijk en gecentraliseerd logmanagementsysteem (zie Bijlage C voor de uitgebreide Graylog-documentatie en configuratiedetails).

Configuratie van Inputs en Datastromen
Om de loggegevens vanuit de bedrijfssoftware efficiënt te verzamelen, zijn er binnen Graylog specifieke Inputs geconfigureerd die de data veilig in ontvangst nemen. Zodra de ruwe logs het platform binnenkomen, worden deze door middel van Streams en vooraf ingestelde routeringsregels gecategoriseerd. Hierdoor worden de loggegevens overzichtelijk van elkaar gescheiden op basis van hun bron, wat cruciaal is om in de toekomst snel en accuraat forensische zoekopdrachten of compliance-audits te kunnen uitvoeren.

Index Sets en Geautomatiseerd Retentiebeleid
Gezien de grote hoeveelheid logdata die over een periode van een jaar wordt gegenereerd, is efficiënt opslagbeheer noodzakelijk. Dit is gerealiseerd door gebruik te maken van Index Sets binnen Graylog. Hierin is een strikt retentiebeleid (retention strategy) van 365 dagen geconfigureerd, exact afgestemd op de bedrijfsvereisten. Zodra loggegevens deze maximale bewaartermijn overschrijden, roteert of verwijdert Graylog de oudste indexen automatisch. Dit garandeert dat de organisatie doorlopend compliant blijft, zonder dat de opslagcapaciteit van de server overbelast raakt of er handmatig beheer nodig is.

3.5. Systeemhardening van de IT-infrastructuur

Een fundamentele vereiste voor een veilige monitoringomgeving is de robuustheid van de onderliggende servers. Tijdens de realisatiefase zijn verschillende besturingssystemen (waaronder Ubuntu en openSUSE) aan uitgebreide beveiligingsaudits onderworpen met behulp van de tool Lynis (Cisofy, z.d.).

De auditresultaten (Lynis-scores) zijn gedocumenteerd en vergeleken om de meest veilige basisconfiguratie te bepalen. Op basis van deze testresultaten is de keuze gevallen op Ubuntu 24.04 LTS (Ubuntu, z.d.). Om de installatie van toekomstige servers te standaardiseren en te beveiligen, is er een geautomatiseerd hardeningscript ontwikkeld. Dit script voert direct de vereiste beveiligingsaanpassingen door op de server, waardoor kwetsbaarheden geminimaliseerd worden.

4. Besluit

Dit afsluitende hoofdstuk biedt een recapitulatie van het uitgevoerde onderzoek en de resulterende Proof of Concept (PoC). Er wordt geëvalueerd in hoeverre de initiële doelstellingen uit het projectplan zijn behaald en er wordt een concreet advies geformuleerd voor de toekomstige inrichting van de productieomgeving.

4.1 Evaluatie van de Proof of Concept en projectdoelstellingen

Het primaire doel van dit project was het onderzoeken en opzetten van een compatibele logmanagement- en monitoringoplossing voor verouderde, out-of-support IT-omgevingen, zonder de stabiliteit van deze fragiele systemen aan te tasten. Uit de resultaten van de testomgeving blijkt dat deze doelstelling succesvol is gerealiseerd.

Door een gelaagde architectuur toe te passen, bestaande uit native Wazuh-agents, agentless monitoring, directe syslog en Vector als gespecialiseerde log-router voor edge cases is het mogelijk gebleken om logs uit uiteenlopende legacy-systemen veilig te extraheren en te standaardiseren. Binnen de Wazuh SIEM zijn vervolgens specifieke detectieregels geconfigureerd die direct inspelen op de bekende kwetsbaarheden van deze systemen. De actieve integratie met een MISP-server voor Threat Intelligence heeft de effectiviteit van het platform verder verhoogd door incidenten te verrijken en bekende kwaadaardige actoren automatisch te blokkeren op netwerkniveau.

Ter vervanging van externe SOAR-oplossingen is een succesvolle, lokaal gehoste webapplicatie ontwikkeld. Deze applicatie centraliseert het case management en verlaagt de reactietijd aanzienlijk dankzij geautomatiseerde alert-correlatie, auditieve waarschuwingen en de integratie van een AI-assistent voor loganalyse.

4.2 Realisatie van leerdoelen

Naast de technische oplevering zijn ook de vooropgestelde leerdoelen uit het projectplan behaald. Tijdens het project is diepgaande praktijkervaring opgedaan met logmanagement, infrastructuurbeveiliging en SIEM in complexe architecturen. De academische uitdaging om abstracte beveiligingsrisico's in verouderde omgevingen te vertalen naar functionele monitoring-eisen, is succesvol in de praktijk gebracht. Dit vertaalde zich onder meer in de applicatieontwikkeling met strikte 'security by design' principes (zoals Zero-Trust en SBOM-generatie) en een grondige, geautomatiseerde verharding (hardening) van de onderliggende besturingssystemen.

4.3 Advies en aanbevelingen voor de productieomgeving

Op basis van de getrokken conclusies uit de PoC wordt een definitief advies geformuleerd voor de implementatie van de SOC-stack in de productieomgeving:

- **Primaire SIEM en dreigingsdetectie:** Behoud Wazuh als het centrale platform voor realtime beveiligingsmonitoring. Maak maximaal gebruik van de native extractiemethoden (zoals agents en agentless monitoring) voor stabiele systemen, en zet Vector uitsluitend in voor de routering en parsing van ongestructureerde loggegevens uit netwerkapparatuur (zoals legacy switches).
- **Threat Intelligence:** Behoud de MISP-server als actieve component binnen de architectuur. Zorg ervoor dat de geautomatiseerde 'active response' functionaliteit (zoals het blokkeren van IP-adressen) met zorg wordt gefaseerd ingeschakeld in de productieomgeving om verstoring van kritieke bedrijfsprocessen te voorkomen.
- **Incidentrespons en Case Management:** Het wordt geadviseerd om de ontwikkelde maatwerkapplicatie als primair dashboard in te zetten voor het beheer van beveiligingswaarschuwingen. De geïntegreerde AI-functionaliteit, samen met de CSV-export mogelijkheden voor compliancedoeleinden, biedt analisten een aanzienlijke meerwaarde.

- **Systeemhardening:** De structurele toepassing van het ontwikkelde hardeningscript op alle nieuwe besturingssystemen is een kritieke voorwaarde om de algehele veiligheid van de SOC-infrastructuur te waarborgen.
- **Logretentie en Alternatieve Trajecten:** Maak gebruik van de geconfigureerde 365-dagen logretentie binnen Wazuh. Voor systemen die strikt buiten de actieve SIEM-monitoring vallen en uitsluitend logopslag vereisen voor compliance, kan de eerder geëvalueerde Graylog-infrastructuur overwogen worden als gescheiden archiefsysteem.

LITERATUURLIJST

Wazuh. (z.d.). *Wazuh documentation*. Wazuh. https://documentation.wazuh.com/current/index.html

Elastic Docs | Elastic Docs. (z.d.). Elastic. https://www.elastic.co/docs

Inc, D. (z.d.). *DockerDocs*. Docker Documentation. https://docs.docker.com/

Graylog. (z.d.). *GrayLog | SIEM, Log Management & API Protection*. https://graylog.org/

Misp. (z.d.). *MISP Open Source Threat Intelligence Platform*. MISP. https://www.misp-project.org/

NXLog. (z.d.). *NXLog - Achieve complete security observability with powerful insights from your log data.* https://nxlog.co/

Vector. (z.d.). A Lightweight, Ultra-fast Tool For Building Observability Pipelines. https://vector.dev/

Lynis - Security auditing and hardening tool for Linux/Unix. (z.d.). Cisofy. https://cisofy.com/lynis/

Ubuntu Pro | Ubuntu. (z.d.). Ubuntu. https://ubuntu.com/pro

Shuffle. (z.d.). *Shuffle*. https://shuffler.io/

StrangeBee. (z.d.). *TheHive | Collaborative Security Case Management Platform*. https://strangebee.com/thehive/

Conserva, A. (2026). Projectplan: Central Logging met SIEM / SOAR. Intersoft Electronics. (intern document).

Bijlagen

Opmerking: Door de documentconversie naar Word is de originele lay-out van de externe documentatie mogelijk gewijzigd. Dit heeft geen invloed op de accuraatheid van de weergegeven data.

Bijlage A: Wazuh Documentation

WAZUH (COMPANY ASSESSMENT) – WHY THIS IS A SUSTAINABLE OPEN-SOURCE CHOICE 25

1)Executive Summary 25

2)Why Evaluate the “Company Behind Open Source”? 25

3)Company Snapshot (Who Is Wazuh?) 25

4)Organizational Structure & Leadership (Not a One-Man Operation) 25

5)Evidence of Staffing Scalability: Teams & Hiring 26

6)Open-Source Governance & Community Ecosystem 26

7)Evidence of Engineering Maturity: GitHub Activity & Contributors 27

8)Support & Services (Continuity Through Commercial Backing) 27

9)Indicators of Adoption & Market Confidence 27

10) Conclusion: Why Wazuh Is a Good Choice in Terms of Continuity 27

11) References (Links) 28

WAZUH DISTRIBUTED DEPLOYMENT (LOCAL VM LAB) — UBUNTU SERVER 28

1.Goal & Architecture 28

2.Prerequisites 28

2.1 Operating system 28

2.2 VM sizing (lab-friendly) 28

2.3 Networking 29

2.4 Required ports 29

3.Deployment Steps 29

Step 1 — Prepare all VMs 29

Step 3 — Copy `wazuh-install-files.tar` to the other VMs 31

Step 4 — Install the Wazuh Indexer on `node-1` 31

Step 5 — Install the Wazuh Server (Manager) on `wazuh-1` 32

Step 6 — Install the Wazuh Dashboard on `dashboard` 32

4.Credentials & Password Retrieval 32

5.Validate End-to-End Functionality 33

5.1 Can the dashboard see indexed alerts? 33

5.2 Why you may already see alerts without adding agents 33

WAZUH INSTALLATION ON UBUNTU 24.04 LTS (SINGLE-NODE DEPLOYMENT) 33

Purpose 33

Scope 33

Recommended sizing 33

1.Installation command 34

Important note about the initial password 34

2.Where the password is stored 34

Retrieve the password file 34
3. First login 34
4. Disable the Wazuh repository after installation 35
Disable the Wazuh APT repository 35
5. Enable the Wazuh repository again for a planned upgrade 35
6. Verify whether the repository is disabled 35
Expected result when disabled 35

SUPPORTED MONITORING METHODS 35
1. Agent-based monitoring 35
2. Syslog monitoring 36
3. Agentless monitoring over SSH 36
4. API-based cloud and service monitoring 36

HOW TO DEPLOY A WAZUH AGENT? 36
Overview 36
Purpose 36
Recommended Deployment Method: Dashboard-Generated Install Command 37
Step 1 – Open the Wazuh Dashboard 37
Step 2 – Start the Agent Deployment Wizard 37
Step 3 – Select the Endpoint Configuration 37
Step 4 – Copy the Generated Command 37
Step 5 – Run the Command on the Target Endpoint 37
What the Dashboard Command Does 37
Verify the Agent Installation 38
Verify in the Dashboard 38

WAZUH AGENT PERMISSIONS ON LINUX 38
Summary 38
Observed process permissions 38
What this means 38
What Wazuh can do 39
What Wazuh cannot do by default 39

AGENT GROUPS – MANAGED CONFIGURATION APPROACH 40
Purpose 40
Why we use agent groups 40
Configuration approach 40

WAZUH AGENTLESS MONITORING 41
What it is? 41
When to use it 41
Step 1 — Install expect 41
Step 2 — Create an SSH key on the Wazuh manager 41
Step 3 — Copy the public key to the remote machine 41
Step 4 — Register the host in Wazuh 42
Step 5 — Add the agentless block to ossec.conf 42

What the fields mean 43
Step 6 — Restart Wazuh 43
Step 7 — Check that it works 43
WAZUH SYSLOG FORWARDING 44
What it is 44
When to use it 44
Important difference from agentless monitoring 44
Step 1 — Configure Wazuh to receive syslog 44
Example: Syslog over TCP 44
Step 2 — Restart the Wazuh manager 45
Step 3 — Configure the endpoint sender with rsyslog 45
Example config to put into rsyslog.conf or create an aditional configuration file. 45
Step 4 — Restart rsyslog on the sender 45
Step 5 — Allow the firewall 46
Step 6 — Test the forwarding 46
WAZUH AGENT ENROLLMENT PASSWORD CONFIGURATION 46
Purpose 46
Scope 46
How it works 46
Configuration on the Wazuh Manager 47
1. Enable password authentication in `ossec.conf` 47
2. Create the enrollment password file 47
3. Set the correct permissions and ownership 47
4. Restart the Wazuh manager 47
WAZUH LOG RETENTION – 365 DAYS 47
Purpose 47
1. Background: What retention means in Wazuh 47
2. Configure the Wazuh indexer (alerts only) 48
2.1 Create the retention policy in the Dashboard 48
2.2 Apply the policy to existing indices 50
2.3 Verification 50
3. Configure the Wazuh manager (local file retention) 50
3.1 Edit the manager retention file 50
3.2 Restart the manager 50
4. Final result for alerts-only retention 51
5. Optional: If you want to enable logall / archives too 51
5.1 Enable all-events archive output on the manager 51
5.2 Enable archives in Filebeat 51
5.3 Add 365-day retention for archives in the indexer 52
5.4 Manager retention remains the same 54
HOW TO CREATE A NEW CUSTOM RULE? 54
1. Purpose 54
2. Why this example is useful 54

3. Background 55
4. Example rule objective 55
5. Where to add the rule in the dashboard 55
6. Custom rule XML 55
Explanation 56
7. Apply the rule 56
DISCORD INTEGRATION FOR WAZUH 57
Overview 57
1. CREATE PYTHON INTEGRATION SCRIPT 57
2. CONFIGURE `OSSEC.CONF` 58
Add integration block 58
3. RESTART WAZUH 58
4. VERIFICATION 58

Wazuh (Company Assessment) – Why This Is a Sustainable Open-Source Choice

1) Executive Summary

Wazuh is not a "one-man open-source project," but an organization with **more than 200 team members worldwide**, supported by a clear corporate structure (CEO/COO/CTO/CFO, as well as directors responsible for areas such as Support, Cloud, Agent, Indexer, QA/Dashboard, Training, CTI, and Community).

In addition, there is clearly **active project development** on GitHub (tens of thousands of commits, thousands of forks/stars, and hundreds of contributors), as well as a formal ecosystem around both **community support** (Slack, mailing list, ambassadors) and **commercial services** (professional support with SLAs).

2) Why Evaluate the "Company Behind Open Source"?

When assessing open-source tooling (such as **SIEM/XDR**), technical quality is important, but so are:

- **continuity** (will the project keep moving forward?),
- **maintainability** (patches, releases),
- **supportability** (community and/or contractual support),
- and **bus factor** (not being dependent on a single maintainer).

Wazuh explicitly positions itself as an **open-source platform with scale and enterprise adoption**, and publicly communicates about its **team size, download figures, and user base**—which helps answer the continuity question in an objective way.

3) Company Snapshot (Who Is Wazuh?)

- **Founded:** Wazuh states that the company has existed since **2015**.
- **Team size:** "more than 200 professionals distributed all around the world" / "Over 200 team members power Wazuh each day."
- **Adoption / scale indicators:** Wazuh mentions, among other things, **30+ million downloads per year**, **15+ million protected endpoints**, and **100+ thousand enterprise users**.

Interpretation from a continuity perspective: these are signals of a mature product organization and broad adoption, which reduces the likelihood that the project could "suddenly stop" due to a low bus factor.

4) Organizational Structure & Leadership (Not a One-Man Operation)

The **Our Team** page shows a clear management structure with multiple executive roles and domain owners, including:

- **FOUNDER & CEO**
  - **COO**
  - **CTO**
  - **CFO**
  - and directors responsible for **Community, Cloud, HR, Marketing, Support, Presales, CTI, Training, TAM, Server/QA/Dashboard, and Agent**

Wazuh also mentions an **Advisory Board** on the same page, which points to governance and

at least some advisory structures beyond a single decision-maker or developer.

Why this matters: a **multi-disciplinary team** (engineering + QA + support + training + customer success) reduces dependency on individual key persons and increases the predictability of **releases, support, and roadmap execution**.

5) Evidence of Staffing Scalability: Teams & Hiring

The **Careers** page shows that Wazuh is actively recruiting and distinguishes multiple teams, such as:

- **AGENT TEAM**
  - **Cloud / Cloud Apps**
  - **Content**
  - **CTI**
- **DASHBOARD**
  - **DevOps**
  - **Finance**
  - **Indexer**
- **MARKETING**
  - **Operations**
  - **QA**
  - **Sales**
  - **Server**
- **TRAINING**

(with multiple remote roles available).

An organization that continuously publishes roles in both **engineering** (e.g. **C++ / Indexer / Cloud**) and **operations / support / training** typically demonstrates the structural capacity required to **maintain and further develop** the platform.

6) Open-Source Governance & Community Ecosystem

Wazuh has an explicit **community page** with multiple channels:

- **Slack** (including **#community**)
- **GitHub** (PRs, issues, contributions)
- **MAILING LIST**
  - **Ambassadors**, who share knowledge and support users

In addition, the documentation repository explicitly states that **"Members of the Wazuh team and community users contribute to its development and daily improvement"**, and that users can participate and discuss through the mailing list.

Why this matters: an active community with multiple communication channels reduces **vendor lock-in**, improves **troubleshooting speed**, and increases the **bus factor**, because knowledge and contributions are not held exclusively inside the company.

7) Evidence of Engineering Maturity: GitHub Activity & Contributors

The public GitHub repository **wazuh/wazuh** shows:

- approximately **14.8k stars**
- approximately **2.2k forks**
- approximately **45k commits**
- and **very recent development activity** (recent merges/commits)

It also has a **large contributor base** ("**Contributors 303+ …**") and a clear release history.

The documentation also has its own repository with many commits and contributors, which is typical of projects that maintain **product-quality documentation and governance**.

Conclusion here: the combination of **high repository activity, many contributors, and a separate documentation repository** supports the statement that Wazuh is a **mature open-source project** that does not rely on a single maintainer.

8) Support & Services (Continuity Through Commercial Backing)

Wazuh offers **Professional Support** with explicit support plans (**Standard / Premium**), including:

- **RESPONSE SLAS**
- **health checks**
- and **follow-the-sun support** across multiple time zones

In addition, Wazuh also offers **consulting services** (architecture, deployment, tuning, custom rules / dashboards / integrations), which indicates that there is a commercial services layer around the platform, something that is important for **enterprise continuity**.

Interpretation for your risk analysis: the platform remains open source, but there is also a **professional support path** available if the business-critical nature of the environment (**SIEM / SOC**) requires it.

9) Indicators of Adoption & Market Confidence

Wazuh claims broad adoption (including **Fortune 500 companies** and **tech giants**) and communicates metrics such as **downloads per year** and **ent erprise users**.

Beyond that, the **Our Customers** / homepage messaging shows that Wazuh positions itself as **enterprise-ready** and **widely deployed**, which often correlates with:

- a larger community,
- more bug reports and fixes,
- and a stronger product lifecycle.

10) Conclusion: Why Wazuh Is a Good Choice in Terms of Continuity

Based on publicly available information, Wazuh as an organization is clearly **much larger than a one-person project** (200+ team members, multi-disciplinary leadership, growth/hiring), with strong indicators of maturity (**active GitHub development, many contributors, extensive documentation**) and a mature ecosystem of **community plus professional support**.

This makes it reasonable to conclude that Wazuh, as an **open-source SIEM/XDR solution**, provides a **sustainable foundation for a SOC platform**
—especially if we also ensure that **basic internal knowledge and documentation are maintained**, and **optionally consider a support contract**
for production-critical environments.

11) References (Links)

- Wazuh – About us
- Wazuh – Our team
- Wazuh – Careers
- Wazuh – Community
- Wazuh – Professional support
- GitHub – wazuh/wazuh
- GitHub – wazuh/wazuh-documentation
- Wazuh – Mailing list (Google Groups)

Wazuh Distributed Deployment (Local VM Lab) — Ubuntu Server

24.04 LTS

1. Goal & Architecture

This tutorial deploys Wazuh in a **distributed (single-node per component)** architecture using **separate VMs** for each central component: **Indexer**, **S erver (Manager)**, and **Dashboard**.

Wazuh architecture consists of three central components: **Wazuh server**, **Wazuh indexer**, and **Wazuh dashboard**.

LAB TOPOLOGY (RECOMMENDED MINIMUM):

- **VM1 — Indexer:** `node-1`
- **VM2 — Server/Manager:** `wazuh-1`
- **VM3 — Dashboard:** `dashboard`

> *Note: Wazuh also supports all-in-one and multi-node clustering. This lab focuses on "separate nodes" first, because it's the easiest stepping stone to real production deployments.*

2. Prerequisites

Operating system

Use **Ubuntu Server 24.04 LTS** on each VM (recommended for server workloads and commonly used for Wazuh central components).

Ubuntu Server baseline requirements (OS only) are modest (minimum RAM and disk), but Wazuh components require more.

VM sizing (lab-friendly)

Use the Wazuh installation guide hardware recommendations per component:

INDEXER (NODE-1):

- Minimum: **4 GB RAM / 2 cores**
- Recommended: **16 GB**

RAM / 8 cores Wazuh Server

(wazuh-1):

- Minimum: **2 GB RAM / 2 cores**
- Recommended: **4 GB**

RAM / 8 cores Dashboard

(dashboard):

- Minimum: **4 GB RAM / 2 cores**
- Recommended: **8 GB RAM / 4 cores**

> *Tip: If your laptop/host is limited, prioritize RAM on the **Indexer** VM because indexing/search tends to be the most memory-intensive part of the stack.*

Networking

Assign each VM a **static IP** (or a DHCP reservation), because the certificates and node definitions in `config.yml` must match real addresses. Example (replace with your lab subnet):

- `node-1 192.168.56.11`
- `wazuh-1 192.168.56.12`
- `dashboard 192.168.56.13`

Required ports

For agent enrollment (later), Wazuh documents these manager ports:

- **1514/TCP** agent communication
- **1515/TCP** enrollment
- **55000/TCP** Wazuh server API (used by some enrollment methods)

Wazuh architecture also states agents connect to the Wazuh server service on **1514/TCP** by default.

For Dashboard access, the Wazuh dashboard runs on **443** by default (you can change it).

For Indexer verification, the installation assistant uses HTTPS on **9200** for checks (curl examples).

3. Deployment Steps

Step 1 — Prepare all VMs

On each VM:

1. Install Ubuntu Server 24.04 LTS and ensure it has network connectivity.
2. Set hostname to match the intended node name (recommended for clarity; not strictly required, but helps troubleshooting).

Example:

sudo hostnamectl set-hostname node-1

> *The `name:` field in `config.yml` is a **node identifier** used for certificates and install commands; it is not your Linux username.*

Step 2 — Initial configuration (generate certs + passwords)

Run this **once** on a single VM (recommended: **Indexer VM `node-1`**). The assistant will generate a `wazuh-install-files.tar` archive containing certificates, cluster key, and passwords.

On **node-1**:

```
curl -sO https://packages.wazuh.com/4.14/wazuh-install.sh curl -sO https://packages.wazuh.com/4.14/config.yml
```

Edit `config.yml` and set your node names and IPs (example):

```
nodes:
  indexer:
    - name: node-1
      ip: "192.168.56.11"

  server:
    - name: wazuh-1
      ip: "192.168.56.12"
      node_type: master

  dashboard:
    - n
      a
      m
      e:
      d
      a
      s
      h
      b
      o
      a
      r
      d
      ip
      :
      "1
      9
      2.
      1
      6
      8.
      5
      6.
      1
      3"
```

This format and the need to define all nodes is described in the assisted installation docs.

GENERATE CONFIG FILES:

sudo bash wazuh-install.sh --generate-config-files

This produces `wazuh-install-files.tar` which you must copy to all nodes.

Step 3 — Copy `wazuh-install-files.tar` to the other VMs

Copy the generated tar file from `node-1` to `wazuh-1` and `dashboard`. The Wazuh docs explicitly state you should copy this file to all servers in the distributed deployment.

> *If you see "permission denied" errors while copying, ensure the file is readable by the user.*

Step 4 — Install the Wazuh Indexer on `node-1`

The assisted method installs the indexer using `--wazuh-indexer <node-name>`,

where the node name must match `config.yml`. On **node-1**, ensure `wazuh-install-files.tar` is in your current directory and run:

curl -sO
https://packages.wazuh.com/4.
14/wazuh-install.sh sudo bash
wazuh-install.sh --wazuh-
indexer node-1

Initialize the indexer cluster (run once)

The assistant requires a

cluster initialization step:

sudo bash wazuh-

install.sh --start-cluste

This is run **only once** (even in a multi-node indexer cluster).

Verify indexer is working

The docs provide curl-based verification:

curl -k -u admin https://192.168.56.11:9200
curl -k -u admin https://192.168.56.11:9200/_cat/nodes?v

Step 5 — Install the Wazuh Server (Manager) on `wazuh-1`

The assisted method installs the server using `--wazuh-server <node-name>`,

and the node name must match `config.yml`. On **wazuh-1** (ensure `wazuh-install-files.tar` is present in your working directory):

curl -sO
https://packages.wazuh.com/4.
14/wazuh-install.sh sudo bash
wazuh-install.sh --wazuh-
server wazuh-1

Step 6 — Install the Wazuh Dashboard on `dashboard`

The assisted method installs the dashboard using `--wazuh-dashboard <node-name>`. On **dashboard**

(ensure `wazuh-install-files.tar` is present):

curl -sO
https://packages.wazuh.com/4.
14/wazuh-install.sh sudo bash
wazuh-install.sh --wazuh-
dashboard dashboard

The dashboard uses HTTPS and is typically reachable on port **443** by default.

4. Credentials & Password Retrieval

The installation assistant stores credentials in `wazuh-passwords.txt` inside `wazuh-install-files.tar`, and the docs show how to print them. Run (on any node where the tar exists):

sudo tar -O -xvf wazuh-install-files.tar wazuh-install-files/wazuh-passwords.txt

> *Treat this file as sensitive: it contains admin credentials and certificate materials used to secure communications.*

5. Validate End-to-End Functionality

Can the dashboard see indexed alerts?

Wazuh dashboard troubleshooting suggests checking whether Wazuh indices exist in the

indexer (this is also a good “pipeline works” test). From the **dashboard VM**:

curl -k -u admin https://192.168.56.11:9200/_cat/indices/wazuh-alerts-*?

If indices exist, it indicates alerts are being stored and indexed.

Why you may already see alerts without adding agents

Wazuh server monitors itself by default using an internal embedded agent with ID **000**, so you can see alerts even before enrolling external endpoints.

To confirm, filter in the dashboard “Security events” by:

- `agent.id:"000"`

Wazuh Installation on Ubuntu 24.04 LTS (Single-Node Deployment)

4.1. Purpose

This page documents how to install **Wazuh on Ubuntu** as a **single-node / all-in-one deployment** using the official **Wazuh installation assistant**. In this deployment model, the **Wazuh server, Wazuh indexer, and Wazuh dashboard** are installed on the **same Ubuntu host**.

4.2. Scope

This procedure is intended for **small environments** where a single-node deployment is sufficient. This deployment is usually enough for **up to 100 endpoints** and **90 days of queryable/indexed alert data**, depending on storage and event volume.

Recommended sizing

Wazuh provides the following quickstart sizing examples for **90 days of queryable/indexed alert data**:

- **1–25 agents**: 4 vCPU, 8 GiB RAM, 50 GB storage.
- **25–50 agents**: 8 vCPU, 8 GiB RAM, 100 GB storage.
- **50–100 agents**: 8 vCPU, 8 GiB RAM, 200 GB storage.

1. Installation command

Run the following command on the Ubuntu server:

curl -sO https://packages.wazuh.com/4.14/wazuh-install.sh && sudo bash ./wazuh-install.sh -a

This is the official **Quickstart** command from Wazuh and installs the **Wazuh server, Wazuh indexer, and Wazuh dashboard** on the same host.

Important note about the initial password

When the installation script finishes, Wazuh displays a summary that includes:

- The **Wazuh dashboard URL**.
- The default **admin** username.
- The generated **administrator password**.

> ***Important: Save the password that is shown after the install script finishes.*** *This is the easiest moment to capture the initial admin password and use it for the first login.*

2. Where the password is stored

If the password was not saved from the installation output, the generated passwords can be found in **`wazuh-passwords.txt`** inside **`wazuh-install-files.tar`**.

Retrieve the password file

Run:

sudo tar -O -xvf wazuh-install-files.tar wazuh-install-files/wazuh-passwords.txt

This prints the contents of **`wazuh-passwords.txt`**, including the credentials generated during installation.

3. First login

After installation, access the Wazuh web interface at:

https://<WAZUH_DASHBOARD_IP_ADDRESS>

Use:

- **Username:** `admin`
- **Password:** the one shown at the end of the installation, or the one retrieved from `wazuh-passwords.txt`.

The browser may show a certificate warning on first login because the default certificate is not issued by a trusted authority.

4. Disable the Wazuh repository after installation

Wazuh recommends disabling the Wazuh package repository after installation to prevent **accidental upgrades** that could break the environment.
This does **not** mean Wazuh should never be upgraded; it means upgrades should be performed **intentionally and in a controlled way**. Wazuh's upgrade documentation states that the **Wazuh server, indexer, and dashboard must remain on identical version numbers, including patch level**

Disable the Wazuh APT repository

Run:

sudo sed -i "s/^deb /#deb /" /etc/apt/sources.list.d/wazuh.list && sudo apt update

This comments out the active Wazuh repository line in `/etc/apt/sources.list.d/wazuh.list` and refreshes APT.

5. Enable the Wazuh repository again for a planned upgrade

When a **planned Wazuh upgrade** is required, re-enable the repository with:

sudo sed -i "s/^#deb /deb /" /etc/apt/sources.list.d/wazuh.list && sudo apt update

This restores the Wazuh APT repository line so that the packages can be upgraded in a controlled maintenance window.

6. Verify whether the repository is disabled

To confirm the Wazuh repository is disabled, check the repository file:

cat /etc/apt/sources.list.d/wazuh.list

Expected result when disabled

#deb [signed-by=/usr/share/keyrings/wazuh.gpg] https://packages.wazuh.com/4.x/apt/ stable main

If the line starts with `#deb`, the Wazuh repository is disabled. If it starts with `deb`, the repository is enabled.

Supported Monitoring Methods

Wazuh supports multiple ways of collecting and monitoring security data, depending on the type of endpoint or service being monitored. In addition to the standard **agent-based monitoring** model, Wazuh also supports **Syslog ingestion**, **agentless monitoring over SSH**, and **API-based integrations** for cloud and SaaS platforms.

1. Agent-based monitoring

The primary Wazuh monitoring method is the **Wazuh agent**, which is installed on monitored endpoints and sends security events and telemetry to the Wazuh manager over an encrypted and authenticated channel. This is the standard approach for monitoring servers, workstations, and other supported operating systems.

2. Syslog monitoring

Wazuh can also collect logs from devices that do not support agent installation, such as **firewalls, routers, and switches**, by receiving events through **Syslog**. On the Wazuh server, Syslog inputs can be configured to use either **TCP or UDP**, depending on the source device and deployment requirements.

3. Agentless monitoring over SSH

For systems where an agent cannot be installed, Wazuh provides **agentless monitoring** using the **SSH protocol**. This method allows Wazuh to connect remotely to supported endpoints and either monitor files, directories, and configurations, or run commands periodically and analyze their output.

4. API-based cloud and service monitoring

Wazuh also supports monitoring at the cloud and service level through **API-based integrations** instead of relying only on local endpoint agents. Wazuh documents this approach for platforms such as **AWS**, **Microsoft Azure**, **Google Cloud**, and **Office 365 / Microsoft 365**, where it collects and analyzes activity data directly from provider services and APIs.

Examples of API-based integrations

- **AWS** – Wazuh supports monitoring AWS services such as **CloudTrail** and other AWS sources through its cloud monitoring modules.
- **Microsoft Azure** – Wazuh can retrieve data through the **Azure Log Analytics REST API**, Azure Storage, and the **Microsoft Graph API**.
- **Office 365 / Microsoft 365** – Wazuh monitors audit activity using the **Office 365 Management Activity API**.
- **Google Cloud** – Wazuh supports integrations with **Google Cloud Pub/Sub** and **Google Cloud Storage buckets** for collecting cloud events and logs.

How to Deploy a Wazuh Agent?

Overview

The **Wazuh agent** is installed on endpoints that need to be monitored and communicates with the **Wazuh manager** over an encrypted and authenticated channel. Once installed and enrolled, the agent can provide endpoint visibility such as log collection, file integrity monitoring, security configuration assessment, system inventory, malware detection, and active response.

This procedure describes the **recommended deployment method**: generating the installation command directly from the **Wazuh Dashboard** and running it on the target endpoint. Wazuh documents this as a supported deployment path through the dashboard.

Purpose

This page explains how to deploy a Wazuh agent by using the **Deploy new agent** function in the dashboard, which generates a command that installs, enrolls, and configures the agent for the selected operating system.

Recommended Deployment Method: Dashboard-Generated Install Command

Step 1 – Open the Wazuh Dashboard

Log in to the **Wazuh Dashboard** and navigate to **Agents management** > **Summary**.

Step 2 – Start the Agent Deployment Wizard

Click **Deploy new agent**.

Step 3 – Select the Endpoint Configuration

In the deployment wizard, fill in the required endpoint details such as:

- **Operating system** of the endpoint.
- **Wazuh manager address** that the agent should connect to.
- Optional values such as **agent name**, **agent group**.

Step 4 – Copy the Generated Command

After the required values are entered, the dashboard generates a **ready-to-run install command**. This command is designed to install the agent and configure it to enroll with the specified Wazuh manager.

Step 5 – Run the Command on the Target Endpoint

Run the generated command on the target system using an account with the required privileges. The exact command varies by operating system, but the purpose is the same: **install the agent, configure the manager address, and enroll the endpoint**.

What the Dashboard Command Does

The generated command typically performs the following tasks:

1. Installs the **Wazuh agent** package for the selected operating system.
2. Configures the agent to communicate with the specified **Wazuh manager**.
3. Enrolls the endpoint so the manager can generate a unique client key and authorize the agent.
4. Starts the agent service so it begins sending data to the Wazuh manager. Wazuh's installation guide describes enabling and starting the `waz uh-agent` service after installation.

Verify the Agent Installation

Verify in the Dashboard

After the command runs successfully, go back to **Agents management** > **Summary** in the Wazuh Dashboard. Here you should see the new agent
/endpoint.

Expected statuses include:

- **Active**

- DISCONNECTED

- **Pending**

- NEVER CONNECTED

The deployment should be considered successful when the agent appears as **Active**.

Wazuh Agent Permissions on Linux

Summary

The Wazuh agent on Linux is installed and managed as a **system service**, and installation/startup requires **root privileges**. However, the agent is made up of multiple processes, so **Wazuh does not run entirely under one single user account**. Different components can run with different permissions.

Observed process permissions

In the tested environment, the main Wazuh agent processes were running as:

- `wazuh-execd` **root**
- `wazuh-modulesd` **root**
- `wazuh-agentd` **wazuh**

This shows that **script and command execution rights depend on which Wazuh component is launching the action**, not just on "the agent" as a whole.

What this means

- **Active Response scripts** are typically handled by `wazuh-execd`, so if `wazuh-execd` runs as **root**, those scripts usually already have elevated privileges and do **not** need `sudo`.
- **Command wodle / remote command execution** is handled by `wazuh-modulesd`, so if `wazuh-modulesd` runs as **root**, those commands also usually do **not** need `sudo`.
- `wazuh-agentd` is responsible for agent communication and may run as the `wazuh` user, meaning **not every Wazuh process has root privileges**.

What Wazuh can do

- Run **Active Response scripts** on monitored endpoints when alerts are triggered.
- Run **commands or scripts periodically** using the **Command wodle**.
- Accept **remote command configuration from the manager**, but only if explicitly enabled on the agent with `wazuh_command. remote_commands=1`.
- Validate command/script integrity using **MD5, SHA1, or SHA256**.

What Wazuh cannot do by default

- It cannot be assumed that **all Wazuh processes run as root**; permissions are **module-dependent**.
- **Remote commands are not enabled by default** and must be explicitly allowed on the agent.
- Wazuh scripts should **not rely on interactive `sudo` prompts**, because they run non-interactively.

Agent Groups – Managed Configuration Approach

Purpose

Agent groups are used as the **managed way to apply configuration** to endpoints with similar roles, operating systems, or application profiles.

Instead of maintaining one-off configurations per host, systems are assigned to a relevant agent group. This makes configuration easier to standardize, maintain, and extend over time.

Why we use agent groups

Using agent groups provides several advantages:

- **Centralized management** of configuration for similar systems
- **Consistency** across hosts with the same role
- **Scalability** when onboarding new endpoints
- **Simpler maintenance** compared to per-agent customization
- **Cleaner separation** between baseline

settings and role-specific settings This approach

allows us to apply configurations in a structured

way, for example:

- Linux_servers
- Linux-web
- windows_workstations_modern
- Windows-IIS

Configuration approach

The intended approach is to use **agent groups as the primary mechanism for configuration scoping**.

This means:

1. Agents are assigned to a group based on **OS**, **role**, or **application profile**
2. Group-specific configuration can then be managed centrally
3. New systems should be aligned to an existing group whenever possible
4. New groups should only be created when there is a **clear operational need** for separate configuration

You can **create a group and edit** an agent configuration inside the dasboard in **Agent management** or you can create a group at ***/var/ossec/etc***
/shared on the Wazuh manager.

Wazuh Agentless Monitoring

What it is?

Wazuh agentless monitoring lets the Wazuh manager connect to a remote machine over SSH and run checks without installing a Wazuh agent on that machine.

When to use it

Use agentless monitoring when:

- you cannot install a Wazuh agent on the target machine,
- the machine has SSH enabled,
- you want to monitor file changes, configuration drift, or command output changes.

Step 1 — Install expect

On the Wazuh manager:

sudo apt install -y expect

Wazuh documents installing `expect` before configuring the agentless module.

Step 2 — Create an SSH key on the Wazuh manager

Generate an SSH key for the `wazuh` user. For a simple setup, use no passphrase so Wazuh can

connect automatically using the following command: sudo -u wazuh ssh-keygen

Step 3 — Copy the public key to the remote machine

Copy the public key (`.pub`) to the target machine:

ssh-copy-id -i /path/to/your/public/key.pub user@target-ip

Example:

ssh-copy-id -i /home/wazuh/.ssh/id_ed25519.pub wazuh@172.16.99.254

Step 4 — Register the host in Wazuh

Add the remote machine as an agentless host:

/var/ossec/agentless/register_host.sh add user@target-ip NOPASS

Example:

sudo /var/ossec/agentless/register_host.sh add wazuh@192.168.20.144 NOPASS

To confirm the host is registered:

sudo /var/ossec/agentless/register_host.sh list

Step 5 — Add the agentless block to ossec.conf

Edit the Wazuh

manager config

file: sudo nano

/var/ossec/etc/os

sec.conf

Example for agentless monitoring:

```
<ossec_config>
 <agentless>
  <type>ssh_integrity_check_linux</type>
  <frequency>3600</frequency>
  <host>wazuh@172.16.99.254</host>
  <state>periodic_diff</state>
  <arguments>/etc /bin /sbin /usr/bin /usr/sbin /etc/cron.d /etc/cron.daily /etc/cron.hourly
  /etc/cron.weekly /etc/cron.monthly</arguments>
 </agentless>

 <agentless>
  <type>ssh_generic_diff</type>
  <frequency>3600</frequency>
  <host>wazuh@172.16.99.254</host>
  <state>periodic_diff</state>
  <arguments>sshd -T | sort</arguments>
 </agentless>

 <agentless>
  <type>ssh_generic_diff</type>
  <frequency>3600</frequency>
  <host>wazuh@172.16.99.254</host>
  <state>periodic_diff</state>
  <arguments>systemctl list-unit-files --type=service --state=enabled</arguments>
```

</agentless>

</ossec_config>

The `ssh_integrity_check_linux` is for Linux file and directory integrity checks, and `periodic_diff` to alert when the current result changes compared to the previous run.

What the fields mean

- `type`: the check type to run. For Linux file integrity, use `ssh_integrity_check_linux`.
- `frequency`: how often the check runs, in seconds.
- `host`: the remote SSH account in `user@hostname` format.
- `state`: `periodic_diff` compares the current output with the previous run and alerts on change.
- `arguments`: the files or directories to monitor. Multiple paths are separated by spaces.

Step 6 — Restart Wazuh

After saving the config, restart the manager:

sudo systemctl restart wazuh-manager

Step 7 — Check that it works

Check the Wazuh manager log:

grep agentless /var/ossec/logs/ossec.log

You should see log lines similar to:

- `wazuh-agentlessd: INFO: Test passed for 'ssh_integrity_check_linux'.`
- `wazuh-agentlessd: INFO: ssh_integrity_check_linux: user@host: Starting.`
- `wazuh-agentlessd: INFO: ssh_integrity_check_linux: user@host: Finished.`

Wazuh Syslog Forwarding

What it is

Syslog forwarding lets a device or server send logs to the Wazuh manager without installing a Wazuh agent on that source system. Wazuh can receive syslog from endpoints such as firewalls, switches, routers, and other devices that do not support agent installation.

When to use it

Use syslog forwarding when:

- the device cannot run a Wazuh agent,
- the device or server can send syslog over TCP or UDP,
- you want the source system to push logs to Wazuh instead of Wazuh pulling data over SSH.

Important difference from agentless monitoring

- Agentless monitoring uses SSH from the Wazuh manager to the remote host and is meant for periodic checks.
- Syslog forwarding means the remote system sends logs to Wazuh over syslog.

Step 1 — Configure Wazuh to receive syslog

Edit the Wazuh manager config file

using the following command: sudo

nano /var/ossec/etc/ossec.conf

Add a new `<remote>` block for syslog inside `<ossec_config>`.

Example: Syslog over TCP

```
<remote>
  <connection>syslog</connection>
  <port>5514</port>
  <protocol>tcp</protocol>
  <allowed-ips>172.16.96.112/32</allowed-ips>
  <local_ip>172.16.99.181</local_ip>
</remote>
```

Wazuh supports both TCP and UDP for syslog listeners.

Step 2 — Restart the Wazuh manager

After saving `ossec.conf`, restart Wazuh using the following command:

sudo systemctl restart wazuh-manager

Step 3 — Configure the endpoint sender with rsyslog

On the remote machine, configure rsyslog to forward logs to the Wazuh manager.

Example config to put into rsyslog.conf or create an aditional configuration file.

. action(type="omfwd" target="172.16.99.181" port="5514" protocol="tcp")

Important matching rule

The sender and the Wazuh listener must match:

- If Wazuh listens on TCP/514, rsyslog must send to TCP/514.
- If Wazuh listens on UDP/514, rsyslog must send to UDP/514.
- If you choose a different port like 5514, Wazuh must also listen on 5514.

Step 4 — Restart rsyslog on the sender

After saving the rsyslog config:

sudo

systemctl

restart

rsyslog

This

applies the

forwarding

rule.

Step 5 — Allow the firewall

Make sure the Wazuh server allows the port and protocol you chose. Example: sudo ufw allow 5514/tcp

Wazuh supports TCP and UDP for syslog input, so the firewall must match the configured listener.

Step 6 — Test the forwarding

On the Linux sender, create a test log message:

logger "test message to Wazuh"

Wazuh Agent Enrollment Password Configuration

Purpose

This page explains how to require a **shared password** during **Wazuh agent enrollment**, so new agents cannot register to the Wazuh manager unless they know the configured enrollment password. Wazuh documents this as **password authentication** for agent enrollment.

Scope

This procedure applies to **Wazuh agent enrollment** through the manager's enrollment service (`wazuh-authd`) on **TCP port 1515**, which Wazuh uses for agent registration over TLS.

How it works

When password authentication is enabled, the Wazuh manager requires a **shared enrollment password** before issuing an agent key during enrollment. The password is read from `/var/ossec/etc/authd.pass`; if that file does not exist, Wazuh can generate a random password on restart.

Configuration on the Wazuh Manager

1. Enable password authentication in `ossec.conf`

Add the following setting to the `<auth>` section of `/var/ossec/etc/ossec.conf`: `<use_password>yes</use_password>` enables shared password authentication for agent enrollment.

2. Create the enrollment password file

Create `/var/ossec/etc/authd.pass` on the manager and place the chosen enrollment password in the file.

3. Set the correct permissions and ownership

Wazuh's documented recommendation is to set the file mode to `640` and the ownership to root:wazuh using the following commands:

chmod 640
/var/ossec/etc/authd
.pass chown
root:wazuh
/var/ossec/etc/authd
.pass

4. Restart the Wazuh manager

Restart the manager so the change takes effect.

using the following command: sudo systemctl

restart wazuh-manager

Wazuh Log Retention – 365 Days

4.3. Purpose

This page explains how to configure Wazuh so that alerts are retained for 365 days in both the Wazuh indexer (searchable in the Dashboard) and the Wazuh manager (local rotated log files on disk). It also includes an optional section for enabling logall / archives later and keeping those for 365 days too.

1. Background: What retention means in Wazuh

The Wazuh indexer stores searchable data in indices such as `wazuh-alerts-*`, which is the index pattern for alerts generated by the Wazuh server. The Wazuh manager stores local log files under `/var/ossec/logs/` and rotates/compresses them over time. These logs are stored on the server and managed by the manager's log rotation process.

2. Configure the Wazuh indexer (alerts only)

Wazuh uses Index State Management (ISM) policies to control how long indexed data is kept.

2.1 Create the retention policy in the Dashboard

Go to:
Indexer management Index Management State management policies Create policy. Use the JSON editor and create the following policy for alerts:

```
{
 "policy": {
  "policy_id": "wazuh-
  retention-365d",
  "description": "Keep
  Wazuh alerts for 365
  days",
  "schema_version": 17,
  "error_notifi
  cation":
  null,
  "default_sta
  te":
  "retention_
  state",
  "states": [
   {
    "name
    ":
    "reten
    tion_s
    tate",
    "actio
    ns": [],
    "transitions": [
     {
      "state_name"
      :
      "delete_old_i
      ndices",
      "conditions": {
       "min_index_age": "365d"
      }
     }
    ]
   },
   {
    "name":
    "delete_
    old_indic
    es",
    "actions"
```

```
      : [
       {
        "retry": {
         "count": 3,
         "bac
         koff":
         "exp
         onen
         tial",
         "dela
         y":
         "1m"
        },
        "delete": {}
       }
      ],
      "transitions": []
     }
    ],
    "ism_template": [
     {
      "
       i
       n
       d
       e
       x
       _
       p
       a
       t
       t
       e
       r
       n
       s
       "
       :
       [
       "
       w
       a
       z
       u
       h
       -
       a
       l
       e
       r
       t
       s
       -
       *
       "
      ],
      "priority": 1
     }
    ]
   }
  }
```

This policy keeps each alert index until it becomes 365 days old, then deletes that old index automatically. The `ism_template` makes the policy apply to future indices that match `wazuh-alerts-*`.

2.2 Apply the policy to existing indices

Creating the policy is not enough for indices that already exist. Existing indices must be selected in the Indexes view and the policy must be applied manually once.
Go to:
Indexer management Index Management Indexes. Then:

1. Search for `wazuh-alerts-*`.
2. Select the existing `wazuh-alerts-*` indices.
3. Click Actions Apply policy.
4. Choose `wazuh-retention-365d`.
5. Apply the policy.

2.3 Verification

After applying the policy the Managed by column for the selected indices should show Yes, and new daily `wazuh-alerts-*` indices should also become managed automatically because of the template.

3. Configure the Wazuh manager (local file retention)

Wazuh documents an internal setting called `monitord.keep_log_days`, which controls the number of days to keep rotated internal logs. The custom internal changes must be placed in `/var/ossec/etc/local_internal_options.conf`, because `internal_options.conf` may be overwritten during upgrades.

3.1 Edit the manager retention file

On the Wazuh manager, edit:

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

Add:

monitord.keep_log_days=365

This changes the manager-side local retention to 365 days for rotated log files.

3.2 Restart the manager

After saving the file, restart the Wazuh manager:

sudo systemctl restart wazuh-manager

Restarting the manager ensures configuration changes are applied.

4. Final result for alerts-only retention

After completing the steps above, the Wazuh indexer keeps alert indices (`wazuh-alerts-*`) for 365 days, and the Wazuh manager keeps rotated local log files for 365 days.

This means the environment keeps approximately the latest 365 days of alerts in a rolling window, both as searchable data in the Dashboard and as local rotated files on the manager.

5. Optional: If you want to enable logall / archives too

If you later decide to keep all events (not just alerts), Wazuh supports this through `logall_json` and the `wazuh-archives-*` index pattern. The `ar chives.json` stores all events when `logall_json` is enabled, and `wazuh-archives-*` is the index pattern for all events sent to the server.

5.1 Enable all-events archive output on the manager

Edit:

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

Inside the `<global>` section, add:

<logall_json>yes</logall_json>

This tells Wazuh to write all events to `/var/ossec/logs/archives/archives.json`.
Restart the manager:

sudo systemctl restart wazuh-manager

Restarting the manager applies the new configuration.

5.2 Enable archives in Filebeat

Edit:

sudo nano /etc/filebeat/filebeat.yml

Set the Wazuh module as follows:

filebeat.modules:
-
m
od
ul
e:
w
az
uh
al
ert
s:
e

```
  n
  a
  b
  l
  e
  d
  :
  t
  r
  u
  e
  a
  r
  c
  h
  i
  v
  e
  s
  :
   enabled: true
```

This makes Filebeat ship archive events to the indexer in addition to alerts. Restart Filebeat:

```
sudo systemctl restart filebeat
```

5.3 Add 365-day retention for archives in the indexer

The `wazuh-archives-*` is the index pattern for all events. To keep archive indices for 365 days, create a policy for `wazuh-archives-*` or extend the existing policy template to include that pattern.
Example archive retention policy:

```
{
 "policy": {
  "policy_id": "wazuh-archives-
  retention-365d",
  "description": "Keep Wazuh
  archives for 365 days",
  "schema_version": 17,
  "error_notifi
  cation":
  null,
  "default_sta
  te":
  "retention_
  state",
  "states": [
   {
    "name
    ":
    "reten
    tion_s
    tate",
    "actio
    ns": [],
    "transitions": [
```

```
      {
        "state_name"
        :
        "delete_old_i
        ndices",
        "conditions": {
          "min_index_age": "365d"
        }
      }
    ]
  },
  {
    "name":
    "delete_
    old_indic
    es",
    "actions"
    : [
      {
        "retry": {
          "count": 3,
          "bac
          koff":
          "exp
          onen
          tial",
          "dela
          y":
          "1m"
        },
        "delete": {}
      }
    ],
    "transitions": []
  }
],
" ism_template": [
  {
    "i
      n
      d
      e
      x
      _
      p
      a
      tt
      e
      r
      n
      s
      "
      :
      [
      "
      w
      a
      z
      u
      h
      -
```

```
      a
      r
      c
      h
      i
      v
      e
      s
      -
      *
      "
     ],
     "priority": 1
    }
   ]
  }
}
```

Then, as with alerts, go to Indexes and apply the archive retention policy to any existing `wazuh-archives-*` indices once. Future archive indices will then inherit the policy automatically through the template.

5.4 Manager retention remains the same

If `monitord.keep_log_days=365` is already configured, that manager-side local retention setting can stay in place when `logall_json` is enabled. The manager will continue to rotate/compress local log files, and the rotated files are kept for the configured number of days.

How to Create a New Custom Rule?

1. Purpose

This page explains how to create a **custom Wazuh rule** and apply it through the **Wazuh dashboard**. The example used in this guide is a rule that generates an alert when **Syscollector detects a newly opened listening port** on a monitored endpoint. Wazuh documents that Syscollector inventory events use **rule ID 221** at **level 0** by default, and that a visible custom rule can be created by inheriting from that event.

2. Why this example is useful

Detecting a newly opened listening port is valuable because it can indicate:

- a new service was installed,
- remote access was enabled,
- a change was made outside of change control,
- or suspicious software opened a listener. Wazuh's **System inventory / Syscollector** capability is designed to collect ports, services, processes, packages, and other endpoint state data specifically for this kind of visibility.

For older systems or legacy environments, this kind of rule is a good first custom detection because it is **easy to understand**, **actionable**, and based on a documented Wazuh use case.

3. Background

Wazuh's **Syscollector** module is enabled by default on agents and collects inventory information such as hardware, operating system, installed software, network data, ports, processes, users, groups, and services. The default Syscollector configuration includes:

- `<interval>1h</interval>`
- `<scan_on_start>yes</scan_on_start>`
- `<ports all="no">yes</ports>`
  This means Syscollector scans **listening ports** by default, runs every **1 hour** by default, and performs an immediate scan when the agent service starts.

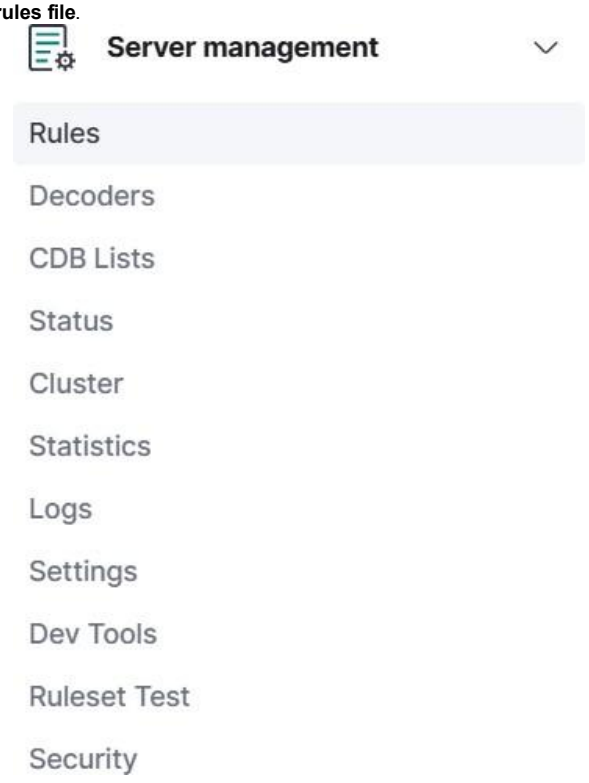
Wazuh also documents that **the initial scan does not generate alerts** for Syscollector change rules. Alerts are only triggered when a later scan detects a **difference** between scans.

4. Example rule objective

Goal: create a custom alert when a monitored endpoint starts listening on a **new port** that was not present in the previous Syscollector scan. Wazuh' s documentation provides an example custom rule chain for **opened**, **modified**, and **closed** ports using **Syscollector rule 221** as the parent event.

5. Where to add the rule in the dashboard

In the **Wazuh dashboard**, go to **Rules** in **Server management** section. Here you can see all the rules. At the top you can **add new rules file**.



6. Custom rule XML

Paste your XML into a **custom rules file** in the Wazuh dashboard, for example `100311-syscollector-ports.xml`. You can use Custom rule IDs in the **100000–120000** range.

EXAMPLE:

```
<group name="syscollector,custom_inventory,">
 <rule id="100310" level="3">
  <if_sid>221</if_sid>
  <field name="type">dbsync_ports</field>
  <description>Syscollector ports event.</description>
 </rule>


 <rule id="100311" level="8">
  <if_sid>100310</if_sid>
  <field name="operation_type">INSERTED</field>
  <description>New listening port detected: $(port.local_port), local IP: $(port.local_ip).</description>
 </rule>
</group>
```

Explanation

- **Rule `100310`** acts as a base rule for Syscollector **port** events by inheriting from **rule 221** and filtering on `type = dbsync_ports`. Wazuh documents this exact structure in its Syscollector custom rule example.
- **Rule `100311`** only fires when `operation_type = INSERTED`, which means a **new port** appeared compared to the previous scan. Wazuh uses the same event logic in the documented example for port creation detection.
- The description uses **`$(port.local_port)`** and **`$(port.local_ip)`** because those are the documented Syscollector fields shown in Wazuh's example.

7. Apply the rule

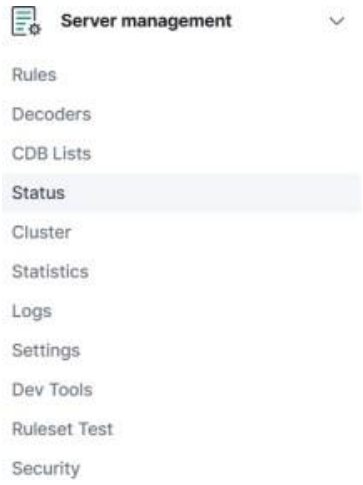
After saving the rule, you need to restart the **Wazuh manager** so the rule is loaded for live alerts. You can do this in the **terminal** of the Wazuh Manager/server **or** in the **dashboard**.

TERMINAL COMMAND:

```
sudo systemctl restart wazuh-manager
```

DASHBOARD:

Go to status and click restart manager at the top right corner.



Discord Integration For Wazuh

4.4. Overview

This guide explains how to configure Wazuh to send alerts to a Discord

channel using a custom integration script. The integration works by:

1. Wazuh triggering an alert
2. Calling a custom script
3. Sending the alert to Discord via webhook

Create Python integration script

Path:

`/var/ossec/integrations/`

Put the custom-discord.py and custom-discord script at that location.

Set permissions

sudo chown root:wazuh
/var/ossec/integrations/custom-
discord.py sudo chmod 750

/var/ossec/integrations/custom-discord.py

sudo chown root:wazuh /var/ossec/integrations/custom-discord sudo chmod 750 /var/ossec/integrations/custom-discord

Configure `ossec.conf`

File:

`/var/ossec/etc/ossec.conf`

Add integration block

```
<integration>
  <name>custom-discord</name>
  <hook_url>https://discord.com/api/webhooks/YOUR_WEBHOOK_URL</hook_url>
  <level>12</level>
  <alert_format>json</alert_format>
</integration
```

Restart Wazuh

Restart Wazuh manager using

the following command: sudo

systemctl restart wazuh-

manager

Verification

Check Wazuh logs using

the following command:

grep custom-discord

/var/ossec/logs/ossec.log

Expected output:

```
INFO: Enabling integration for: 'custom-discord'.
```

Bijlage B: MISP Documentation

MISP DEPLOYMENT AND CONFIGURATION 61
1. Purpose 61
2. Installation Summary 61
3. Why MISP was deployed 61
4. Feed Configuration 62
4.1 Important distinction: enable vs cache vs fetch 62
4.2 Enabled feeds used in this setup 62
4.3 Why these feeds were chosen 63
5. Warninglists Configuration 64
5.1 What warninglists are for 64
5.2 Enabled warninglists used in this setup 64
5.3 Why these warninglists were chosen 65
6. Operational Lessons Learned in MISP 65
6.1 Feeds had to be fetched into the database 65
6.2 Warninglists are only a false-positive control layer 66

MISP AUTOMATED UPDATES AND FEED FETCHING 66
Overview 66
Goal 66
Manual Fetch Command 66
Automated Daily Fetch at 05:00 67
Verification 67

MISP WAZUH INTEGRATION AND VALIDATION 67
1. Purpose 67
2. Architecture Overview 68
3. MISP User for Wazuh 68
3.1 Why a separate user was created 68
3.2 Role used 68
3.3 API key / AuthKey 68
4. Wazuh-Side Integration Configuration 68
4.1 Manager-side integration block 68
4.2 Integration script location 69
4.3 Script permissions 69
5. Example Wazuh Integration Block 69
6. Wazuh Rules for MISP Integration 69
7. Certificate / HTTPS Notes for the Wazuh MISP API Path 70
8. Testing Procedure 70
8.1 Why EICAR was used 70
8.2 Required preparation in MISP 70
8.3 Required preparation in Wazuh 71
8.4 Validation steps 71
9. Troubleshooting Notes 71

9.1 No MISP match even though the file hash is correct 71
9.2 wazuh-integratord errors seen during troubleshooting 72
9.3 Useful Wazuh logs during troubleshooting 72
10. Final Notes 72

WAZUH + MISP MALICIOUS IP LOOKUP WITH ACTIVE RESPONSE 72

Goal72
High-level flow 72
Prerequisites 73
Step 1 — Start Apache2 on the protected endpoint 73
Step 2 — Monitor the Apache access log with the Wazuh agent 73
Step 3 — Use a real Wazuh web-attack alert as the trigger 73
Step 4 — Add the malicious IP to MISP (for testing) 73
Step 5 — Place the custom integration script on the Wazuh manager 73
Step 6 — Add the Wazuh custom integration block 74
Step 7 — Add custom local rules for the MISP match 74
Step 8 — Configure Active Response 75
Step 9 — Test the workflow 75

MISP Deployment and Configuration

1. Purpose

This page documents the MISP deployment used as the threat-intelligence platform for later Wazuh integration. It explains:

- why MISP was deployed,
- how MISP was installed,
- which feeds were enabled,
- which warninglists were enabled,
- why selected feeds had to be enabled and fetched into the database,
- and what operational lessons were learned during setup.

2. Installation Summary

2.1 Deployment method

MISP was installed using the official Ubuntu 24.04 installation script provided by the MISP project:

https://raw.githubusercontent.com/MISP/MISP/refs/heads/2.5/INSTALL/INSTALL.ubuntu2404.sh

2.2 Environment details

- Installation method: official MISP Ubuntu 24.04 installation script
- Operating system: Ubuntu 24.04
- MISP URL: `https://misp.local` or `https://172.16.96.73`
- Certificate model: self-signed HTTPS certificate
- Certificate update performed later: the certificate was re-created so it included both:
  - `misp.local`
  - `172.16.96.73`

3. Why MISP was deployed

MISP was deployed as the central platform to:

- ingest external threat-intelligence feeds,
- normalize indicators into searchable MISP events and attributes,
- reduce noise and false positives through warninglists,
- and provide a threat-intelligence API backend for Wazuh.

Long-term goal

The long-term goal was to use MISP as a searchable

IOC source for Wazuh, especially for:

- file hashes,
- URLs,
- domains,
- and malicious ip addresses.

4. Feed Configuration

4.1 Important distinction: enable vs cache vs fetch

This distinction was critical during setup:

- Enabled feed → the feed is active and available for processing.
- Cached feed → feed content is downloaded for correlation / feed-hit lookups, but is not stored as normal MISP events in the database.
- Fetched feed → feed data is ingested into the MISP database as real searchable events/attributes.

Why fetching into the database matters

For the Wazuh integration, MISP is queried through the API for actual searchable indicators. That means the relevant IOC data must exist as normal MISP events/attributes in the database, not only in cache.

Operational workflow used

For the feeds intended for operational use, the workflow was:

1. Enable the feed
2. Fetch and store the feed data into the MISP database

4.2 Enabled feeds used in this setup

The following feeds were enabled:

1. CIRCL OSINT Feed
2. The Botvrij.eu Data
3. Phishtank online valid phish
4. Feodo IP Blocklist
5. OpenPhish url list
6. abuse.ch SSL IPBL
7. URLHaus Malware URLs
8. ThreatFox
9. MalwareBazaar

Feed observations

- Several of these feeds use MISP format, which makes them useful for ingestion as real MISP events/attributes.
- Others use CSV or freetext format and were enabled because they provide phishing, URL, or malicious ip addresses coverage.
- Together they provide a balanced IOC source set for:
    - malware hashes,
    - phishing URLs,
    - domains,
    - malicious IPs,
    - and broader OSINT content.

4.3 Why these feeds were chosen

CIRCL OSINT Feed

Used as a broad OSINT source to populate MISP with general indicators.

Botvrij.eu Data

Used as an additional OSINT source to improve general IOC coverage.

Phishtank online valid phish

Used to add phishing URL intelligence.

Feodo IP Blocklist

Used for malicious IP and infrastructure coverage.

OpenPhish url list

Used to improve phishing URL coverage.

abuse.ch SSL IPBL

Used for malicious SSL infrastructure coverage.

URLHaus Malware URLs

Used for malware-delivery URL coverage.

ThreatFox

Used because it provides high-value malware-related IOCs in MISP format.

MalwareBazaar

Used because it provides malware hash intelligence and is useful for file-hash lookups.

5. Warninglists Configuration

5.1 What warninglists are for

Warninglists were enabled to reduce obvious false positives and operational noise.

Important design note

Warninglists are not the same as real IOC attributes/events.
Enabling a warninglist does not create a normal malicious IOC that Wazuh can match through the MISP file-hash integration. Warninglists only mark values as possible false positives or cautionary values.

5.2 Enabled warninglists used in this setup

The following warninglists were enabled:

1. TLDs as known by IANA
2. Hashes that are often included in IOC lists but are false positives
3. List of known sinkholes
4. Second level TLDs as known by Mozilla Foundation
5. List of RFC 6761 Special-Use Domain Names
6. List of RFC 6598 CIDR blocks
7. List of RFC 5735 CIDR blocks
8. List of RFC 3849 CIDR blocks
9. List of RFC 1918 CIDR blocks
10. List of known IPv6 public DNS resolvers
11. List of known IPv4 public DNS resolvers
12. List of known hashes for benign files
13. List of RFC 5771 multicast CIDR blocks
14. List of IPv6 link local blocks
15. List of known hashes for empty files

5.3 Why these warninglists were chosen

These warninglists were enabled to reduce noise from:

- public DNS resolvers,
- private and reserved IP space,
- link-local ranges,
- multicast ranges,
- sinkholes,
- benign file hashes,
- and empty-file hashes.

This keeps the MISP dataset cleaner and reduces avoidable false positives.

6. Operational Lessons Learned in MISP

6.1 Feeds had to be fetched into the database

Simply enabling or caching feeds was not enough for the planned Wazuh IOC lookup workflow. For MISP API searches to return usable indicator data, the selected feeds had to be fetched into the database.

6.2 Warninglists are only a false-positive control layer

Warninglists were useful, but they do not replace real MISP IOC events/attributes. This became important during EICAR testing:

- enabling the EICAR warninglist alone would not create a real IOC match for the Wazuh integration,
- so the EICAR hashes had to be added manually to a MISP test event.

MISP Automated Updates and Feed Fetching

4.5. Overview

In MISP, **enabled feeds are not automatically imported into the database by default**.
Caching only stores feed data in **Redis** for correlations / "Feed hits," while **fetching** is the action that **imports feed data into the MISP database**.

4.6. Goal

The goal is to make MISP **check enabled feeds regularly** and **import new threat intelligence automatically** into the database.
MISP maintainers recommend using **cron jobs with CLI commands** for this instead of relying on Scheduled Tasks.

4.7. Manual Fetch Command

The following command fetches **all enabled feeds** and imports them into the MISP database:

```
sudo -u www-data /var/www/MISP/app/Console/cake Server fetchFeed 1 all
```

Here, `1` is the **numeric MISP user ID** and `all` means **all enabled feeds**.

4.8. Automated Daily Fetch at 05:00

To run the fetch automatically every day at **05:00**, add

this cron job for the `www-data` user: sudo crontab -u

www-data -e

Add:

0 5 * * * /var/www/MISP/app/Console/cake Server fetchFeed 1 all >> /var/log/misp-feed-fetch.log 2>&1

This makes MISP **poll all enabled feeds every day at 05:00** and log the output to `/var/log/misp-feed-fetch.log`.

4.9. Verification

To confirm the cron job is installed:

sudo crontab -u www-data -l

MISP Wazuh Integration and Validation

1. Purpose

This page documents how MISP was integrated with Wazuh so that file hashes detected by Wazuh File Integrity Monitoring (FIM) could be looked up automatically against indicators stored in MISP.

Integration goal:

- Wazuh monitors a file path
- Wazuh extracts file hashes on file creation
- Wazuh queries MISP through the API
- If the hash matches an IOC in MISP, Wazuh generates an alert

2. Architecture Overview

Detection flow:

1. Wazuh FIM monitors a directory and detects file creation/modification.
2. Wazuh generates the base file alert rule 554.
3. The Wazuh Integrator module runs a custom script named `custom-misp_file_hashes.py`.
4. The script queries the MISP API for the file hashes (`md5`, `sha1`, `sha256`).
5. If MISP returns a matching attribute, Wazuh can generate the MISP match alert.

3. MISP User for Wazuh

3.1 Why a separate user was created

A dedicated least-privilege MISP user was created for Wazuh instead of using the admin account.

Reason

- API/AuthKeys inherit the permissions of the user they belong to.
- Using a dedicated read-only user is safer than using an admin API key.

3.2 Role used

A dedicated read-only role was created

for the Wazuh integration with:

- read-only style permissions,
- auth key access enabled,
- and without admin / modify / publish capabilities.

3.3 API key / AuthKey

The Wazuh integration authenticates to MISP using a MISP API/AuthKey associated with that dedicated user account.

4. Wazuh-Side Integration Configuration

4.1 Manager-side integration block

The Wazuh `<integration>` block must be configured in the Wazuh server / manager file:

`/var/ossec/etc/ossec.conf`

4.2 Integration script location

The custom script must be placed in:

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

The script used in this setup was:

```
custom-misp_file_hashes.py
```

4.3 Script permissions

The script permissions were adjusted so Wazuh could execute it correctly. Example used:

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

5. Example Wazuh Integration Block

```
<integration>
  <name>custom-misp_file_hashes.py</name>
  <hook_url>https://misp.local</hook_url>
  <api_key>YOUR_MISP_API_KEY</api_key>
  <group>syscheck</group>
  <rule_id>554</rule_id>
  <alert_format>json</alert_format>
  <options>{"timeout":10,"retries":3,"debug":true}</options>
</integration>
```

Notes

- group=syscheck and rule_id=554 tie the integration to Wazuh FIM file-add alerts.
- alert_format=json is required so the script receives JSON alert data.
- debug=true is useful during setup because the script writes debug output to /var/ossec/logs/integrations.log only when debug is enabled.

6. Wazuh Rules for MISP Integration

The important rule outcomes are:

- `100800` MISP integration event wrapper
- `100801` hash not found (`found = 0`) and this rule is level 0
- `100802` hash matched (`found = 1`) and this is the visible positive alert
- `100803/100804` API / server error conditions

Important note

A working integration can still appear “quiet” if the result is `found = 0`, because the not found rule is level 0.

7. Certificate / HTTPS Notes for the Wazuh MISP API Path

HTTPS trust for the MISP API was one of the main troubleshooting topics during setup.

Key observations

- The Wazuh integration script runs with Wazuh's embedded Python.

- The script uses Python `requests`.
- A self-signed certificate was initially used.
- The certificate had to be re-created so that it included:
    - `misp.local`
    - `172.16.96.73`

Local certificate path used during testing

```
/var/ossec/etc/certs/misp.crt
```

Practical note

The certificate path was used for HTTPS validation tests from the same embedded Wazuh Python runtime.

8. Testing Procedure

8.1 Why EICAR was used

EICAR was used as the safe test artifact for validating the MISP–Wazuh hash lookup flow.

8.2 Required preparation in MISP

To trigger a positive match, the EICAR hashes had to be added to MISP as real searchable attributes, not only handled through warninglists.

Recommended test event

A dedicated event was created:

`Wazuh EICAR test`

Then three separate attributes were added:

- `md5 44d88612fea8a8f36de82e1278abb02f`
- `sha1 3395856ce81f2b7382dee72602f798b642f14140`
- `sha256 275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabf651fd0f`

Important attribute setting

For each of those attributes, this had to be enabled:

`For Intrusion Detection System / IDS / to_ids`

Without `to_ids = true`, the integration script can miss the IOC even if the hash exists in MISP.

Visibility requirement

The test event must be visible to the MISP API user used by Wazuh.

8.3 Required preparation in Wazuh

- Configure Wazuh FIM to monitor the target directory with real-time monitoring and checksum collection.
- Restart the Wazuh agent after changing FIM configuration.

8.4 Validation steps

1. Confirm the EICAR test event exists in MISP and the hash attributes are searchable.
2. Place the EICAR file in the Wazuh-monitored location.
3. Confirm Wazuh generates the base rule 554 alert.
4. Confirm the integration script runs and queries MISP by reviewing `integrations.log`.
5. If the IOC exists and is visible, the integration should return `found = 1` and Wazuh should raise the match alert.

9. Troubleshooting Notes

9.1 No MISP match even though the file hash is correct

Check these in order:

1. The EICAR hash exists in MISP as a real attribute, not only as a warninglist hit.
2. The hash attribute type is exactly `md5`, `sha1`, or `sha256`.
3. The hash attribute has IDS / to_ids enabled.
4. The event is visible to the MISP API user used by Wazuh.
5. If tags are configured in the Wazuh integration options, remove them temporarily during debugging because they can filter out the match.

9.2 wazuh-integratord errors seen during troubleshooting

- Exit status 4 no response from MISP
- Exit status 7 invalid JSON, usually caused by malformed integration options JSON

9.3 Useful Wazuh logs during troubleshooting

```
tail -f /var/ossec/logs/alerts/alerts.log
tail -f /var/ossec/logs/ossec.log |
grep wazuh-integratord tail -f
/var/ossec/logs/integrations.log
```

10. Final Notes

This integration depends on three separate layers all working correctly:

1. Wazuh FIM must generate the file event with hashes.
2. Wazuh Integrator must run the custom script from the manager-side integration block.
3. MISP must return a searchable IOC that is visible to the API user and satisfies the query conditions (`type`, `to_ids`, visibility, optional tags).

When all three are correct, a file placed in the monitored directory that matches an IOC stored in MISP should produce the expected alert flow.

Wazuh + MISP Malicious IP Lookup with Active Response

Goal

This implementation extends the existing Wazuh–MISP integration from **file hashes** to **malicious IP addresses**. The objective is to let Wazuh detect a real web attack from Apache logs, query MISP for the attacker IP, create a dedicated Wazuh alert when MISP confirms the IP, and then temporarily block that IP on the protected endpoint using Wazuh Active Response.

High-level flow

The final workflow is: Apache runs on the protected Ubuntu endpoint, the Wazuh agent monitors the Apache access log, a real web attack (for example SQL injection or Shellshock) triggers a built-in Wazuh alert, the custom Wazuh integration script receives that alert and queries MISP for the attacker IP, Wazuh raises a new “MISP malicious IP matched” alert, and Active Response executes `firewall-drop` on the protected endpoint.

Prerequisites

Before starting, the protected endpoint must have Apache2 installed and have the Wazuh agent already be enrolled and active. The logs folder should also be already monitored with FIM.

MISP must also contain the test malicious IP as a searchable network attribute, typically `ip-src` for the attacker IP in this scenario.

Step 1 — Start Apache2 on the protected endpoint

The first step is to make sure Apache2 is installed and actively running on the protected Ubuntu endpoint.

After Apache2 is running, confirm that the site is reachable locally or from another host. This is important because the later Wazuh detections depend on Apache receiving and logging the incoming requests that will be analyzed.

Step 2 — Monitor the Apache access log with the Wazuh agent

On the protected endpoint, configure the Wazuh agent to monitor the Apache access log. This should already been done if the endpoint is added to the linux-web group.

After updating the agent configuration, restart the Wazuh agent so the log monitoring change is applied.

Step 3 — Use a real Wazuh web-attack alert as the trigger

A normal page visit to Apache is usually **not enough** for this workflow, because Wazuh integrations run on **alerts**, not on every raw log line. Wazuh's data analysis and logtest documentation explain that logs must first be decoded and matched to a rule before they become alerts, and the Integrator only processes those alerts.

For that reason, this implementation uses **existing built-in web-attack detections** as the base trigger.

Step 4 — Add the malicious IP to MISP (for testing)

Create or update a MISP event so the attacker IP is present as a real network IOC, typically as `ip-src` in this scenario.

Step 5 — Place the custom integration script on the Wazuh manager

Copy the malicious-IP integration script to `/var/ossec/integrations/` on the Wazuh manager.

After copying the script, set the permissions with `sudo chmod 750 /var/ossec/integrations/custom-misp_ip_reputation.py` and set the owner with `sudo chown root:wazuh /var/ossec/integrations/custom-misp_ip_reputation.py`. These are the same permission and ownership patterns Wazuh documents for custom integrations.

Step 6 — Add the Wazuh custom integration block

Add the integration block below to `/var/ossec/etc/ossec.conf` on the Wazuh manager. In this implementation, filtering by `rule_id` is used so MISP lookups only happen for real web attacks.

```
<ossec_config>
 <integration>
  <name>custom-misp_ip_reputation.py</name>
  <hook_url>https://YOUR_MISP_SERVER_IP</hook_url>
  <api_key>YOUR_API_KEY</api_key>
  <group>web,accesslog,apache</group>
  <rule_id>31103,31106,31168</rule_id>
  <alert_format>json</alert_format>
  <options>{"timeout":10,"retries":3,"debug":true}</options>
 </integration>
</ossec_config>
```

The chosen rule IDs correspond to built-in web attack detections. Wazuh's SQL injection id `31103` and `31106`, and Wazuh's Active Response example for a web-accesslog attack shows the Shellshock-style alert using rule `31168`.

After changing `ossec.conf`, restart the Wazuh manager with `sudo systemctl restart wazuh-manager`. (or wait till the end to restart the manager)

Step 7 — Add custom local rules for the MISP match

In the UI, dashboard of Wazuh go to rules. There you should see a button 'manage rules file' , than you click the other button called 'import files' and add the `misp_ip_reputation.xml` so that the JSON event returned by the custom integration becomes a visible Wazuh alert.

After saving the local rules, restart the Wazuh manager again so the new rules are loaded. (or wait till the end to restart the manager)

Step 8 — Configure Active Response

On the Wazuh manager, open the main configuration file /var/ossec/etc/ossec.conf and find the active reponse section. Add the following active response configuration on the Wazuh manager. (the first code block should be there already check this)

```
<command>
  <name>firewall-drop</name>
  <executable>firewall-drop</executable>
  <timeout_allowed>yes</timeout_allowed>
</command>
```

```
<active-response>
  <disabled>no</disabled>
  <command>firewall-drop</command>
  <location>local</location>
  <rules_id>100902,100903</rules_id>
  <timeout>300</timeout>
</active-response>
```

The active response will be triggerd when rule `100902` or `100903` is matched.

This configuration means the following:

- `firewall-drop` is the command Wazuh will execute, and on Linux/Unix endpoints it blocks the IP through `iptables`,
- `location = local` means the block happens on the endpoint that generated the MISP match alert,
- `rules_id = 100902` and `100903` ties the response to the custom “MISP malicious IP matched” rule created in the previous step,
- `timeout = 300` makes the block temporary,

After adding or changing the Active Response configuration, restart the Wazuh manager again with `sudo systemctl restart wazuh-manager`.

Step 9 — Test the workflow

To validate the setup, generate a web attack from the attacker host against the monitored Apache endpoint. MISP must also contain the test malicious IP of the attacker by creating a new event with a attribute of this IP.

curl -XGET "http://<UBUNTU_IP>/users/?id=SELECT+*+FROM+users";

Bijlage C: Graylog Documentation

GRAYLOG DOCUMENTATION 76

Graylog Installation Guide 77

Purpose 77

Architecture 77

Step 1 – Prepare both VMs 77

VM 1 – Install MongoDB and Graylog Server 77

Step 2 – Install MongoDB on `graylog01` 77

Step 5 – Configure Graylog Server 79

Step 6 – Set Graylog JVM heap 80

Step 7 – Start Graylog Server 80

VM 2 – Install Graylog Data Node 81

Step 8 – Install the Graylog package repository on the `datanode` 81

Step 9 – Configure Data Node 81

Step 10 – Start Data Node 81

Access the UI 82

Enable HTTPS on Graylog (Direct on Graylog, IP-based) 82

Purpose 82

Assumptions 82

1. Create OpenSSL config 82
2. Generate certificate and key 84
3. Convert the key to PKCS#8 84
4. Copy files to Graylog 84
5. Configure Graylog HTTPS 84
6. Make Graylog trust the self-signed HTTPS certificate 85
7. Restart Graylog 85

Purpose 86

Architecture 86

Prerequisites 86

Graylog Server Configuration 86

Kali Client Configuration 87

Testing - 89

Graylog Installation Guide

Purpose

This page describes how to install **Graylog Open** on **two Ubuntu 24.04 virtual machines** using Graylog's current **Core deployment** model:

- **VM 1**: Graylog Server + MongoDB
- **VM 2**: Graylog Data Node

Architecture

VM Roles

- **VM 1 – `GRAYLOG01`**
  - Graylog Server
  - MongoDB
- **VM 2 – `DATANODE01`**
  - Graylog Data Node

Graylog's official Ubuntu guide states that the **Data Node should be maintained on its own node**, separate from Graylog and MongoDB, for this deployment model.

Recommended starting point:

- **4 vCPU** per VM
- **8 GB RAM** per VM
- **100 GB SSD storage** per VM

Step 1 – Prepare both VMs

Run the following on **both VMs**:

```
sudo apt update && sudo apt upgrade -y
```

VM 1 – Install MongoDB and Graylog Server

Step 2 – Install MongoDB on `graylog01`

Download the correct package from the website:

https://www.mongodb.com/try/download/community Run the following:

```
sudo apt install ./mongodb-org-server_8.2.6_amd64.deb
```

Step 3 – Configure MongoDB (`/etc/mongod.conf`)

Find:

```
net:
  port: 27017
  bindIp: 127.0.0.1
```

Change it to include your IP, for example:

```
net:
  port: 27017
  bindIp: 127.0.0.1,172.16.96.119
```

RESTART MONGODB AND ENABLE MONGODB:

```
sudo systemctl daemon-reload
sudo systemctl enable mongod.service
sudo systemctl restart mongod.service
sudo systemctl status mongod.service
```

Step 4 – Install the Graylog package repository on `graylog01`

Install the package using the following commands:

```
wget https://packages.graylog2.org/repo/packages/graylog-7.0-repository_latest.deb sudo dpkg -i graylog-7.0-repository_latest.deb
sudo apt-get update
```

Install Graylog Server

```
sudo apt-get install -y graylog-server
```

Step 5 – Configure Graylog Server

We need to change the following:

- `password_secret`
- `root_password_sha2`
- `http_bind_address`

These values are stored in /etc/graylog/server/server.conf

Generate password_secret with the following command:

openssl rand -hex 32

Graylog requires the same `password_secret` value on Graylog server and Data Node.

Generate root_password_sha2 with the following command (it will prompt you to enter a password remember this):

echo -n "Enter Password: " && head -1 </dev/stdin | tr -d '\n' | sha256sum | cut -d" " -f1

Edit /etc/graylog/server/server.conf with the following command:

sudo nano /etc/graylog/server/server.conf

Find and change the following:

password_secret =

REPLACE_WITH_SHARED_SECRET

root_password_sha2 =

REPLACE_WITH_GENERATED_HAS

H http_bind_address =

172.16.96.119:9000

message_journal_max_age = 72h

message_journal_max_size = 10gb

Graylog recommends setting journal retention to 72 hours and sizing the journal according to expected log volume over that window.

Step 6 – Set Graylog JVM heap

Edit with the

following

command: sudo

nano

/etc/default/grayl

og-server

Example for an **8 GB** VM:

GRAYLOG_SERVER_JAVA_OPTS="-Xms4g -Xmx4g -server -XX:+UseG1GC -XX:-OmitStackTraceInFastThrow"

Graylog's docs recommend setting heap to **half of available RAM**, with min and max values equal where possible.

Step 7 – Start Graylog Server

Using the following commands:

sudo systemctl daemon-reload

sudo systemctl enable

graylog-server.service

sudo systemctl start

graylog-server.service

sudo systemctl status

graylog-server.service

VM 2 – Install Graylog Data Node

Step 8 – Install the Graylog package repository on the `datanode`

Using the following commands:

wget
https://packages.graylog2.org/repo/packages/graylo
g-7.0-repository_latest.deb sudo dpkg -i graylog-
7.0-repository_latest.deb
sudo apt-get update

Install Data Node using the

following command: sudo apt-get

install -y graylog-datanode

Step 9 – Configure Data Node

Edit it using the following command:

sudo nano /etc/graylog/datanode/datanode.conf

Find and **change** the following(or add):

password_secret =

REPLACE_WITH_SHARED_SECRET

mongodb_uri =

mongodb://172.16.96.119:27017/graylo

g opensearch_heap = 4g

`password_secret` must match the

Graylog server value, `mongodb_uri`

defines the MongoDB connection,

`opensearch_heap` should be set to

half of system memory

Step 10 – Start Data Node

sudo systemctl daemon-reload

sudo systemctl enable

graylog-datanode.service

sudo systemctl start

graylog-datanode.service

sudo systemctl status

graylog-datanode.service

First Login / Preflight

Graylog 7 uses a **preflight login** for the first sign-in. Graylog's web interface docs state that the **initial credentials are written to the Graylog log file after first startup**, and that the **temporary preflight password is different from the normal admin password** configured via `root_password_sha2`.

View the initial preflight password using the following command(on VM1 where the server

is installed): sudo grep -i "Initial configuration" /var/log/graylog-server/server.log

Access the UI

Finish up setting Graylog in the UI by following the steps provided.

You probably must add VM2(datanode) to your /etc/hosts file for the final setup to work.

Enable HTTPS on Graylog (Direct on Graylog, IP-based)

Purpose

This guide enables **HTTPS directly on the Graylog web interface** using a **self-signed certificate** for an **IP address**. Graylog requires an **X.509 PEM certificate**, and a **PKCS#8 PEM private key** for HTTPS.

Assumptions

- Graylog server IP: `172.16.96.119`
- Graylog config file: `/etc/graylog/server/server.conf`
- HTTPS will run on `https://172.16.96.119:9000/`

1. Create OpenSSL config

Create the config file with the following command and paste the code into the file:
nano ~/openssl-graylog-ip.cnf

[req]
distinguished_name =

```
req_distinguished_nam
e x509_extensions =
v3_req
prompt = no


[req_dis
tinguish
ed_nam
e] C =
BE
S
T
=
A
n
t
w
e
r
p
L
=
O
l
e
n
O
=
I
n
t
e
r
n
a
l
O
U
=
G
r
a
y
l
o
g
CN = 172.16.96.119

[v3_req]
keyUsage = nonRepudiation,
digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[alt_names]
IP.1 = 172.16.96.119
```

2. Generate certificate and key

Using the following command:

```
openssl req -x509 -days 365 -nodes -newkey rsa:2048 \
-config ~/openssl-graylog-ip.cnf \
-keyout ~/graylog-web-pkcs5.pem \
-out ~/graylog-web.crt
```

This creates a self-signed **X.509 certificate** and a temporary private key. Graylog requires the final certificate to be **X.509 PEM**.

3. Convert the key to PKCS#8

Using the following command:

```
openssl pkcs8 -in ~/graylog-web-pkcs5.pem -topk8 -nocrypt -out ~/graylog-web-key.pem
```

Graylog requires the private key to be **PKCS#8 PEM**. We also used `-nocrypt`, the key is **not encrypted**, so **`http_tls_key_password` is not needed**.

4. Copy files to Graylog

Using the following commands:

```
sudo cp ~/graylog-web.crt /etc/graylog/server/graylog-web.crt
sudo cp ~/graylog-web-key.pem /etc/graylog/server/graylog-web-key.pem
sudo chown root:graylog /etc/graylog/server/graylog-web.crt
/etc/graylog/server/graylog-web-key.pem sudo chmod 640
/etc/graylog/server/graylog-web.crt /etc/graylog/server/graylog-
web-key.pem
```

5. Configure Graylog HTTPS

Using the following command:

sudo nano

/etc/graylog/server/ser

ver.conf Add or

update:

```
http_enable_tls = true
http_tls_cert_file =
/etc/graylog/server/graylog-web.crt
http_tls_key_file =
/etc/graylog/server/graylog-web-
key.pem http_external_uri =
https://172.16.96.119:9000/
```

> ***Note:*** *If the key is* ***not encrypted****, leave out* `http_tls_key_password`*. If the key is encrypted, set* `http_tls_key_passwo rd` *to the key password.*

6. Make Graylog trust the self-signed HTTPS certificate

Because HTTPS was enabled with a **self-signed certificate**, Graylog's Java runtime did **not automatically trust** the certificate chain. Graylog's HTTPS documentation states that for **self-signed** or **private/custom CA** certificates, the certificate (or CA) must be added to a **JVM truststore**, and Graylog must be configured to use that truststore.

6.1 *Copy the bundled JVM truststore*

Graylog in this setup used the bundled JVM at `/usr/share/graylog-server/jvm/bin/java`, and the default Java truststore was located at `/us r/share/graylog-server/jvm/lib/security/cacerts`. For Java 9+ layouts, `cacerts` is typically stored under `<JAVA_HOME>/lib /security/cacerts`.

Use the following command:

sudo cp /usr/share/graylog-server/jvm/lib/security/cacerts /etc/graylog/server/cacerts.jks

6.2 *Import the Graylog HTTPS certificate into the copied truststore*

Use the bundled `keytool` from the same JVM that Graylog is using. `keytool` is the standard Java

utility for managing keystores and truststores. Use the following command:

```
sudo /usr/share/graylog-server/jvm/bin/keytool -importcert \
-alias graylog-self-signed \
-keystore /etc/graylog/server/cacerts.jks \
-storepass changeit \
-file /etc/graylog/server/graylog-web.crt
```

When prompted, confirm the import by answering yes

6.3 *Configure Graylog to use the custom truststore*

Edit the following file:

/etc/default/graylog-server

Change the following:

GRAYLOG_SERVER_JAVA_OPTS="-Xms4g -Xmx4g -server -XX:+UseG1GC -XX:-OmitStackTraceInFastThrow -Djavax.net.ssl.trustStore=/etc /graylog/server/cacerts.jks -Djavax.net.ssl.trustStorePassword=changeit"

7. Restart Graylog

```
sudo systemctl
restart graylog-
server sudo
systemctl status
graylog-server
```

Graylog service changes take effect after restarting `graylog-server`.

Graylog - Configure Syslog TCP + TLS (Server Authentication Only)

Purpose

This guide secures syslog traffic from a Linux client (Kali / rsyslog) to Graylog using **TCP + TLS** with **server-side authentication only**. This means:

- the connection is **encrypted**,
- the client verifies the **Graylog server certificate**,
- but the client does **not** present its own certificate to Graylog (so this is **not mutual TLS**).

Architecture

- **Graylog server** hosts a **Syslog TCP input** with **TLS enabled**.
- **Kali client** forwards logs with **rsyslog** `omfwd` over **TCP + TLS** using the `gtls` stream driver.

Prerequisites

- Graylog already has a working server certificate and private key (for example the HTTPS certificate created earlier). The Graylog server requires a **public certificate** and **private key** to enable secure communication.
- The client has a copy of the **public certificate** (or CA certificate) needed to verify the Graylog server certificate.
- On Kali, the `rsyslog-gnutls` package is installed so the `gtls` driver is available.

Graylog Server Configuration

1. *Prepare the certificate and key*

Reuse the existing Graylog TLS certificate and private key, for example:

/etc/graylog/server/graylog-web.crt
/etc/graylog/server/graylog-web-key.pe

2. *Create the Syslog TCP input*

In Graylog:

- Go to **System Inputs**.
- Select **Syslog TCP**.
- Configure:
  - **Title:** `Syslog TCP TLS`
  - **Bind address:** `0.0.0.0`
  - **Port:** `6514`
  - **Global:** enabled

3. *Enable TLS on the input*

Use these values:

- **TLS cert file:** `/etc/graylog/server/graylog-web.crt`
- **TLS private key file:** `/etc/graylog/server/graylog-web-key.pem`
- **Enable TLS:** checked
- **TLS key password:** leave blank if the key is unencrypted

- **TLS CLIENT AUTHENTICATION: DISABLED / NOT REQUIRED / OPTIONAL**
  - **TLS Client Auth Trusted Certs:** leave blank

Kali Client Configuration

1. *Install TLS support for rsyslog*

On Kali:

```
sudo apt update
sudo apt install rsyslog-gnutls
```

Debian's package documentation describes **`rsyslog-gnutls`** as the package that adds **TLS protocol support** to rsyslog using **GnuTLS**.

2. *Place the Graylog certificate on Kali*

Copy the Graylog server certificate to:

/etc/rsyslog.d/certs/graylog-web.crt

The rsyslog TLS client must have the CA/trust material needed to validate the server certificate. In this setup, the Graylog public certificate is used as the trust anchor.

3. *Create the rsyslog TLS forwarding config*

Create:

/etc/rsyslog.d/

70-graylog-

tls.conf Use:

```
global(
 DefaultNetstr
 eamDriver="
 gtls"
 DefaultNetstreamDriverCAFile="/etc/rsyslog.d/certs/graylog-web.crt"
)

* *
 actio
 n(
 type
```

```
 ="o
 mfw
 d"
 target="172.16.96.119"
 port="6
 514"
 protoc
 ol="tcp
 "
 Stream
 Driver=
 "gtls"
 Stream
 Driver
 Mode=
 "1"
 StreamDriverAuthMode
 ="x509/certvalid"
 queue.type="linkedList"
)
```

This configuration uses:

- **TCP** transport,
- the **`gtls`** TLS stream driver, which rsyslog defines as TLS-protected transport via GnuTLS,
- **`StreamDriverMode="1"`**, which rsyslog documents as **TLS-protected operation**,
- **`StreamDriverAuthMode="x509/certvalid"`**, which validates the remote certificate chain but does not strictly enforce subject-name matching,
- and **`queue.type="linkedList"`**, which rsyslog recommends for TCP forwarding so local logging does not block if the remote server is temporarily unavailable.

4. *Validate and restart rsyslog*

Validate the configuration:

sudo rsyslogd -N1

Then restart the service:

sudo
systemctl
restart rsyslog
sudo
systemctl
status rsyslog

Testing

1. *Send a test message from Kali*

logger "test syslog over tcp tls from kali"

2. *Search in Graylog*

Search for:

test syslog over tcp tls from kali

If the message appears, the setup is working and the traffic is being transmitted over **encrypted TLS transport**.