UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Dipartimento di Informatica - Scienza e Ingegneria

Corso di Laurea in Ingegneria e Scienze Informatiche

Wazuh, una soluzione di security monitoring per Server Farm

Relatore:
Prof. Vittorio Ghini

Co-Relatore:
Dott. Ciro Barbone

Candidato:
Manieri Davide

Anno accademico 2020-2021

Contents

1 Introduzione

Questa tesi è nata con lo scopo di fornire un nuovo layer di sicurezza alla server farm del Campus di Cesena. Oggi giorno moltissime aziende fanno uso di *Intrusion Detection System* o IDS, software capaci di rilevare anomalie all'interno delle reti, questi però risultano sempre meno efficaci dopo l'avvento del protocollo HTTPS. Quest'ultimo blocca la maggior parte delle informazioni visibili dall'esterno del pacchetto rendendo quindi l'analisi del traffico completamente superflua. Per via di questo è stato deciso di installare un *Host Intrusion Detection System* o HIDS piuttosto che un IDS.
Un HIDS è un software in grado di rilevare anomalie *interne* ad un sistema e inviarle ad un manager capace di gestire i vari log inviati, attivando in determinati casi anche contromisure specifiche. Grazie a questo è possibile ricostruire ciò che è avvenuto internamente ad una macchina colpita da un attacco informatico. L'HIDS in questo caso specifico verrà installato nei server designati sotto forma di agente, ovvero software che agiscono per conto di un manager alla quale invieranno log e messaggi per attuare un sistema di controllo. Questo aiuterà in caso di minacce un ambiente estremamente denso di host (PC, Server e smart object) come quello del Campus a massimizzare l'efficienza delle proprie difese considerando anche la fondamentale importanza che l'infrastruttura interna di rete ha per tutte le attività didattiche e di ricerca.

All'interno dell'ambiente universitario gli attacchi avvengono con frequenza e spesso sono causati inavvertitamente dagli studenti, ignari di essere diventati un mezzo di trasmissione di attività potenzialmente malevole. Tuttavia queste minacce non possono sempre essere trovate con sistemi di sicurezza standard e quindi risulta necessario utilizzare strumenti capaci di gestire tipologie di situazioni simili.

La maggior parte del progetto è incentrata sulla configurazione e il deployment di un *Host Intrusion Detection System* che sarà poi coadiuvato da un sistema per la gestione dei log, derivanti dall'analisi della server farm sulla quale verrà predisposto. Questo aumenterà in maniera consistente l'usabilità rendendo quindi il sistema facilmente utilizzabile.

Un'altra sezione di grande importanza del progetto sarà invece dedicata alla piattaforma di virtualizzazione di *Docker*. Questa scelta è stata effettuata per via della sua capacità di rendere il progetto estendibile e scalabile senza aumentarne la complessità, permettendo anche di poter eseguire modifiche ad hoc in base all'utilizzatore e ai cambiamenti che possono avvenire all'interno dell'infrastruttura.

La tesi sarà sviluppata in dettaglio in 4 capitoli. Nel primo capitolo si descriveranno gli aspetti teorici fondamentali alla base degli *Intrusion Detection System* in modo tale da poter comprendere appieno il contenuto della trattazione. Il secondo capitolo tratterà invece delle tecnologie che sono state sfruttate durante il progetto, analizzando in particolare l'architettura dell'Host Intrusion Detection System scelto e gli aspetti

fondamentali della piattaforma Docker. Nel terzo capitolo verrà spiegato il processo per la configurazione corretta del sistema illustrando anche i vari moduli che vengono utilizzati per la prevenzione. Il capitolo successivo descriverà invece le minacce incontrate durante la fase di testing del progetto e la tipologia di log derivanti dai diversi tipi di server presenti internamente alla server farm. L'ultimo capitolo invece sarà completamente dedicato alle conclusioni e ai possibili sviluppi futuri.

2 Introduzione alla sicurezza

In questo capitolo verrà introdotta la terminologia utilizzata poi nel corso di tutta la tesi, spiegando in maniera generale aspetti fondamentali della sicurezza informatica. Verranno illustrate le principali differenze tra i diversi tipi di software utilizzati come difesa da particolari tipi di minacce che saranno quindi spiegate per dare un contesto più esemplificativo al tutto.

2.1 Cos'è un IDS?

Un *Intrusion Detection System* (o *IDS*) è un applicativo software capace di monitorare network oppure un sistema (host in generale) da attività malevole o violazione di policy. Ogni qualvolta si verificherà un'intrusione o attività sospetta questa verrà notificata ad un sistema centrale utilizzato per immagazzinare informazioni riguardanti la sicurezza della macchina ospitante. Questi sistemi ricevono file di log, alert o messaggi di violazione da una moltitudine di host ma grazie a tecniche di filtering riescono a rilevare attività malevole piuttosto che falsi allarmi.

Gli *Intrusion Detection System* possono essere similari ai Firewall ma hanno differenze sostanziali. Ad esempio un Firewall utilizza dei set di regole prefissasti che andranno a gestire le richieste di connessione al network (rifiutandole o permettendole). Questo limiterà le interazioni tra network evitando quindi intrusioni ma, allo stesso tempo non notificherà alcun attacco in corso. Un *IDS* invece principalmente tenderà a rilevare attacchi interni al sistema ospitante oppure che hanno superato la barriera creata dal firewall (solitamente traffico di tipo http), allertando in maniera tempestiva l'amministratore. Questa operazione viene effettuata solitamente esaminando la comunicazione con
l'esterno, oppure identificando pattern di attacchi comuni effettuati su host attraverso l'uso di euristiche (un esempio può essere Mitre Attack). Non è raro che qualche *IDS* presenti anche funzionalità di risposta alle minacce ed in questo caso si parlerà di *Intrusion* **Prevention** *System* [Bar21].

Gli IDS possono essere di diversi tipi che a loro volta saranno diversificati in base alla tipologia di approccio utilizzato per la scoperta di intrusioni o attività malevole.

2.1.1 Diversi tipi di IDS

Principalmente gli *Intrusion Detection System* sono divisi in due macrocategorie che sono [Bar21] : *HIDS* (o *Host-based Intrusion Detection System*) e *NIDS* (o *Network Intrusion Detection System*).

Un *Host-based Intrusion Detection System* è un particolare tipo di *IDS* capace di monitorare e analizzare internamente un host e anche di rilevare pacchetti sulle interfacce di rete della macchina. In base alle configurazioni assegnatogli un *HIDS* è in grado di rilevare in maniera dinamica lo stato e gli avvenimenti interni al filesystem e alle porte di un host.

Ad esempio un *HIDS* è in grado di verificare se programmi installati sulla macchina hanno accesso a determinate risorse e controllerà lo stato del sistema, sfruttando le informazioni immagazzinate nelle memorie interne e nei file di log generati giornalmente. Lo scopo principale dell'*HIDS* però è quello di scoprire eventuali tracce lasciate da un attaccante (hacker) della macchina ospitante. Solitamente questi ultimi puntano ad ottenere il controllo della macchina installando ed utilizzando applicazioni dedite a aiutare gli attaccanti a perseguire l'obbiettivo da loro prefissato (furti d'identità, furto di password, etc...). Se l'attacco ha avuto successo il sistema verrà appositamente configurato per permettere agli aggressori un ingresso sicuro dedicato appositamente al loro utilizzo.

Gli *HIDS* per eseguire un monitoraggio costante utilizzano comunemente dei database dove vengono salvate informazioni riguardanti tutti gli oggetti all'interno del filesystem. Per ogni elemento all'interno del database l'*HIDS* possiede svariati attributi come ad esempio permessi dell'elemento, dimensione, date di modifica e inoltre creerà un checksum utilizzando funzioni di hash [Waz21](ad esempio MD5 e SHA1) per il contenuto di esso.

Una situazione sfavorevole per un *HIDS* potrebbe essere un sistema con numerosissimi elementi dinamici. Questo tipo di situazione rende difficile riconoscere eventuali attacchi, quindi per superare questa difficoltà gli *HIDS* utilizzano altre tecniche di verifica come ad esempio controllare il cambiamento degli attributi, comparare la dimensione dei file di log mandati dai vari agenti rispetto a quelli mandati precedentemente. Non appena l'*HIDS* rileverà qualche anomalia interna al sistema questo allerterà prontamente gli utenti utilizzando email, inviate direttamente sugli indirizzi indicati, o simili. Esempi di Host Intrusion Detection System possono essere Wazuh, Ossec, etc...

Un *Network Intrusion Detection System* è un software utilizzato per monitorare il traffico in ingresso e in uscita dai dispositivi presenti all'interno della nostra rete. Solitamente per ottenere accesso alla rete si collegano in punti strategici di transito, applicando anche tecniche come il port mirroring su switch. Il loro scopo è quello di anal-

izzare i pacchetti di passaggio e compararli con le firme di attacco da essi conosciute. Quando si verificherà un evento di attacco o non appena un'anomalia viene rilevata, l'*HIDS* invierà un alert agli utenti (o amministratori) che hanno configurato il software. Può venire utilizzato anche per controllare che non ci siano attacchi indirizzati al firewall.

Nel caso di configurazione errata un *NIDS* può creare problemi prestazionali alla rete, ad esempio se viene utilizzato per sniffare tutti i pacchetti, sia in uscita che in entrata, è possibile che generi un bottleneck all'interno della nostra rete causando un calo della velocità generale. Grazie alla loro capacità di poter comparare le firme di attacco è possibile utilizzarli anche per causare drop event su pacchetti potenzialmente pericolosi. Un esempio di *NIDS* possono essere Suricata e Snort.

Gli *IDS* inoltre si differenziano anche in base al metodo di rilevamento delle anomalie [Bar21]:

1. **Signature-based:** Gli *IDS* di tipo Signature-Based rilevano possibili minacce comparandole con pattern ben precisi come ad esempio un'istruzione malevola solitamente utilizzata da malware . Il termine quindi deriva proprio dalla capacità di comparare pacchetti in arrivo con le signatures di attacchi (i pattern stessi). Nonostante tutto questa tipologia di *IDS* comporta dei problemi, anche se capace di identificare ogni tipo di attacco conosciuto ad oggi non riuscirà ad identificare nuove tipologie di minacce per la quale non è ancora esistente una signature.

2. **Anomaly-based:** Questo tipo di metodo è stato progettato per riconoscere ed adattarsi a nuovi tipi di attacco. Ciò è possibile utilizzando il machine learning puntando a creare un modello di sistema affidabile standard. Questo sarà poi comparato con i nuovi comportamenti che verranno rilevati nel corso dell'attività dell'*IDS* decretando se questi sono minacce reali o fittizie. Il punto debole di questa tecnologie è la possibilità di rilevare attività lecite come possibili pattern di attacco.

2.1.2 Cosa sono gli Intrusion Prevention System

Un *Intrusion* **Prevention** *System* (abbreviato *IPS*), è un software dedito a monitorare e rispondere a potenziali minacce provenienti dalla rete. Il suo scopo è quello di rilevare nel traffico network pacchetti potenzialmente pericolosi esattamente come un *IDS*. La differenza però risiede nel fatto che quando viene scoperto un attacco l'*IPS* risponde a questo in modo automatico, basandosi sulle regole configurate dagli amministratori.

Quindi le sue funzionalità sono quelle di rilevare attività malevole, riportarle utilizzando un sistema di logging, bloccare gli eventi ritenuti pericolosi e successivamente segnalare l'avvenuta risposta al problema.

Gli *IPS* solitamente includono firewall, un software anti-virus e a volte anche uno anti-spoofing (furto d'identità) utilizzati per rispondere appunto agli eventi malevoli. Inoltre vengono utilizzati anche per portare a galla eventuali problemi di policy e per incrementare la documentazione a proposito di determinate minacce.

Come nel caso degli *IDS* anche gli IPS utilizzano lo stesso metodo di analisi, controllano il traffico di rete in cerca di attività considerate malevole e di signatures di attacchi, comparando l'oggetto in analisi con gli attacchi di sua conoscenza interni al database. In caso di pacchetti potenzialmente pericolosi l'*IPS* agisce in maniera autonoma eseguendo un evento di drop sul pacchetto, successivamente bloccherà anche l'IP mittente evitando quindi che invii ulteriori attacchi verso il network sotto monitoraggio.

2.2 Che cos'è CVE?

CVE [The21b](Common Vulnerabilities and Exposures) è un progetto che si pone come obiettivo quello di identificare, definire e catalogare pubblicamente vulnerabilità di cybersecurity. Per ogni vulnerabilità esiste un'istanza all'interno del catalogo.Le vulnerabilità sono scoperte e catalogate grazie allo sforzo combinato delle aziende che hanno aderito al programma CVE. Tra le aziende possiamo annoverare anche Apple, Microsoft e Ubuntu, mentre l'aziende manutentrice del programma è Mitre Corporation. In ogni record all'interno del catalogo si può trovare la descrizione della vulnerabilità e altre informazioni riguardanti essa.

CVE è utilizzato ampiamente nel mondo degli IDS e IPS in modo tale da fornire uno standard ben preciso e identificare le vulnerabilità sconosciute. Esse infatti sono identificate attraverso degli id univoci che permettono agli utenti di ricercare la vulnerabilità da essi trovata e trovare informazioni riguardo quest'ultima. Gli ID possono anche non risultare nei database per svariato tempo a causa di problemi di pubblicazione, ad esempio nei sistemi custom o in servizi addizionali di applicazioni le CVE non vengono assegnate.

Le vulnerabilità presenti nel database possono avere anche diversi stati:[The21b]

1. **Reserved:** Una CVE è indicata come Reserved quando viene utilizzata da una CNA o da un gruppo di ricerca specifico, ma non è ancora stata pubblicata sul database. Può uscire da questo stato in qualunque momento.

2. **Disputed:** Le CVE vengono messe in questo stato quando delle CNA non la riconosco come vulnerabilità, in questo caso quindi all'utente viene consigliato di cercare ulteriore documentazione.

3. **Reject:** Ad una CVE viene assegnato lo stato Reject quando non viene riconosciuta come vulnerabilità, la ragione solitamente è insita nella descrizione. Questo potrebbe essere dato dal fatto che sia un duplicato, è stata ritirata da colui che l'ha richiesta oppure è stata mal assegnata.

Oltre al programma CVE esiste anche **CWE** [The21c] (Common Weakness Enumeration) che invece è una lista di debolezze riguardanti software ed hardware. La correlazione tra debolezze e vulnerabilità è che tutte le vulnerabilità derivano dalle prime mentre le debolezze non sempre precludono vulnerabilità. Le debolezze sono quindi bug, errori o imperfezioni insiti all'interno dell'implementazione e della progettazione di software ed hardware e questi potrebbero risultare in vulnerabilità suscettibili ad attacchi. CWE quindi predispone una lista per categorizzare queste debolezze descrivendole ed identificandole. Lo scopo è quello di istruire progettisti e programmatori ad eliminare errori comuni portatori di debolezze, eliminando di fatto il problema delle vulnerabilità alla radice.

2.3 Principali minacce di sistemi

Al giorno d'oggi ogni host può essere attaccato da innumerevoli minacce, presenti in particolar modo sulla rete. Gli IDS puntano proprio a difenderci da questi eventi, tentando di metterci in allerta riguardo alle vulnerabilità sfruttabili per attaccare la nostra macchina. Andremo ora ad introdurre le principali minacce dalla quale un IDS può proteggere un host.

2.3.1 Cosa sono le Botnet?

Una botnet è un raggruppamento di dispositivi connessi alla rete infettati da malware che permettono quindi all'attaccante di prenderne il controllo e di utilizzarlo per i propri scopi. Gli hacker utilizzano questi gruppi di macchine per eseguire attacchi di massa verso host, con lo scopo di attività malevole come furto di identità, accessi non autorizzati, furto di dati.

Queste botnet sono composte anche da migliaia di computer infetti e a comando possono essere utilizzati a piacimento. Comunemente i criminali assumono il controllo della macchina utilizzano speciali virus di tipo Trojan per attaccare i sistemi di sicurezza

del computer obiettivo. Dopo aver eseguito con successo questa operazione installano software di tipo Command and Control per eseguire poi le proprie attività malevole. Queste attività poi verranno eseguite a piacimento e su larga scala, e solitamente includono [Clo21]:

1. **Distributed Denial of Service:** L'attaccante punta ad esaurire tutte le risorse di un sistema informatico fornitore di un servizio, ad esempio un sito web su un web server, fino a farlo collassare e a renderlo inabile ad eseguire una qualunque operazione.

2. **Esposizione di credenziali:** Divulgazioni di credenziali che portano al controllo degli account

3. **Furto di dati:** Gli attacchi vengono eseguiti nei confronti di pagine web con lo scopo di rubare dati.

4. **Accesso non autorizzato:** Vengono forniti ad utenti malevoli accessi a dispositivi e quindi anche connessione alla rete annessa.

In altri casi invece le botnet vengono semplicemente affittate da criminali per eseguire operazioni a proprio tornaconto. Queste botnet quindi possono essere utilizzate da chiunque, anche da chi non ha alcun tipo di conoscenza in questo settore.

Il numero di computer controllati varia da botnet a botnet e questo dipende semplicemente dall'abilità del proprietario ad infettare dispositivi vulnerabili. Ad esempio esistono botnet comprendenti quasi centinaia di migliaia di macchina, un attacco provocato da una mole simile di dispositivi può generare anche più di 350.000 richieste ad un server erogatore di servizi, che possono portarlo a rallentamenti rilevanti oppure a sottrazione di dati personali.

Una delle botnet più conosciute è Mirai [Clo21], che nel suo massimo comprendeva qualche milione di dispositivi. Questo malware attaccava i dispositivi IOT infettandoli e controllandoli a proprio piacimento. Comunemente venivano utilizzati per eseguire attacchi DDoS con i più disparati obiettivi. Uno degli obbiettivi ad esempio è stato un fornitore di servizi DNS attaccato da innumerevoli dispositivi IOT sui quali era installato il malware. Questo attacco ha reso non disponibile moltissimi siti di grande importanza come ad esempio GitHub, Twitter e Reddit per svariate ore.

2.3.2 Cosa sono i Rootkit

I rootkit sono particolari tipi di malware progettati appositamente per garantire accesso e controllo all'attaccante di una macchina bersaglio. Solitamente questi compromettono software o sistema operativo ma non è raro che possano anche danneggiare parti hardware del dispositivo attaccato. I rootkit inoltre sono anche in grado di occultare la loro presenza rimanendo però attivi. Ad esempio possono anche utilizzare keylogger per riuscire a carpire informazioni come password, codici di carte di credito o credenziali di accesso di qualsivoglia genere.

Dopo essersi infiltrato il rootkit da la possibilità all'attaccante di rubare dati personali, installare ulteriori malware o utilizzare la macchina per partecipare ad attacchi DDoS inserendolo anche all'interno di una botnet. L'installazione dei rootkit può avvenire in svariati modi [AO 21]:

1. Solitamente tramite phishing, ingannando la vittima scaricare applicazioni, file media o anche PDF, che però sono stati infettati dal malware in questione.

2. Un'altro metodo è quello di sfruttare debolezze insite nel sistema operativo della macchina o in una applicazione istallata.

Esistono inoltre diversi tipi di rootkit. Questa differenziazione viene effettuata basandosi sul tipo di dispositivi su cui operano [AO 21].

1. **Hardware or firmware rootkit:** Questo tipo di rootkit è in grado di infettare parti hardware della macchina come il disco rigido, un router o anche il BIOS del sistema. Essi hanno la peculiarità di essere molto difficili da individuare. Grazie a questo possono rimanere operativi sulla macchina per lunghi periodi, rubando dati ed informazioni sensibili grazie all'utilzzo di keylogging e grazie al continuo monitoraggio dell'attività di rete.

2. **Bootloader rootkit:** Il bootloader è responsabile del caricamento di un sistema operativo su una macchina. I rootkit possono sfruttare questo per attivarsi durante il boot del PC rendendo possibile la loro esecuzione e ciò rende possibile anche attaccare sistemi con hard disk completamente criptati o intercettare chiavi di cifratura e password.

3. **Memory rootkit:** I memory rootkit si infiltrano all'interno della memoria RAM e sfruttano le risorse del computer per eseguire le loro attività in background. Ciò quindi compromette le performance delle memoria RAM. Questo tipo di

rootkit però non viene considerato come una minaccia significativa, poiché non utilizza codice permanente per infettare la macchina e con un reboot può venire eliminato per via della volatilità della memoria RAM.

4. **Application rootkit:** Questi tipi di rootkit si sostituiscono a file standard presenti all'interno del computer cambiando anche il modo nella quale determinate applicazioni operano. L'attaccante prenderà controllo del device ogni qualvolta l'applicazione manomessa verrà avviata, rendendo difficile quindi l'individuazione di tale rootkit.

5. **Kernel mode rootkit:** Sono il tipo di rootkit più pericoloso in assoluto, poiché attaccando il computer a livello kernel non solo rendono possibile l'accesso della macchina all'attaccante ma possibilitano anche quest'ultimo a manomettere il funzionamento del sistema operativo grazie ad injection di codice malevolo.

6. **Virtual rootkit:** Operano sfruttando i meccanismi di virtualizzazione per ospitare il sistema operativo di destinazione come una macchina virtuale. Questo dà la possibilità di intercettare chiamate hardware fatte dall'OS della macchina attaccata. Inoltre non hanno necessità di modificare il kernel del sistema per rendere efficace l'attacco. Anche se sono molto difficili da individuare possono essere scoperti attraverso i possibili rallentamenti che possono verificarsi durante l'utilizzo della macchina.

In generale rilevare rootkit può essere alquanto complesso, data la loro capacità di disabilitare software di sicurezza o di mascherare la loro presenza. Solitamente però è possibile realizzare di essere stati infettati attraverso qualche evento particolare. Ad esempio è possibile che si verifichino dei "blue screen" dati appunti da una failure generata dal rootikit stesso. Altri segnali di allarme potrebbero essere delle prestazioni al di sotto dello standard della nostra macchina oppure un cambiamento nelle impostazioni del sistema operativo non effettuate dall'utente stesso.

Per rimuovere un rootkit sarà necessario l'utilizzo di software di sicurezza come ad esempio antivirus, che tramite una scansione del sistema potrebbero essere in grado di individuare eventuali rootkit infiltrati. Ciononostante anche questa operazione potrebbe non funzionare e potrebbe essere necessario utilizzare degli integrity check sul sistema oppure analizzare accuratamente la system call table, permettendo di ricercare funzioni che possono essere sfruttate dal rootkit per eseguire operazioni malevole.

2.4 Metodi di difesa

2.4.1 Hardening

Con la parola *hardening* si intende un insieme di pratiche specifiche per configurare nel miglior modo possibile un sistema informatico, puntando a ridurre al minimo le entità dei possibili attacchi informatici sfruttanti le vulnerabilità del sistema stesso.

Le classiche pratiche per effettuare hardening consistono principalmente nella riduzione delle superfici di attacco. Ad esempio è buona norma chiudere porte inutilizzate, arrestare servizi in non utilizzati, disabilitare privilegi ad utenti, eliminare account di utenti o di amministratori, etc.

Per effettuare queste operazioni però bisogna procedere a step, in primis partendo da un'analisi sullo stato della macchina e sulle possibili componenti vulnerabili, successivamente si potrà passare alla fase vera e propria di intervento. In questa fase è necessario redigere un elenco di tutte le operazioni effettuate sulla macchina in modo da poter permettere, in caso di ripristino di rieffettuare tutte le pratiche eseguite. Attualmente esistono anche script o tool capaci di eseguire hardening sulle macchine ospitanti, uno di questi ad esempio è SElinux [Red21].

2.4.2 Informatica forense

L'informatica forense [Nex21] è quella branca del settore informatico che mira a raccogliere informazioni utilizzate poi nell'ambito di processi giudiziari. Il focus è incentrato sullo studio dei dati digitali, in particolar modo sul trattamento e sul mutamento durante il loro ciclo di vita.

Le metodologie e i software utilizzati per svolgere queste mansioni devono rispettare normative e procedure molto stringenti per via della delicatezza dei dati trattati (nella maggior parte dei casi sono appunto dati sensibili). Al giorno d'oggi questo settore ha visto un grande sviluppo anche grazie alla moltitudine di device presenti nella vita di tutti i giorni, contenenti informazioni e dati possibilmente di grande importanza in determinati casi.

Tutte le attività devono essere svolte seguendo determinate fasi con accortezza poiché bisogna garantire l'integrità dei dati trattati che non dovranno essere poi eliminati. Le operazioni eseguite riguardano principalmente il recupero di file eliminati, ritrovamento di file esportati, ricerche su caselle di posta, integrity check di server, file non autorizzati salvati in memoria e scoprire eventuali utilizzi impropri di applicazioni.

I dati possono essere rovinati anche solo con un'errata accensione di un dispositivo, per evitare ciò quindi sono state istituite diverse fasi in modo da prestare attenzione ad ogni azione eseguita [Nex21].

1. **Identificazione:** In questa fase viene prestata attenzione ai dispositivi da dove devono essere acquisiti i dati alla quale si è interessati. Per via dell'immenso numero di questi, capaci di immagazzinare informazioni è quindi necessario selezionare attentamente quali possono contenere dati sensibili. Ad esempio gli elementi del mondo IOT possono essere molteplici.

2. **Acquisizione:** Nella fase di acquisizione ci si concentra sul come estrapolare dati dai dispositivi ed esistono due modalità a seconda se il dispositivo è acceso o spento. Se il dispositivo è funzionante bisogna accertarsi di fare una raccolta dati completa partendo dai processi in esecuzione, account presenti e connessioni effettuate. Inoltre è buona prassi utilizzare funzioni di hash per garantire che i dispositivi utilizzati siano effettivamente originali. In caso invece di dispositivi non funzionanti bisognerà ricercare la modalità di estrazione dati o di prelevazione di memorie o componenti hardware.

3. **Analisi:** Durante questa fase verranno analizzati i vari dati recuperati precedentemente dai dispositivi. Ciò verrà eseguito utilizzando software, analizzando keywords o individuando tracce digitali di manomissioni.

4. **Report:** Successivamente all'analisi verrà redatto un report che potrà poi essere consultato ogni qualvolta si desidera.

3 Tecnologie utilizzate

In questo capitolo tratteremo le tecnologie che sono state utilizzate all'interno del progetto in modo da comprendere poi in maniera chiara i vari passaggi eseguiti per la realizzazione. Partiremo con introdurre Docker che e' stato utilizzato come piattaforma di deploying per l'HIDS, dopodiché introdurremo Wazuh, il core centrale del nostro progetto e infine parleremo di Suricata utilizzato principalmente come test per sperimentare la capacita' di modularizzazione di Wazuh.

3.1 Docker

3.1.1 Cos'è Docker?

Docker [Doc21] è un progetto open-source per eseguire applicazioni all'interno di un *container*. Esso è oggigiorno il developer tool più utilizzato e famoso, a tal punto da essere diventato un sinonimo per la parola *container*. Con questa parola si intende un tipo di virtualizzazione a livello di sistema operativo. Un *container* quindi è inteso come una partizione di sistema operativo nella quale è possibile eseguire operazioni in un ambiente isolato.

E' possibile installarlo su un server o un host qualsiasi e comprende già svariati tool che rendono semplice e veloce il suo utilizzo. Docker sfrutta meccanismi propri del kernel Linux come *cgroups* e *namespaces* [Doc21]. La prima versione si basava su LXC [Lin21], un ambiente di virtualizzazione a *container*, che opera a livello del sistema operativo e permette di eseguire diversi ambienti Linux virtuali isolati tra loro. Mano a mano Docker si è sempre più scostato da questo andando a rivoluzionarsi nell'architettura e anche nella finalità di utilizzo andando quindi a contemplare non solo la semplice esecuzione di un applicativo, ma anche la distribuzione di un software e in generale la gestione del ciclo di vita del *container*.

3.1.2 Architettura Docker

Docker utilizza un'architettura di tipo *client-server*, il client comunicherà con il docker deamon che andrà a runnare i vari *container*. Client e deamon comunicano attraverso l'utilizzo di socket o di REST API (elemento di intermediazione tra gli utenti o i clienti e le risorse che questi intendono ottenere).

Introduciamo quindi le varie componenti sovracitate in maniera più specifica, in modo da comprendere poi appieno il funzionamento dell'architettura Docker [Doc21].

1. **docker deamon:** un processo persistente che gestisce immagini, *container*, rete e volumi di storage. Questa componente è sempre in attesa di richieste da parte della REST API e le esegue non appena queste si presentano.

2. **REST API:** Le API sono utilizzate dalle varie applicazioni per utilizzare nella maniera opportuna il docker deamon.

3. **docker CLI:** un'interfaccia a linea di comando lato client utilizzata per sfruttare il demone di Docker. Semplifica in maniera consistente le varie interazioni con le istanze dei *container* e quindi estremamente user-friendly.

A questo punto possiamo illustrare le varie componentistiche presenti all'interno dell'architettura docker ed esse sono [Doc21]:

1. **Docker client:** gli utenti utilizzanti docker interagiranno con l'architettura attraverso il client. Ogni volta che un comando docker viene immesso questo viene inviato al docker deamon, che procede a trasferirli alle varie API. Docker rende anche possibile la comunicazione con più demoni.

2. **Docker host:** L'host fornisce un ambiente di esecuzione per le varie applicazione. Solitamente comprende il deamon, le immagini, i *container*, le reti e uno spazio di archiviazione. Il demone inoltre riceve i vari comandi dal CLI (interfaccia a linea di comando) oppure tramite le API.

3. **Docker image:** Le immagini sono file binary in sola lettura che possono essere utilizzati per costruire poi i vari *container* dove vengono eseguite le varie applicazioni. Le immagini sono utilizzare per distribuire le applicazioni possono anche essere modificate a piacimento per estendere le configurazioni già esistenti. Le varie immagini inoltre possono essere condivise con altri utenti attraverso registri pubblici come *Docker Hub*, e grazie a questa feature è facilmente notabile come le immagini siano uno dei core fondamentali dell'esperienza Docker.

4. **Docker container:** I *container* sono ambienti isolati dove le applicazioni ven-

gono eseguite, sono costruiti tramite le immagini e possono avere configurazioni aggiuntive come ad esempio connessioni network o opzioni di storage.

5. **Docker networks:** I network sono le vie attraverso la quale i vari *container* possono comunicare e principalmente sono di 5 tipi:

   (a) **Bridge:** Connessione di default per *container*, viene utilizzata nel caso standard cioè quando abbiamo più *container* comunicanti con lo stesso host.

   (b) **Host:** Connessione che rimuove l'isolamento network tra il *container* e il docker host (andando ad utilizzare quindi la stessa connessione dell'host) e rimuovendo quindi l'ip del *container* stesso. Può essere utile per incrementare le performance o in casi ove il *container* deve gestire un grande numero di porte. Questa funzionalità è utilizzabile solo su host Linux.

   (c) **Overlay:** Connessione usata quando si hanno più *container* in esecuzione su diversi hosts (quindi con diversi docker deamon).

   (d) **Macvlan:** Le connessione Macvlan permettono di assegnare MAC address ad un *container*, facendolo quindi apparire come un device fisico sulla rete. Il docker deamon quindi reindirizza il traffico in base al MAC address dei *container*. Viene utilizzato quando si ha la necessitare di trasportare Vlan ad un container.

   (e) **None:** Disabilita qualunque tipo di networking e questo è utilizzato di frequente quando si vuole utilizzare un driver custom. I driver custom possono essere scaricati attraverso Docker Hub o da terze parti.

6. **storage:** E' possibile memorizzare dati all'interno dei *container* ma per fare ciò è richiesto uno storage driver. Solitamente questi driver non sono persistenti, cioè quando il *container* relativo smette di essere eseguito i vari dati memorizzati verranno persi. Docker quindi ci fornisce diverse opzioni per fare in modo di non perdere alcuna informazione. Tra le principali abbiamo [Doc21]:

   (a) **Data volumes:** Hanno la capacità di creare uno storage persistente che può essere rinominato e interrogato su quali *container* lo stiano utilizzando. Essendo localizzato al di fuori del *container* è possibile farlo utilizzare a più applicazioni contemporaneamente.

   (b) **Volume container:** E' possibile creare anche un *container* completamente dedicato all'attività di storage.

   (c) **Directory mounts:** Permette di utilizzare una qualsiasi directory appartenente all'host (ed esterna al volume mount del docker) come uno spazio di archiviazione per il *container*.

   (d) **Storage plugins:** Permette di connettersi a spazi di archiviazione esterni. Su Docker Hub è possibile avere una lista dei plugin utilizzabili.

7. **docker registry:** Sono servizi che permettono di memorizzare immagini e scaricarle a piacimento. I Docker Registry quindi contengono delle repositories contenenti quest'ultime. Uno di questi è Docker Hub dove si possono trovare le più disparate applicazioni. I comandi per utilizzare i registri sono principalmente tre e sono *docker push*, *docker pull* e *docker run*. Le immagini scaricate tramite Docker Hub sono subito deployabili e si riferiscono sempre all'ultima versione dell'applicazione relativa.

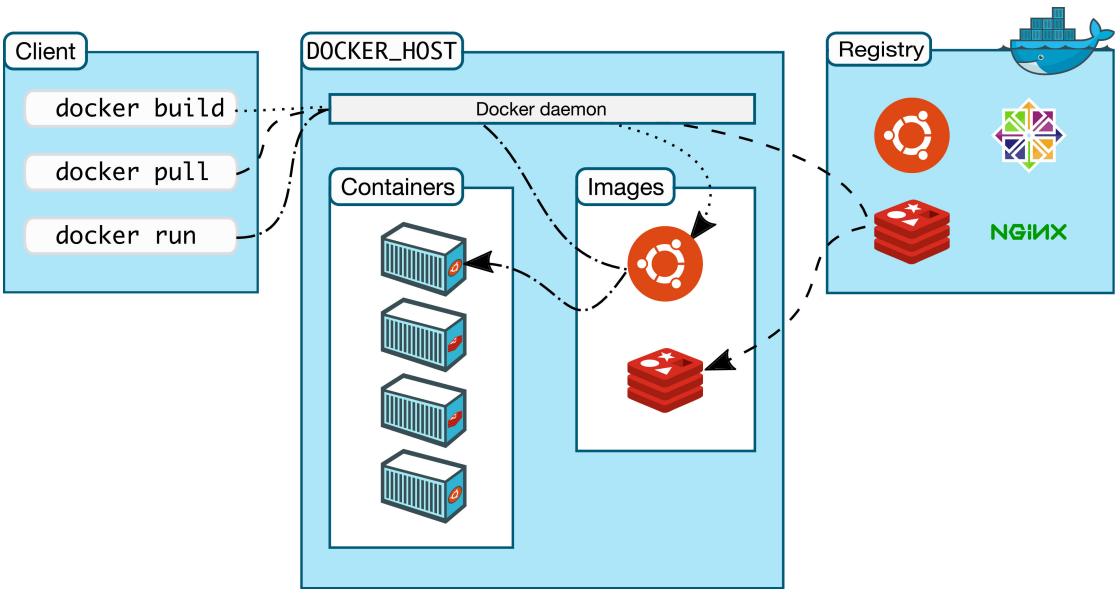

Figure 1: Architettura tipica di Docker.

Attraverso il comando *docker pull* si seleziona la versione dell'immagine che vogliamo deployare, dopo aver fatto ciò si andrà a creare un *container* contenente il dockerfile appena scaricato sfruttando *docker build*, infine, utilizzando il comando *docker run* si metterà in esecuzione il tutto.

3.1.3 Alternative a Docker

Oggigiorno una delle proprietà più importanti di un sistema è la *scalabilità*, cioè la capacità di un sistema di aumentare o diminuire di scala in funzione delle necessità e disponibilità. Docker però non rende possibile questo poiché attraverso il CLI è solo possibile amministrare un *container* alla volta. A questo punto entra in gioco *Docker Swarm* [Doc21], la soluzione per clustering sviluppata da Docker stesso. Swarm ci permette di amministrare un cluster di server sui quali sono stati deployati dei *container* Docker. Non è quindi un software di *container*izzazione ma un software di orchestrazione *container*. Docker Swarm però sta venendo piano piano soppiantato da un altro tool di questo tipo, Kubernetes [The21a].

Figure 2: Architettura di un cluster Kubernetes.

Kubernetes è un *container* orchestration tool sviluppato da Google e si pone come obiettivo quello di gestire correttamente le risorse di un cluster più o meno grande di server. L'architettura è composta da un Control Plane Node (o anche Master), uno o più nodi e dai pods.

Il Control Plane Node è il nodo centrale del cluster in quanto è lui che orchestra tutti quanti gli altri nodi organizzando l'esecuzione dei running *container* su di essi. Il Master può essere anche replicato su più nodi in modo da garantire un servizio affidabile. Tra le sue componenti più importanti abbiamo [The21a]:

1. **kube-controller-manager:** Control-loop che si occupa di leggere lo stato del cluster attraverso l'API Server ponendosi come obiettivo quello di fare in modo che lo stato attuale del sistema sia il più vicino possibile a quello desiderato.

2. **kube-scheduler:** Decide come assegnare i vari carichi sui nodi che compongono il cluster, solitamente la scelta viene effettuate in base alle risorse dei singoli nodi.

3. **Etcd:** E' un storage di tipo key-value che si prepone di mantenere e amministrare le informazioni critiche che Kubernetes necessita per funzionare, come ad esempio file di configurazione, file di stato e metadati.

Un'altra componente fondamentale sono i *nodi*. Si occupano di eseguire i carichi demandati dal master secondo modalità definite sempre da quest'ultimo. Anche i nodi hanno al loro interno componenti molto importanti. Il *kubelet* ad esempio appartiene al master ed è dedito a controllare che i vari *container* all'interno dei pod siano in esecuzione e non richiedenti risorse. All'interno di un nodo è presente anche il kube-proxy che permette ai vari nodi di creare un newtork e di eseguire anche semplici operazioni TCP, UPD. Il carico di lavoro sui nodi è eseguito dal *container* runtime controllato dal kubelet. Docker è il *container* runtime più utilizzato compatibile con Kubernetes.

I pod invece sono semplicemente l'unità elementare dell'architettura Kubernetes. Qui sono contenuti i *container* in esecuzione sul nodo, le risorse infatti sono condivise fra questi ultimi. I pod sono fortemente scalabili, poiché possono essere facilmente replicati e spostati su altri nodi della rete Kubernetes [The21a].

3.2 Elastic Stack

Elastic Stack [Ela21] è un insieme di tool open-source utilizzate per indicizzare dati, immagazzinare log, e visualizzare in maniera chiara e veloce tutti i documenti salvati all'interno dei vari componenti. Andremo quindi ora ad introdurre tutte le componenti principali di ELK (Elasticsearch, Logstash, Kibana).

3.2.1 ElasticSearch

Elasticsearch [Ela21] è un motore di ricerca e analisi dati basato su Apache Lucene. Permette di immagazzinare strutture dati complesse come file JSON, che poi possono essere ricercati e analizzati in tempo reale dando risposte tempi brevissimi. Questo è reso possibile grazie alla sua capacità di non ricercare un dato direttamente ma un indice.

Un documento salvato su Elasticsearch può essere anche qualcosa di diverso da un testo, potrebbe anche essere ad esempio una struttura dati complessa in JSON quindi numeri, stringhe o date di calendario. Ognuno di questi documenti avrà un proprio id che identificherà la tipologia di file salvato.

I vari documenti sono poi organizzati grazie agli *index*. Questi sono collezioni di documenti simili e quindi logicamente collegati tra loro. Sono solitamente identificati da un nome che viene utilizzato poi nella varie operazioni che si possono effettuare, come ad esempio ricerche, aggiornamenti e cancellazioni.

Il core della tecnologia Elasticsearch sono gli *inverted index*. Questo tipo di struttura permette di salvare un mapping direttamente dal contenuto consistente di parole (o numeri) e della loro locazione all'interno di un documento e di una serie di essi. Questo permette di reindirizzare direttamente un utente alla locazione esatta della parola ricercata. Il meccanismo consiste nel separare un documento in parole mappando ogni singolo termine al documento nella quale esso occorre.

Oltre a ciò Elastisearch è in grado anche di creare dei cluster cioè degli insiemi di nodi. Ognuno di questi nodi esegue un'istanza di Elasticsearch che potrà quindi partecipare in maniera cooperativa all'indicizzazione dei dati e alla ricerca delle informazioni immagazzinate all'interno di tutti i nodi che compongono il cluster. Un nodo può avere diversi ruoli ben precisi [Ela21]:

1. **master-elegible-node:** Nodo che può essere promosso a master e quindi effettuare indexing e ricercare i nodi all'interno del cluster. Bisogna assicurarsi che questi nodi abbiano una potenza computazionale sufficiente in modo che il cluster funzioni in maniera corretta. Decidono anche quali shard allocare ai vari data node. Solitamente è il tipo di nodo default.

2. **data-node:** Nodo che ha il ruolo di eseguire operazioni su dati come letture, scritture, rimozioni, update, ricerca ed aggregazioni sulle shard al loro interno. I dati vengono poi salvati sulla memoria fisica del nodo stesso. Dovendo effettuare molte operazioni I/O, gravando quindi su memoria e CPU, bisogna assegnare a questi nodi risorse congrue al loro carico di lavoro.

3. **ingest-node:** Nodo che ha come obiettivo quello di eseguire operazioni di arricchimento e trasformazioni sui vari documenti prima che quest'ultimi vengano indicizzati. Quando si ha un carico elevato di tipo ingest conviene avere nodi predisposti solo a questo tipo di utilizzo, evitando di appesantire nodi master o data.

4. **remote-elegible-node:** Nodo che può essere utilizzato anche da più cluster come ponte di unione.

5. **machine-learning-node:** Nodi che sfruttano la tecnologia del machine-learning per effettuare analisi dei dati. Implementato nativamente all'interno di Elasticsearch.

6. **transform-node:** Nodo utilizzato per effettuare modifiche all'index relatavo ai documenti.

Elasticsearch inoltre consente di dividere l'index in multiple parti chiamate *shards*. Ogni shard contiene un indice indipendente e completamente funzionante che può essere contenuto su qualunque nodo. Distribuendo quindi i documenti tra gli shard e questi ultimi tra multipli nodi Elasticsearch fornisce ridondanza, che protegge sia dalla rotture hardware, e anche più prestazioni nell'eseguire le query, visto che possono essere spartite tra tutti i nodi.

Un'altra feature di Elasticsearch è quella di poter controllare la tipologia di autenticazione effettuata dagli utenti. Tra le tante abbiamo LDAP, SAML e Active Directory. E' possibile anche crearsi un proprio realm di autenticazione fatto su misura per le proprie esigenze. Questo servizio però è attivabile solo dopo previo pagamento di un abbonamento annuale.

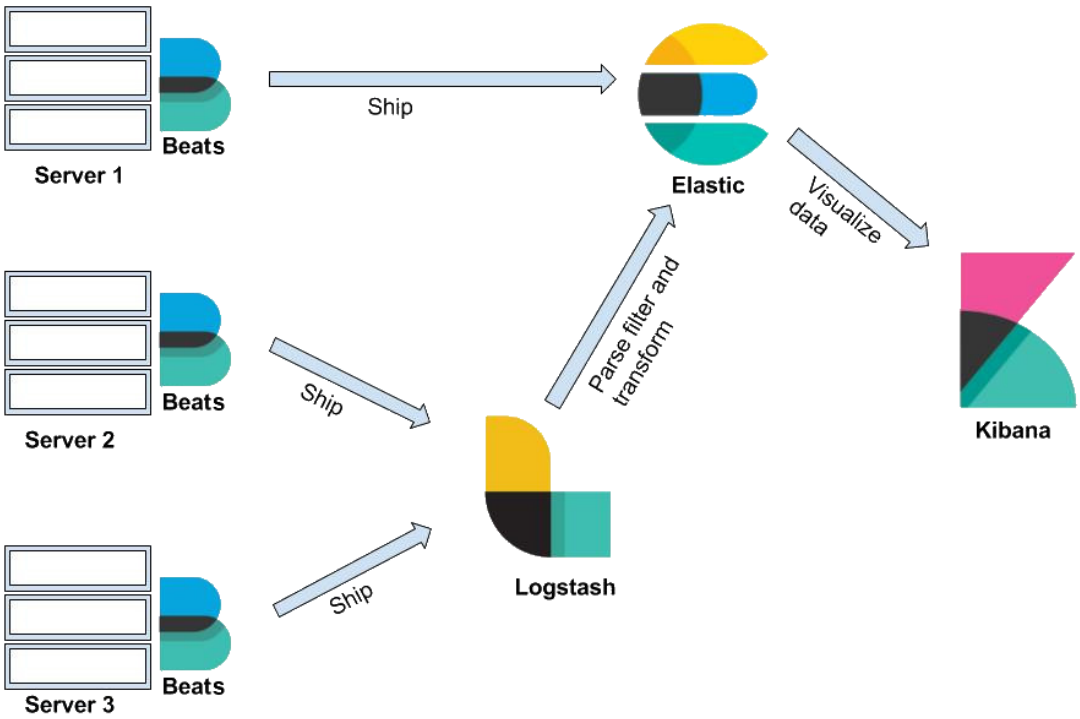

Figure 3: Elastic Stack.

3.2.2 Kibana

Kibana [Ela21] è il tool di Elastic Stack in grado di visualizzare e gestire i dati inviati alla piattaforma in *real-time*. Permette anche di interrogare il cluster Elasticsearch e trasformare le informazioni sotto forma di grafici di vario tipo e anche utilizzando tabelle, rendendo molto più semplice la loro lettura. Inoltre grazie a questo tipo di servizio è un ottimo complementare ad un sistema di monitoring, come nel nostro caso, filtrando eventi, generando report e anche creando meccanismi di notifica personalizzati aiuta a comprendere in maniera chiara quello che sta avvenendo all'interno dei nostri host. E' possibile anche creare *dashboard* per la visualizzazione dei dati custom, dando la possibilità di visualizzare ciò che si considera prioritario.

3.2.3 Logstash

Logstash [Ela21] è un software open-source e server-side capace di prendere in input file da diversi host, modificarli e successivamente inviarli, nel nostro caso, al master

Elasticsearch. I file ingeriti da Logstash possono essere anche molto complessi e di diverso tipo, ma saranno tutti convertiti in maniera tale da poter essere letti o da avere informazioni aggiuntive, ad esempio è possibile visualizzare la geolocalizzazione dell'indirizzo address dalla quale il documento giunge.

3.2.4 Filebeat

Filebeat [Ela21] è un software light-weight e open-source utilizzato per l'invio di file in rete destinati al cluster Elasticsearch. Quest'ultimo poi immagazzinerà e indicizzerà i vari log inviati. Nel nostro caso l'utilizzo di Filebeat è stato necessario per poter inviare i log recuperati dall'analisi effettuata da Wazuh al master node di Elasticsearch. Solitamente per poter trasmettere i file è necessario specificare un percorso di sistema dalla quale filebeat potrà recuperare i log files creati dai vari IDS.

3.3 Wazuh

Wazuh [Waz21] è un *HIDS* (Host Intrusion Detection System) che permette quindi di avere un monitoraggio delle intrusioni all'interno dei sistemi, monitoraggio dell'integrità file, e risposta alle varie minacce presenti e future. Esso è un fork del progetto OSSEC e punta a migliorare la facilità di utilizzo così come la scalabilità, visto che è possibile anche deployarlo attraverso una piattaforma Kubernetes. Wazuh inoltre per essere eseguito nella maniera corretta ha bisogno di 3 componenti fondamentali per essere in grado di fornire tutti i servizi desiderati.

3.3.1 Architettura di Wazuh

L'architettura Wazuh [Waz21] si basa principalmente sull'utilizzo di agenti, che sono in continua esecuzione su degli host remoti mandando continuamente dati relativi alla sicurezza del sistema al server centrale (Wazuh Master). Inoltre è possibile avere anche degli endpoint senza agente o agentless (switch, router, etc..) essendo supportati nativamente. Questi sono abilitati a mandare messaggi di log tramite protocollo Syslog, SSH o utilizzando una API propria. Il server centrale ha il compito di analizzare e decodificare i dati in ingresso e successivamente di inviarli al master Elasticsearch dove saranno poi salvati utilizzando la tecnica dell'indexing. Wazuh utilizza principalmente 3 tipi di index che sono [Waz21]:

1. **wazuh-alerts:** Indice utilizzato per gli alert generati dal Wazuh Server. Questi ultimi vengono creati ogni qualvolta un evento infrange una regola con una priorità abbastanza alta.

2. **wazuh-events:** Indice per tutti gli eventi che vengono inviati dagli agenti sia se hanno violato una regola sia se non l'hanno fatto.

3. **wazuh-monitoring:** Indice utilizzato per dati relativi allo stato durante il tempo. viene utilizzato dalla dashboard per rappresentare quando un agente è stato "Attivo", "Disconnesso", "Mai Connesso".

Se si vuole utilizzare un cluster di piccole dimensioni è necessario un singolo nodo master, poiché le informazioni in ingresso possono ancora essere gestite da un singolo nodo. Se invece il cluster è di dimensioni importanti bisogna considerare di creare più master in modo da poter contenere grandi flussi di informazioni. Svolgendo un compito così importante è necessario avere il master sempre in funzione per evitare perdite di dati importanti.

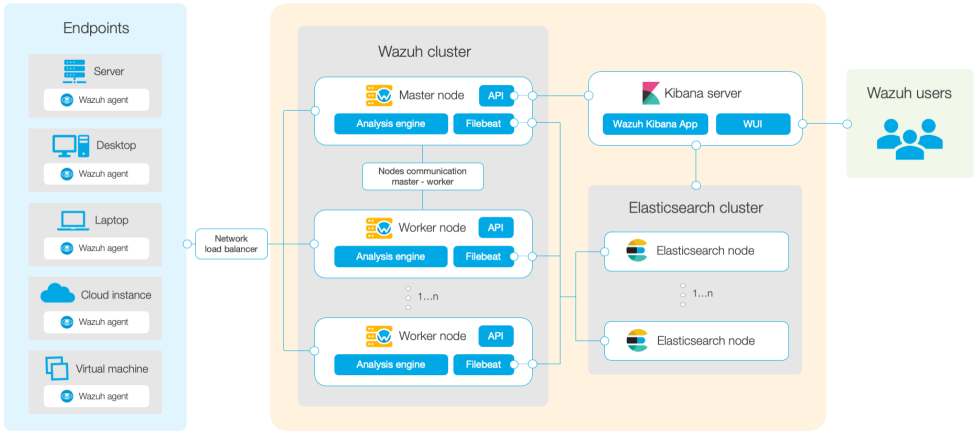

Figure 4: Cluster Wazuh.

Sarebbe bene considerare di separare il server Wazuh con il server Elasticsearch in modo di avere la possibilità, tramite Filebeat, di poter inviare i vari alert generati Wazuh oppure immagazzinarli su Elasticsearch utilizzando la crittografia TLS [Waz21].

Andiamo ora a parlare più nel dettaglio dei due componenti fondamentali di Wazuh (*Wazuh Agents*, *Wazuh Servers*) e delle loro funzionalità più interessanti.

3.3.2 Wazuh Agents

Gli agenti Wazuh [Waz21] possono essere eseguiti su qualunque tipo di sistema operativo (Windows, macOS, Linux, Debian, Solaris etc..). Un agente può essere installato su una qualunque piattaforma come ad esempio server, macchine virtuali, container o

servizi cloud. Il loro scopo è quello di fornire un servizio di prevenzione dalle minacce, individuazione di eventuali intrusioni e risposta ad eventuali attività malevole. Inoltre sono utilizzati anche per inviare file di logging o informazioni di sistema al Wazuh Server attraverso un canale criptato e solo dopo previa registrazione.

Gli agenti hanno un'architettura di tipo modulare e quindi è possibile scegliere quale tipo di componenti usare e quali invece disabilitare modificando semplicemente le configurazione relative. Tra le capacità dell'agente abbiamo quella di monitorare i file di sistema, leggere messaggi di log e ricercare eventuali malware intrusi all'interno dell'host.

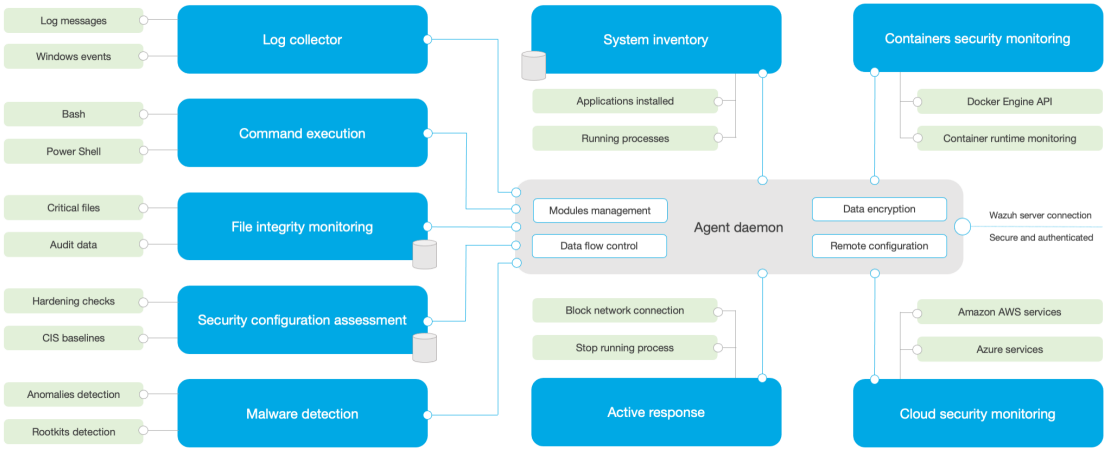

Figure 5: Architettura di un agent.

Andiamo quindi ad elencare il compito di ogni modulo all'interno dell'agent [Waz21]:

1. **Log collector:** Componente che permette la lettura di log file Linux e di eventi Windows collezionando inoltre i vari messaggi che vengono inviati da possibili applicazioni. Supporta XPath (che permette di individuare i nodi all'interno di documenti XML) per gli eventi Windows ed è capace di identificare formati multi linea (come ad esempio Linux Audit). Può anche arricchire eventi Json con metadati aggiuntivi. Il suo scopo è quello di identificare possibili errori di applicazioni o di sistema, errori di configurazione, tentativi di intrusione o problemi di sicurezza.

2. **Command execution:** Gli agenti sono possibilitati a eseguire determinati comandi autorizzati, utilizzando poi l'input del suddetto e inviarlo al server per effettuare ulteriori analisi. Questa funzionalità può risultare utile per controllare ad esempio lo spazio vuoto su hard disk, per controllare quali utenti hanno eseguito un accesso, riavviare determinati servizi, eseguire script più o meno complessi o far eseguire deamon indefinitamente.

3. **File integrity monitoring (FIM):** Modulo che permette di monitorare il file system mandando avvisi ogni qualvolta si verifica un qualsiasi evento (creazione del file, eliminazione o modifica) e aggiungendo anche chi lo causa, cosa lo causa e quando avviene. Tiene traccia di tutti gli attributi relativi al file come ad esempio permessi, proprietà e contenuto. Inoltre è anche possibile salvare lo stato di tutti i file controllati all'interno di un database, interrogabile tramite query anche da remoto.

4. **Security configuration assessment (SCA):** Modulo che esegue una valutazione continua delle configurazioni presenti, utilizzando test basati sui benchmark del Center of Internet Security (CIS). Si possono creare degli SCA test ad hoc per rinforzare le policy di sicurezza già previste. Inoltre fornisce la capacità di interpretare e di controllare configurazioni anche in formato YAML.

5. **System inventory:** Componente che permette di recuperare informazioni che riguardano hardware e software degli host controllati. I dati ottenuti possono essere interrogati utilizzando le API di Wazuh, questi includono consumo della memoria, spazio d'archiviazione, specifiche CPU, interfacce network, porte aperte, processi in esecuzione e lista delle applicazioni installate. Per collezionare queste informazioni Wazuh esegue uno scanning dell'host, sulla quale è installato un agent, in un intervallo di tempo modificabile tramite configurazione.

Quando lo scan viene completato, vengono comparati i nuovi dati trovati con quelli precedentemente collezionati. In questo modo è possibile identificare eventuali differenze che possono essere ad esempio nuove porte aperte, eventi di sistema, una nuova applicazione installata.

6. **Malware detection:** Componente capace di scoprire se anomalie o possibili root-kit si sono infiltrati nel sistema. Analizzando le system call, controlla i vari processi nascosti, file e porte in modo da capire se effettivamente ci siano attività malevole. I malware possono utilizzare svariate modalità di attacco quindi questo modulo è capace a sua volta di sfruttare molteplici approcci per scoprire i vari pattern di entità malevole.

7. **Active response:** Modulo che automatizza le reazioni verso eventi malevoli rilevati. Ad esempio può fermare processi, eliminare file sospetti, bloccare un tentativo di connessione. Possono essere create anche delle risposte custom che possono variare da un blocco di traffico ad eseguire una scansione tramite un antivirus.

8. **Containers security monitoring:** Componente incluso già all'interno delle Docker API e che permette di controllare tutti i cambiamenti che possono avvenire all'interno di un ambiente containerizzato. Può rilevare cambiamenti alle immagini container, alle configurazioni di rete, ai volumi utilizzati come storage dia container. Inoltre informa anche su quali container utilizzando privilegi di amministrazione o se qualche utente esegue comandi in container in esecuzione. Questo modulo non funziona solo con Docker, ma è anche compatibile con Kubernetes. Con quest'ultimo è possibile anche avere un monitoraggio a livello di container installando l'agente direttamente dentro il DeamonSet Kubernetes permettendo anche di inviare tutte le informazioni provenienti da tutto il cluster direttamente al Wazuh Server.

9. **Cloud security monitoring:** Modulo che permette di monitorare le attività di un servizio cloud come Amazon AWS, Microsoft Azure, Google GCP. Comunicando con le loro API è in grado di rilevare cambiamenti all'interno della loro infrastruttura. I servizi di monitoraggio si differenziano da cloud a cloud, Amazon AWS ad esempio è quello più supportato, con servizi che vanno da monitorare continuamente l'infrastruttura in modo da bloccare qualunque attività malevola al valutare le configurazioni delle risorse assegnate al cloud e automatizza il processo per effettuare questa operazione.

L'agente comunica con il server in modo da far immagazzinare le informazioni collezionate e gli eventi di sicurezza. Inoltre manda anche dati riguardo configurazione e stati attuali. Effettuata la connessione l'agente è upgradabile, monitorabile e configurabile direttamente dal Wazuh Server o utilizzando anche la Dashboard Kibana.

Le configurazioni degli agent possono essere anche centralizzate, cioè è possibile gestirle tutte direttamente dalla dashboard. Questo permette di avere le stesse funzionalità per ogni agent senza dover modificare nulla uno per uno. E' possibile anche dividerli in gruppi per differenziare le varie configurazioni.

Le comunicazioni con il server avvengono utilizzando canali sicuri (TCP e UDP) che sfruttano tecniche di criptazione e compressione. In questo modo è anche possibile utilizzare un controllo del flow del traffico gestendo quindi eventuali problemi di flooding, accodamento degli eventi e congestioni causate da eventuali problemi di banda.

L'accesso dell'agente però può essere effettuato solo dopo previa registrazione al server. Questo processo fornisce all'agente una chiave unica che sarà poi utilizzata per autenticazione e per criptare le informazioni. Questo procedimento può essere facilmente svolto attraverso la Dashboard Kibana.

3.3.3 Wazuh Server

Il server Wazuh [Waz21] ha il compito di analizzare le informazioni in ingresso dagli agenti, generando poi notifiche ogni qualvolta una minaccia o un'anomalia viene rilevata. Inoltre viene sfruttato anche per gestire le configurazioni degli agenti e controllare che il tutto funzioni nella maniera corretta.

Grazie all'uso della Cyber Security Intelligence il server Wazuh può migliorare la sua capacità di rilevamento anomalie. Sono informazioni raccolte durante attacchi ad aziende o enti pubblici e permettono di prevedere cosa può succedere in una determinata situazione, creando quindi un ambiente preventivo piuttosto che reattivo. I dati sono inoltre arricchiti utilizzando un framework chiamato *MITRE ATT&CK* [Waz21].

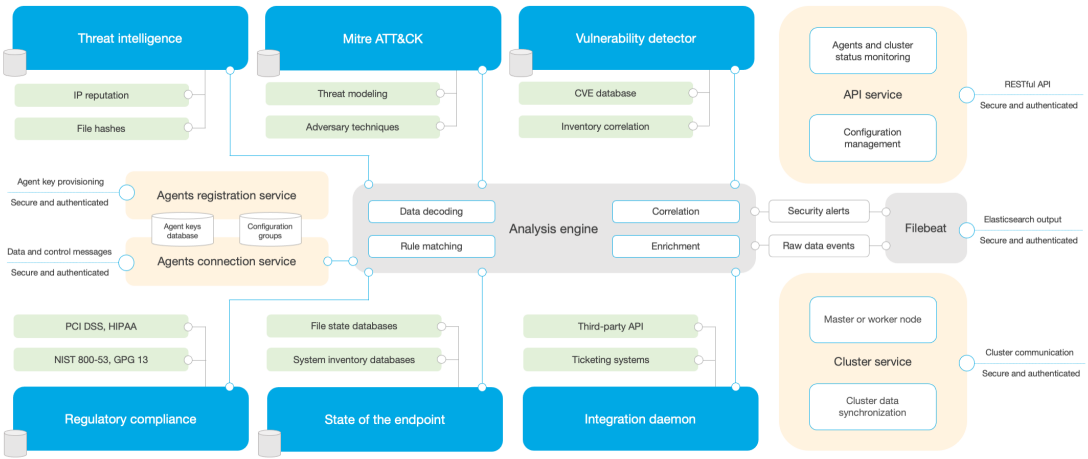

Figure 6: Architettura di un Server Wazuh.

Come i vari agent anche il server contiene molti moduli per diverse operazioni e questi sono [Waz21]:

1. **Agents registration service:** Componente che viene utilizzato per registrare un nuovo agente fornendo chiavi di autenticazioni uniche per ogni agente. Inoltre, essendo eseguito su una rete, fornisce la possibilità di effettuare autenticazioni tramite certificati TLS/SSL o utilizzando una password prefissata.

2. **Agents connection service:** Questo è il modulo predisposto per ricevere informazioni dagli agenti. Utilizza le chiavi scambiate precedentemente per confermare l'identità di ogni agente e per criptare i canali di comunicazione tra gente e server stesso. Inoltre questo servizio fornisce la possibilità di gestire in maniera centralizzata tutte le configurazioni degli agenti registrati, dando la possibilità di modificare direttamente dal server tutti gli agenti.

3. **Analysis engine:** Componente che esegue l'analisi dati. Per seguire questa operazione utilizza dei decoders che permettono di identificare che tipo di informazione sta venendo processate e successivamente estrarre informazioni utili dal messaggio di log (indirizzi IP, ID dell'evento, etc..). All'interno dei vari eventi decodificati è possibile identificare pattern specifici che possono attivare alert o attivare contromisure automatizzate (ad esempio un ban dell'ip), questo è possibile grazie all'uso di regole che possono essere anche customizzate.

4. **Wazuh RESTful API:** Permette l'interazione con l'interfaccia Wazuh. Viene usato per amministrare gli agenti e le impostazioni del server, inoltre permette di monitorare lo stato dell'infrastruttura e lo stato vitale dei vari nodi. Un'altra feature importante è quella di modificare e gestire i decoder, le regole ed eseguire query riguardanti lo stato degli endpoints. Anche la dashBoard Kibana fa uso di questo servizio.

5. **Wazuh cluster daemon:** Modulo utilizzato per far scalare un server Wazuh in maniera orizzontale creando di fatto un cluster. Utilizzare questo approccio affiancando un bilanciamento del carico di rete permette di avere high-availability, in modo da garantire sempre un servizio ad alte prestazioni. Questa componente viene anche utilizzata per comunicare con gli altri server del cluster e mantenerli sincronizzati.

6. **Filebeat:** Componente utilizzati per inviare eventi e i vari alert ad Elasticsearch. Questa operazione viene effettuata in real time leggendo gli output generati dall'analysis engine di Wazuh e successivamente spedirli. Quando connesso ad un cluster Elasticsearch multi nodo fornisce anche bilanciamento della rete.

Come detto precedentemente Wazuh utilizza *MITRE ATT&CK* per arricchire le informazioni in ingresso dagli agenti. MITRE [The21d] è una banca dati globalmente accessibile di tattiche e tecniche di attacco basata sulle osservazioni del mondo reale. Quest'ultime sono elencate in una tabella specifica chiamata *matrice* che contiene nelle celle le tecniche, ovvero le modalità per arrivare all'obiettivo, e nelle colonne le tattiche, cioè l'obiettivo prefissato dell'attacco. Le tecniche sono centinaia e possono suddividersi anche in sotto-tecniche ovvero descrizioni molto accurate del modus operandi, le tattiche invece sono 14 e sono divise in [The21d]:

1. **Reconnaissance:** L'attaccante cerca di estrapolare informazioni dalla macchina attaccata. Questi dati rubati possono essere sfruttati per aiutarsi in altre fasi dell'attacco o per perseguire ulteriori obiettivi con l'utilizzo di altre tattiche, come l'Initial Access. Le informazioni rubate solitamente includono dettagli sull'azienda vittima, sulle infrastrutture o sullo staff.

2. **Resource Development:** L'attaccante tenta di sfruttare risorse per eseguire successive operazioni. Le risorse utilizzate possono essere account rubati o infrastrutture. Queste possono essere impiegate per utilizzare tattiche ulteriori come Command and Control, Initial Access o Defense Evasion

3. **Initial Access:** L'attaccante prova a forzare un ingresso nel network. Questa tattica punta a sfruttare debolezze di server accessibili al pubblico o ad effettuare spearphishing (invio di mail apparentemente sicure).

4. **Execution:** L'attaccante cerca di eseguire codice malevolo sulla macchina attaccata. Questo può essere eseguito o da remoto oppure direttamente in locale. La tattica è solitamente eseguita in combinazione con altre in modo da avere una superficie più ampia di attacco.

5. **Persistence:** L'attaccante cerca di mantenere un malware o un backdoor capace di continuare a garantirgli accesso al sistema della vittima. Potrebbe comportare il rimpiazzo o la manomissione di parti di codice.

6. **Privilege Escalation:** L'attaccante cerca di ottenere permissioni ad alto livello come ad esempio administrator o root. Questo risultato è ottenuto sfruttando vulnerabilità, debolezze del sistema o configurazioni errate. Solitamente a seguito di questo vengono utilizzate tattiche di tipo Persistance in modo da continuare ad accedere con gli stessi permessi.

7. **Defense Evasion:** L'attaccante tenta di non essere scoperto all'interno del sistema. L'obiettivo è quello di disattivare o disinstallare sistemi di sicurezza presenti o nascondere eventuali script. E' possibile anche che vengano utilizzati processi già presenti sul pc per offuscare la presenza di malware.

8. **Credential Access:** L'attaccante prova a rubare account e password. Le tecniche principalmente utilizzate si basano sul keylogging e sul credential dumping. Sfruttare credenziali legittime può fornire un elusione maggiore all'attaccante rendendolo complesso da rilevare. Inoltre da la possibilità di creare ulteriori account per realizzare i suoi obiettivi.

9. **Discovery:** Tentativo di scoperta dell'ambiente attaccato. Vengono utilizzate per carpire informazioni e per scoprire cosa è possibile controllare, prima di decidere in quale modo agire.

10. **Lateral Movement:** Tattica utilizzata per cercare di muoversi all'interno del sistema attaccato. Solitamente viene esplorata anche la rete per cercare un bersaglio, prendendone poi accesso. Verranno installati strumenti per accesso

remoto o verranno utilizzate credenziali legittime per passare da un sistema all'altro.

11. **Collection:** L'attaccante tenta di avere un controllo sulle informazioni d'interesse (file audio, email, password, etc...) con l'uso di keylogger o screenshot. I file vengono così controllati per poi perseguire altri obiettivi. Solitamente i dati vengono poi sottratti alla vittima.

12. **Command and Control:** L'attaccante cerca di comunicare con la macchina compromessa per prenderne il controllo. Solitamente viene contattata attraverso traffico rete camuffato, questo per fare in modo che sia il più nascosto possibile

13. **Exfiltration:** L'attaccante dopo aver preso controllo di dati inizia ad esportare i file su un proprio server. I file sono compressi e crittografati per renderne più difficile la scoperta del furto.

14. **Impact:** L'attaccante tenta di manipolare, eliminare dati all'interno del sistema compromesso. Questa tattica può essere utilizzata per eseguire ulteriori azioni o per coprire una plausibile breccia del sistema.

Oltre ad utilizzare *MITRE ATT&CK* per rilevare intrusioni possono essere dispiegati altri plugin, propedeutici alla scoperta di file mancanti o di processi manomessi. Uno di questo è *Osquery* [Waz21] ed è uno strumento utilizzato dal sistema operativo (disponibile sia su Windows, Linux e macOS) che permette di mostrare il sistema operativo come un database relazionale ad alte prestazioni. Ciò permette di sfruttare query SQL per ricercare informazioni e file all'interno del sistema. Le tabelle sono astratte e rappresentano solitamente processi in esecuzione, moduli kernel, connessioni aperte, plugin dei browser o eventi hardware. Il suo obiettivo è quello di notificare i possibili cambiamenti sull'infrastruttura del sistema generando quindi log per ognuno di questi eventi.

Un altro plugin esterno utilizzabile tramite Wazuh è *VirusTotal* [Waz21]. Esso è un servizio online che permette di eseguire analisi su file e anche su URLs con lo scopo di rilevare virus, worms, trojans e altri tipi di oggetti malevoli, utilizzando diversi tipi di antivirus o scanner per siti web (permettono di trovare parti di codice html o php insicuri).

VirusTotal inoltre permette di salvare i risultati dell'analisi eseguite e questo dà la possibilità di ricercare un hash di un file specifico. Inviando l'hash a *VirusTotal* è possibile venire a conoscenza di una precedente analisi di quel file specifico e quindi analizzare

il risultato dell'antecedente rilevazione. Inoltre fornisce la possibilità di utilizzare API che permettono di accedere ad informazioni generate dall'engine senza dover utilizzare alcun interfaccia web.

Un'altra funzionalità importante è quella di poter controllare i logs dei vari container docker presenti all'interno degli host con un agent. Questo permette di ricevere un alert ogni qualvolta un container subisce un cambiamento (un riavvio, un cambiamento al dockerfile).

Il server inoltre permette di archiviare tutti gli alert e non al suo interno, oltre ad inviarli al nodo master di Elasticsearch. I file archiviati sono in formato JSOn e sono compressi e firmati digitalmente utilizzando checksum di tipo *MD5*, *SHA1* e *SHA256* [Waz21]. Ogni giorno vengono archiviati tutti i log che verranno mantenuti per un lasso di tempo configurabile.

3.4 Suricata

Suricata [Ope16] è un IDS open-source ma può venire classificato anche come un *Network Intrusion Detection System*. Il fatto di poter reagire in modo reattivo alle minacce e agli eventi può anche renderlo un *IPS (Intrusion Prevention System)* rendendolo quindi molto completo.

Inoltre è eseguibile sui più disparati sistemi operativi ed è in grado di essere eseguito con alte prestazioni utilizzando un'architettura multi-threaded e sfruttando l'accelerazione dell'hardware. Grazie a queste sue proprietà è adatto per essere combinato insieme a Wazuh potendo spedire a quest'ultimo il file eve.json.

Il file sovracitato è la destinazione di tutti i log Suricata contenenti principalmente il contenuto di ogni singolo pacchetto passante sul network sniffato.

Tutti i pacchetti aventi al loro interno malware o simili verranno notificati dentro eve.json.

I pacchetti sono ispezionati basandosi su delle regole che possono applicare quattro azioni diverse che sono [Ope16]:

1. **Pass:** Se la firma del pacchetto corrisponde a quella della regola Suricata contenente *pass* smette l'ispezione e lascia passare quel singolo.

2. **Drop:** Utilizzato per sfruttare le qualità di *Intrusion Prevention System*. Se la firma del pacchetto corrisponde a quella della regola contenente *drop*, il pacchetto viene immediatamente scartato terminando quindi il controllo. Il ricevente non avrà alcun informazione su quello che sta succedendo, se non attraverso al visione di un timeout all'interno di Eve.json.

3. **Reject:** In questo caso sia colui che lo spedisce che colui che lo riceve vengono notificata dell'azione eseguita. Esistono due tipi di "Reject", il primo riguarda i pacchetti TCP che vengono vengono gestiti con "Reset-Packet", invece se il pacchetto è di un qualsiasi altro tipo questo verrà scartato come nell'azione "Drop".

4. **Alert:** Se la firma del pacchetto corrisponde a quella della regola contenente *alert* Suricata genererà un alert visibile soltanto dall'amministratore.

4 Implementazione e configurazione

In questo capitolo si tratteranno le *configurazioni* utilizzate per l'ambiente Wazuh, e per la gestione della server Farm. Le *configurazioni* tratteranno quindi ogni aspetto relativo all'hardening, al logging e all'Intrusion Prevention. Tutte le decisioni effettuate sono state pensate per essere migliorate e modificate in futuro, essendo il sistema estremamente flessibile e aggiornato settimanalmente. Importanti saranno anche le immagini Docker e le accortezze utilizzate per ottimizzare i procedimenti di aggiornamento di Wazuh.

4.1 Creazione Wazuh Master tramite Docker Compose

Per la creazione del Wazuh Master ci siamo avvalsi di Docker Compose. Questo permette di definire ed eseguire le varie applicazioni desiderate composte da più servizi, che saranno poi create all'interno di un docker *container*. Grazie all'utilizzo di un file YAML vengono definiti i servizi che comporranno poi un'applicazione distribuita utilizzando Compose, inoltre all'interno del file sono presenti anche le strutture riguardanti i container che implementeranno i vari servizi. Questi file YAML sono chiamati *compose* file e solitamente hanno come nome standardizzato *docker-compose.yml*. Il file è suddiviso in sezioni specifiche nella quale andranno inserite le varie informazioni che andranno a comporre il nostro container. Ad esempio nella sezione *services* andranno inseriti tutti i parametri specifici dei container che andranno a definire il servizio preso in esame. Qui andranno inserite quindi le immagini da istanziare nel container, la tipologia di rete che si vuole utilizzare, i volumi, le politiche di riavvio dell'applicazione, le variabili d'ambiente e le porte in esposizione sull'interfaccia di rete. Oltre a questa sezione vengono specificate anche la sezione *version* dove si andrà a specificare la versione utilizzata dal Docker Compose per utilizzare il file, *networks* per le reti che verranno utilizzate e *volumes* dove si definiranno strutture dati da utilizzare nell'applicazione che verrà creata.

A seguire è mostrato il docker-compose file utilizzato per la generazione dei container.

```
version: '3.7'

services:
  wazuh:
    image: wazuh/wazuh-odfe:4.1.4
    hostname: wazuh-manager
    restart: always
    ports:
      - "1514:1514"
      - "1515:1515"
      - "514:514/udp"
      - "55000:55000"
    environment:
      - .env_wazuh
    volumes:
```

```
      - ./ossec_api_configuration:/var/ossec/api/configuration
      - ./ossec_etc:/var/ossec/etc
      - ossec_logs:/var/ossec/logs
      - ossec_queue:/var/ossec/queue
      - ossec_var_multigroups:/var/ossec/var/multigroups
      - ossec_integrations:/var/ossec/integrations
      - ossec_active_response:/var/ossec/active-response/bin
      - ossec_agentless:/var/ossec/agentless
      - ossec_wodles:/var/ossec/wodles
      - filebeat_etc:/etc/filebeat
      - filebeat_var:/var/lib/filebeat

  elasticsearch:
    image: amazon/opendistro-for-elasticsearch:1.12.0
    hostname: elasticsearch
    restart: always
    ports:
      - "9200:9200"
    environment:
      - .env_elasticsearch
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
    volumes:
        - ./dfe-data1:/usr/share/elasticsearch/data:rw
        - ./internal_users.yml:/usr/share/elasticsearch/plugins
          /opendistro_security/securityconfig/internal_users.yml
        - ./config.yml:/usr/share/elasticsearch/plugins
          /opendistro_security/securityconfig/config.yml
        - ./elasticsearch.yml:/usr/share/elasticsearch/plugins
          /opendistro_security/securityconfig/elasticsearch.yml
  kibana:
    image: wazuh/wazuh-kibana-odfe:4.1.4
    hostname: kibana
    restart: always
    ports:
      - 443:5601
    environment:
      - .env_kibana
    depends_on:
      - elasticsearch
    links:
      - elasticsearch:elasticsearch
      - wazuh:wazuh

volumes:
  ossec_api_configuration:
  ossec_etc:
  ossec_logs:
  ossec_queue:
  ossec_var_multigroups:
  ossec_integrations:
  ossec_active_response:
  ossec_agentless:
  ossec_wodles:
  filebeat_etc:
  filebeat_var:
```

Per il docker-compose file si è deciso di utilizzare quello ufficiale di Wazuh, che è stato successivamente modificato per permettere di avere una diversificazione nelle password e nelle impostazioni standard. Ad esempio le password sono state tutte occultate grazie all'uso di environment variables, in modo da poter distribuire in sicurezza il compose file. Un altro aspetto modificato è stato l'heap size di ElasticSearch, che è stato aumentato per fare in modo di avere una cache interna più capiente. Questo però non va incrementato esageratamente poiché si potrebbe incorrere in problemi di queueing.

L'aggiornamento del Wazuh Master è possibile effettuarlo tramite questo file cambiando semplicemente la versione del servizio "wazuh", facendo questo però conviene sempre eliminare i volumi esistenti sul container per evitare problemi di compatibilità tra le impostazioni preesistenti e quelle aggiornate. Sarebbe possibile anche effettuare un aggiornamento ogni qualvolta che il docker-compose file viene utilizzato, questo però porterebbe a dover aggiornare tutti i nodi sulla quale risiedono gli *agent*i wazuh tutte le volte che la situazione si verifica. Conviene quindi aggiornare solo quando l'aggiornamento successivo è perfettamente stabile ed integrato.

Bisogna prestare attenzione inoltre alle versione di Kibana e di ElasticSearch ogni volta che aggiorniamo il Wazuh Master. Potrebbero esserci problemi di compatibilità tra i tre prodotti sopracitati portando quindi a malfunzionamenti o ad errori risolvibili solo tramite aggiornamento.

Per aggiornare tutti i nodi presenti all'interno dell'ambiente è possibile utilizzare degli script già forniti da Wazuh stesso. Questi permettono di sapere quali sono i nodi con un *agent* al loro interno e quale versione stanno utilizzando al momento. Inoltre sempre tramite questi è possibile aggiornare tutti gli *agent* tramite il Wazuh Master, senza agire in alcuna maniera sul nodo prefissato.

4.2 Configurazione Wazuh

Dopo aver modificato il Dockerfile in maniera opportuna è stato possibile creare l'ambiente effettivo. Avendo già scelto i server da utilizzare come master si è passato ad identificare server che necessitavano un servizio di real-time monitoring. Lo scopo era quello di creare una specie di cluster capace di poter essere esteso ogni qualvolta fosse necessario. Ovviamente l'aggiunta di nuovi master è resa necessaria solo in caso di carichi di lavoro elevati e questa possibilità si ha solo quando si hanno un grande numero di agenti, capaci di inviare centinaia di migliaia di log ogni giorno. Nell'immagine seguente sarà presente un Cluster comprensivo di più server capaci di ricevere log. Nel nostro caso invece ci sarà solamente un server master capace sia di ricevere i log sia di elaborarli, questo poiché il quantitativo di lavoro sul nostro cluster è contenuto.

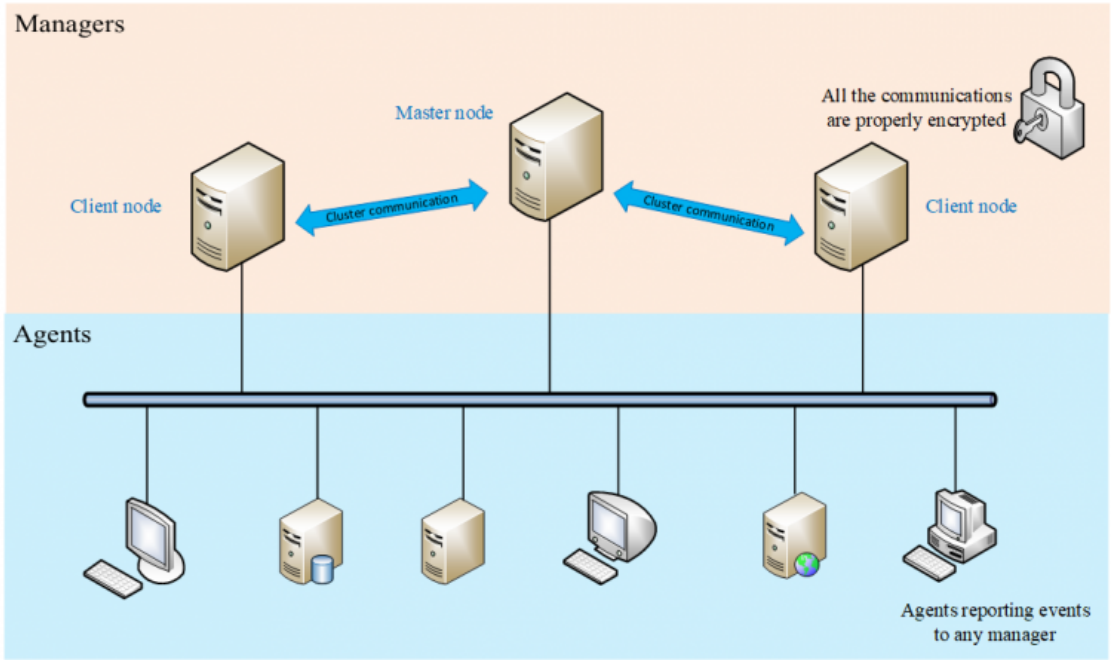

Figure 7: Ambiente Wazuh.

4.2.1 Installazione Agenti

Per ottenere questo risultato risulta necessario procedere all'installazione dei vari agenti. Gli agenti possono essere deployati su qualunque sistema operativo e anche su una qualsiasi architettura di questi ultimi. L'installazione risulta essere molto user-friendly grazie all'interfaccia di Wazuh che ci permette semplicemente di copiare un comando e incollarlo sulla console del sistema da noi indicato. In base alla piattaforma scelta il comando varierà e saranno diversificate anche le azioni da susseguirsi. I comandi da immettere differiscono anche in base al tipo di init che possiede il sistema (ad esempio ci sono differenza tra systemd e SysV init, anche se il secondo è estremamente obsoleto).

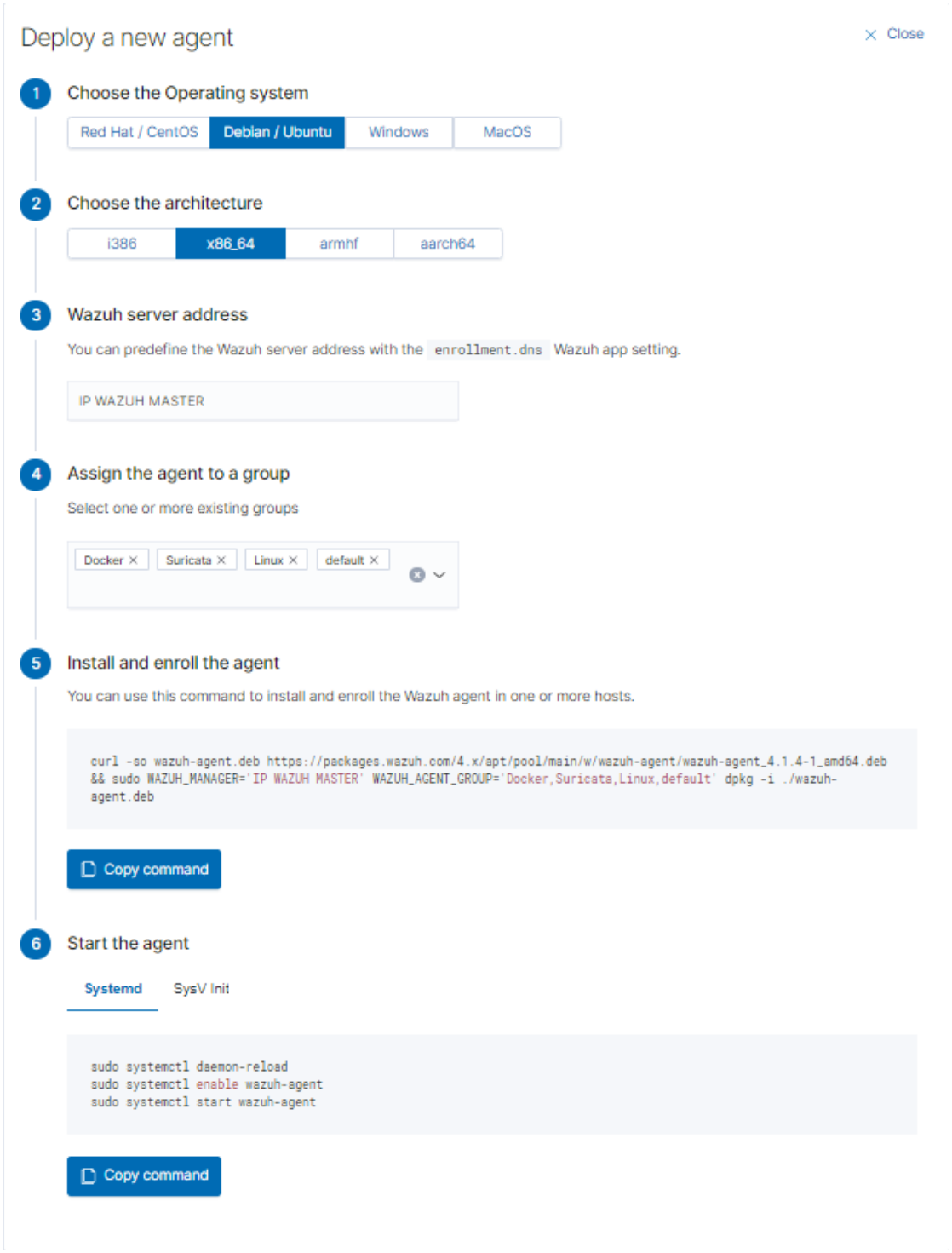

Figure 8: Interfaccia per installazione agente.

L'interfaccia si divide in diverse sezioni e per l'installazione di un agente su macchina Debian o Ubuntu queste sono sei.

1. In questa sezione viene scelto il sistema operativo della macchina sulla quale deve essere installato l'agente. Cambiando l'OS cambieranno di conseguenza anche le altre sezioni.

2. In questa sezione verrà scelta l'architettura sulla quale installare l'agente. Wazuh permette l'installazione su quasi tutti i tipi di architettura e questo permette di poter controllare anche macchine piuttosto obsolete.

3. In questa sezione è necessario immettere l'ip del Wazuh Master. Se non viene indicata in questo momento sarà sempre possibile cambiarla all'interno dei file di configurazione dell'agente, Il file di configurazione in questione si trova al percorso */var/ossec/etc/ossec.conf*, se il manager è composto da più worker è anche possibile aggiungere tutti i vari ip dei nodi master in modo da consentire, in caso di crash di uno di questi, di avere il servizio attivo permanentemente.

```
<client>
    <server>
        <address>172.0.0.4</address>
        <port>1514</port>
        <protocol>tcp</protocol>
    </server>
    <server>
        <address>172.0.0.5</address>
        <port>1514</port>
        <protocol>tcp</protocol>
    </server>
    <config-profile>ubuntu, ubuntu18, ubuntu18.04</config-profile>
    <notify_time>10</notify_time>
    <time-reconnect>60</time-reconnect>
    <auto_restart>yes</auto_restart>
    <crypto_method>aes</crypto_method>
</client>
```

Come si può vedere da questo esempio gli ip all'interno del tag *"server"* sono quelli relativi ai worker node del manager. In caso il primo non sia disponibile l'agente inizierà ad utilizzare il nodo secondario. In questa sezione è possibile quindi aggiungere quanti nodi worker si vuole. Inoltre è possibile aggiungere ulteriori opzioni come *"notify time"* che indica i secondi che intercorrono tra le varie prove di check-in tra agent e manager. Altre opzioni possono essere *"time-reconnect"* che indica il numero di secondi tra una riconnessione e l'altra, questo valore deve essere sempre superiore a notify-time. Ad esempio utilizzando i valori base di 10 secondi per notify e 60 per time-reconnect quando si verificherà

un fallimento di check-in la riconnessione verrà effettuata solamente dopo 60 secondi. La procedura sarà ripetuta fino a che non arriverà un evento di check-in valido.

4. Nella quarta sezione l'interfaccia da la possibilità di assegnare il nostro agent a diversi gruppi. Questi ultimi si riferiscono alla possibilità di dividere i vare agenti in base alle configurazione che riteniamo necessarie per la loro esecuzione. Ad esempio nel nostro caso mettendo un agente all'interno del gruppo Docker è possibile sfruttare la funzione di Wazuh di inviare al master i log di stato dei vari container docker in esecuzione sull'host (sapremo quindi quando essi si riavviano, spengono etc..). E' possibile inserire un agente all'interno di tutti i gruppi in nostro possesso.

5. In questa sezione Wazuh ci fornisce il comando da immettere all'interno della console per installare l'agente. Il comando ovviamente cambierà in base alle selezioni effettuate precedentemente.

6. Questa sezione non è sempre presente in quanto determinati sistemi operativi non necessitano di ulteriori comandi oltre all'installazione. Nel caso di Debian invece è necessario attivare i vari deamon per rendere il funzionamento di Wazuh costante.

4.2.2 Gestione Agenti

Dopo che l'installazione di un agente è andata a buon fine questo verrà mostrato in un'interfaccia dove sono contenuti tutti gli agenti in funzione e non. Oltre a questo sono mostrate anche diverse informazioni come il nome del server e il suo indirizzo IP, il nodo master alla quale fa riferimento, la data di registrazione e a quali gruppi appartiene l'agente.

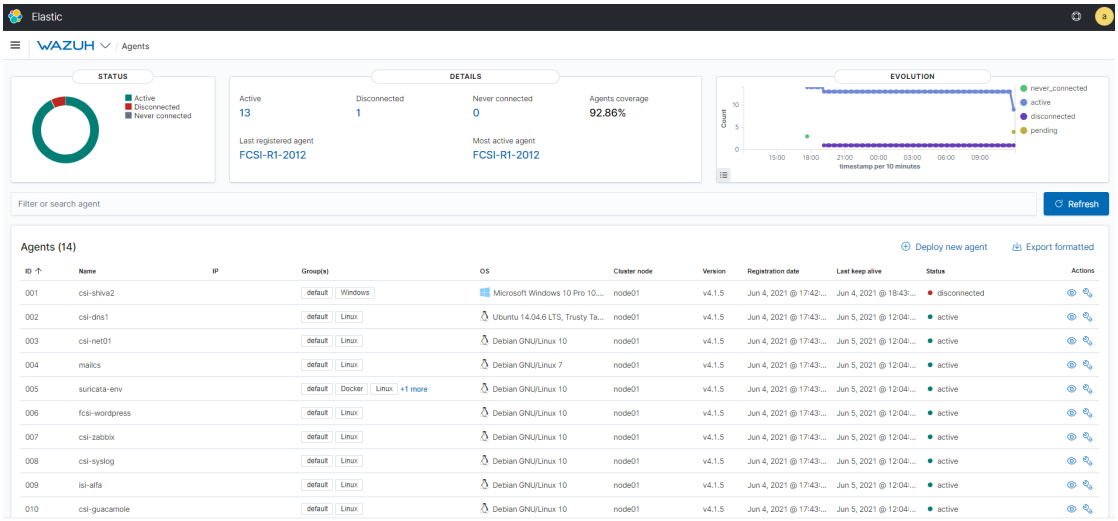

Figure 9: Interfaccia per visualizzazione agenti.

In alto a destra si possono vedere i cambianti di stato che gli agent possono avere durante il loro ciclo di vita. Questa feature può essere effettivamente utile quando un agent rimane attivo ma non produce più alcun tipo di messaggio. Nel grafico quindi non sarà mostrato alcun tipo di colore facendo così mostrare una mancanza nella linea di vita, scoprendo quindi quale agente ha smesso di funzionare.
Ci sono 4 diversi stati ed essi sono:

1. **active:** L'agente è attivo e ha compiuto con successo la registrazione, ora potrà comunicare con il master.

2. **Never connected:** L'agente è stato registrato al master ma non ha ancora effettuato la connessione.

3. **Pending:** Il processo di autenticazione è in sospeso. Il manager dopo aver ricevuto la richiesta di connessione non ha più ricevuto nulla, aspettando quindi nuove informazioni. Solitamente questo tipo di problema è dato da problematiche lato firewll. Appena avviato ogni agente sarà in questo stato per un brevissimo periodo.

4. **Disconnected:** Il manager considera disconnessi gli agenti che non inviano alcun tipo di messaggio di attività entro il tempo di disconnessione dell'agente (solitamente impostato di default a 10 minuti)

Gli agenti sono inoltre tutti quanti updatabili tramite il manager del cluster. Questo è possibile attraverso uno script all'interno di una cartella specifica che ci permette di selezionare quale client aggiornare e anche a quale versione portarlo.

```
./var/ossec/bin/agent_upgrade
```

```
usage: agent_upgrade.py [-h] [-a AGENTS [AGENTS ...]] [-r REPOSITORY] [-v VERSION] [-F] [-s] [-l]
[-f FILE] [-x EXECUTE] [--http]

optional arguments:
  -h, --help            show this help message and exit
  -a AGENTS [AGENTS ...], --agents AGENTS [AGENTS ...]
                        Agent IDs to upgrade.
  -r REPOSITORY, --repository REPOSITORY
                        Specify a repository URL. [Default: packages.wazuh.com/4.x/wpk/]
  -v VERSION, --version VERSION
                        Version to upgrade. [Default: latest Wazuh version]
  -F, --force           Allows reinstall same version and downgrade version.
  -s, --silent          Do not show output.
  -l, --list_outdated   Generates a list with all outdated agents.
  -f FILE, --file FILE  Custom WPK filename.
  -x EXECUTE, --execute EXECUTE
                        Executable filename in the WPK custom file. [Default: upgrade.sh]
  --http                Uses http protocol instead of https.
```

Grazie a questo non sarà necessario dover accedere ad ogni singola macchina per aggiornare l'agent ma basterà semplicemente fare un comando di questo tipo:

```
./var/ossec/bin/agent_upgrade -a 001 002 003 004
```

Fatto ciò gli agenti **001, 002, 003, 004** saranno aggiornati all'ultima versione disponibile, solitamente essa sarà la stessa versione del manager. Successivamente saranno anche riavviati per caricare correttamente le nuove funzionalità installate dall'aggiornamento.

Gli agenti possono avere due diversi tipi di configurazione. La prima è la *configurazione locale* dove un agente sfrutta un file chiamato **ossec.conf** per reindirizzare i log verso il manager. Questo è un file XML e le varie *configurazioni* possono essere inserite all'interno delle loro apposite sezioni. Solitamente questa configurazione è usata per un cluster con pochissimi agent poiché il procedimento di modifica di ossec.conf dovrà essere ripetuto su tutte le macchine in caso di necessità.
Quindi se si ha un un ambiente Manager-Agent con un grande numero di questi ultimi è conveniente passare alla *configurazione centralizzata*. Questa permette di avere la stessa identica configurazione per tutti gli agenti, che potranno essere anche suddivisi in gruppi in base ai tipi di log che essi dovranno inviare, inoltre i vari agenti potranno essere inseriti anche in più gruppi ottenendo quindi tutte le varie *configurazioni* relative al gruppo in cui risiedono. In entrambe le *configurazioni* inoltre è possibile inserire non solo i log da inviarli, ma anche esplicitare di compiere azioni nel caso di determinati eventi. Ad esempio in base alla gravità di un evento è possibile generare un alert che sarà inviato ad un email scelta, in modo da contattare in maniera repentina

l'amministratore del sistema. Il file nel caso della *configurazione centralizzata* è chiamato **agent.conf** e anche in questo caso sarà un file di tipo XML del tutto similare ad **ossec.conf**. I log possono avere diversi formati e solitamente ricadono nel tipo *syslog*, usato principalmente per log di sistema. Nel caso in oggetto si hanno 4 diversi gruppi:

1. **Linux:** In questo gruppo risiedono tutti gli agenti che sono stati installati su un sistema di tipo Linux/Debian. La configurazione è ottimizzata per fare in modo che tutti i vari file più importanti siano controllati. Nel nostro caso è stata aggiunta anche la possibilità di inviare i log audit, in modo da poter tenere traccia di tutti i cambiamenti in atto nel sistema.

```
<agent_config>
        <!-- Shared agent configuration here -->
        <localfile>
                <log_format>audit</log_format>
                <location>/var/log/audit/audit.log</location>
        </localfile>
        <localfile>
                <log_format>syslog</log_format>
                <location>/var/ossec/logs/active-responses.log</location>
        </localfile>
        <localfile>
                <log_format>syslog</log_format>
                <location>/var/log/messages</location>
        </localfile>
        <localfile>
                <log_format>syslog</log_format>
                <location>/var/log/auth.log</location>
        </localfile>
        <localfile>
                <log_format>syslog</log_format>
                <location>/var/log/syslog</location>
        </localfile>
        <localfile>
                <log_format>syslog</log_format>
                <location>/var/log/dpkg.log</location>
        </localfile>
        <localfile>
                <log_format>syslog</log_format>
                <location>/var/log/kern.log</location>
        </localfile>
</agent_config>
```

2. **Windows:** Gruppo in cui risiedono tutti gli agenti installati su host Windows Qui è stata lasciata la configurazione standard non avendo aggiunto ulteriori plugin.

```
<agent_config>
        <!-- Shared agent configuration here -->
        <localfile>
                <location>Application</location>
                <log_format>eventchannel</log_format>
```

```
        </localfile>
        <localfile>
        <location>Security</location>
        <log_format>eventchannel</log_format>
        <query>
          Event/System[EventID != 5145 and EventID != 5156 and EventID != 5447 and
          EventID != 4656 and EventID != 4658 and EventID != 4663 and EventID != 4660 and
          EventID != 4670 and EventID != 4690 and EventID != 4703 and EventID != 4907 and
          EventID != 5152 and EventID != 5157]
        </query>
        </localfile>
        <localfile>
                <location>System</location>
                <log_format>eventchannel</log_format>
        </localfile>
        <localfile>
                <location>active-response\active-responses.log</location>
                <log_format>syslog</log_format>
        </localfile>
</agent_config>
```

3. **Suricata:** In questo gruppo sono inseriti tutti gli host che utilizzano un NIDS Suricata. Nel caso specifico viene utilizzato in combinazione con docker. Il client Suricata invia solamente il file contiene semplicemente gli eventi che vengono rilevati sulla rete. E' possibile inoltre, grazie a Suricata, controllare il flow di dati di eve.json. Qui sono contenuti tutti gli eventi che sono stati rilevati, e molti di questi possono essere poco interessanti o talmente comuni da nascondere quasi eventi di minaccia ben più gravi. Infatti molti eventi comuni sono stati eliminati dalla rilevazione cambiando le regole di Suricata grazie al file suricata.rules. Il file in questione è aggiornabile quotidianamente per poter scovare ulteriori eventi potenzialmente pericolosi all'interno della nostra rete.

```
<agent_config>
        <!-- Shared agent configuration here -->
        <localfile>
                <log_format>json</log_format>
                <location>/docker/suricata/log/eve.json</location>
        </localfile>
</agent_config>
```

4. **Docker:** In questo gruppo verranno aggiunti tutti i client che hanno un container docker funzionante al loro interno. Permette ai partecipanti del gruppo di inviare i log dei container docker al posto di doverli visualizzare manualmente all'interno del client stesso. Per utilizzare questa funzionalità è però necessario avere determinati prerequisiti. Bisogna utilizzare un Linux-based OS, una versione di Python dalla 2.7 in poi, la libreria di Python docker installabile utilizzando il comando *pip install docker* e una versione di Wazuh dalla 3.9 in poi, versione nella quale è stato aggiunta la funzionalità dei *docker-listener*.

```
<agent_config>
    <!-- Shared agent configuration here -->
    <wodle name="docker-listener">
            <disabled>no</disabled>
            <interval>10m</interval>
            <attempts>5</attempts>
            <run_on_start>yes</run_on_start>
    </wodle>
</agent_config>
```

4.2.3 Hardening e prevenzione

Wazuh inoltre fornisce un servizio di hardening estremamente efficiente. Questo è chiamato Security Configuration Assessment module (o SCA) d è stato aggiunto nell'aggiornamento 3.9.0. Lo scopo di questo è cercare di migliorare la sicurezza dell'host riducendone la *superficie attaccabile*. Questo processo è chiamato per l'appunto hardening e la valutazione delle *configurazioni* perpetrata dal modulo SCA è una pratica estremamente efficiente.

Lo SCA esegue delle scansioni per scoprire eventuali esposizioni o *configurazioni* malprodotte negli host, sottoposti a monitorizzazione da parte di un agente. Queste valuteranno le i vari oggetti interessati per mezzo d file di policy, contenenti svariate regole per testare le *configurazioni* dell'host. Le scansioni possono produrre diverse soluzioni per limitare la *superficie attaccabile* ad esempio possono essere il cambiamento di password, la rimozione di software inutilizzato, la disabilitazioni di servizi non necessari per il funzionamento generale dell'host, eventi audit dello stack TCP/IP o anche smontaggio di filesystem non usati.

Le policy relative al modulo sono scritte in YAML. Questo permette quindi di avere una maggior comprensione dello scopo e anche per permettere ai diversi user di scriverne di proprie o di estendere quelle già esistenti adattandole alle necessità del sistema stesso. Wazuh ad esempio rilascia già di base policy redatte in base ai benchmark CIS, essendo quest'ultimi uno standard elevato per hardening effettuato su host.

Per attivare questo modulo risulta necessario aggiungere una nuova parte al file ossec.conf. Nel nostro caso invece, utilizzando una versione di Wazuh successiva alla 3.9 il modulo era già incluso in ogni agente evitandoci quindi di inserirlo all'interno di ogni file.

```
<sca>
  <enabled>yes</enabled>
  <scan_on_start>yes</scan_on_start>
  <interval>12h</interval>
  <skip_nfs>yes</skip_nfs>
</sca>
```

Queste sono solo alcune delle opzioni che possono essere inserite all'interno della sezione di codice. Ad esempio è possibile inserire anche a che ora e giorno della settimana deve essere effettuato lo scan.

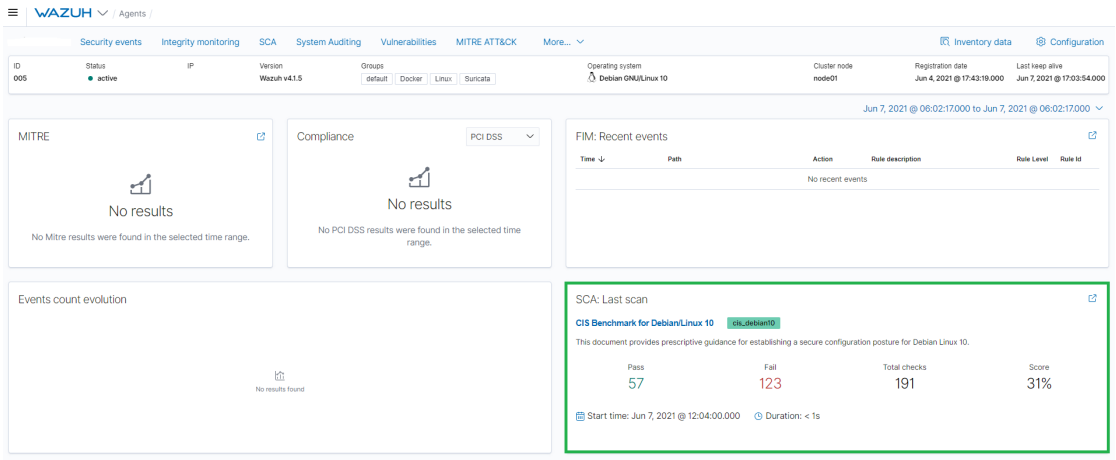

Figure 10: Overview dell'agente con modulo SCA evidenziato.

Come si vede dall'immagine nella sezione modulo SCA si possono vedere quanti test sono stati effettuati e quanti sono quelli passati e falliti. Oltre a questo è assegnato anche un punteggio in percentuale, più è alto questo punteggio maggiore sarà la sicurezza di questo host e quindi minore sarà la sua *superficie attaccabile*. Nell'immagine in esempio il numero di test totale è superiore al numero di test eseguiti (sia falliti che riusciti), questo perchè a volte qualche valutazioni non può essere eseguita. Questo può aiutare a capire quanto effettivamente la nostra macchina sia sicura ed incentivarci anche ad utilizzare questo modulo il più possibile.

Dall'immagine si può notare che Wazuh aiuta l'utente a capire come risolvere le varie problematiche all'interno dell'host in oggetto. Nella voce Rationale spiega lo scopo del test in modo da dare anche un contesto al problema. In Remedation invece fornisce la soluzione dell'inconsistenza rendendo così semplicissimo risolvere le varie problematiche rilevate. La Description invece deludica all'utente la motivazione della soluzione facendo capire il perchè si è agito in una determinata maniera.

Un'altra feature estremamente interessante implementata da Wazuh è la capacità di identificare le *vulnerabilità* all'interno dei pacchetti applicativi installati sulle macchine. Questo permette agli utenti di verificare le varie falle attaccabili all'interno del loro sistema. Le *vulnerabilità* (o CVE, Common Vulnerabilities and Exposures) sono suddivise in 4 tipi diversi che sono Critical, High, Medium, Low. Inoltre le CVE trovate sono poi categorizzate grazie ai CWE (Common Weakness Enumeration) che permetto quindi di capire qual è il tipo di *vulnerabilità* presenta all'interno del pacchetto.

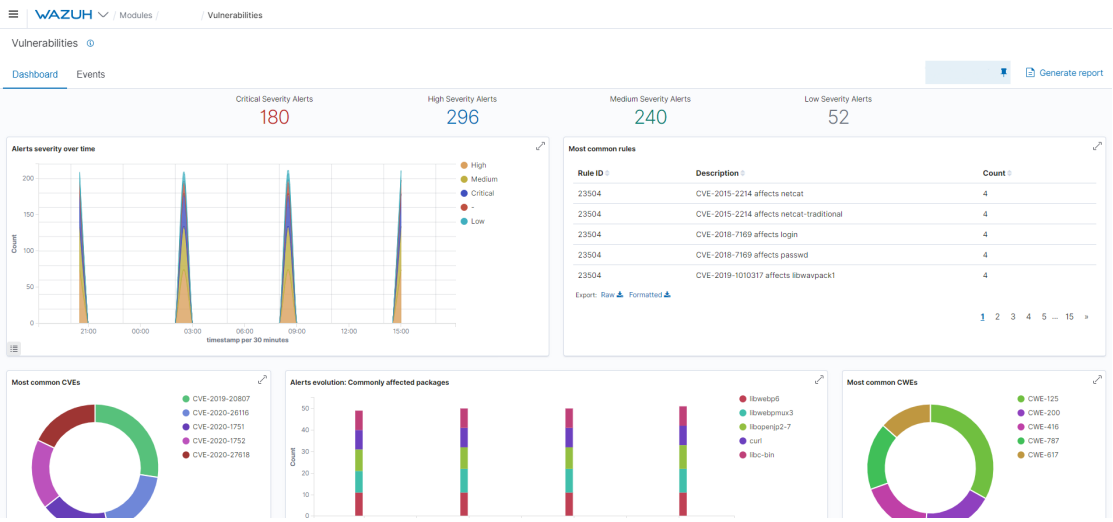

Figure 11: Dashboard Vulnerabilities.

All'interno di ogni CVE sarà possibile trovare due voci particolarmente utili che sono Rationale e References. La prima sezione descriverà la *vulnerabilità* trovata indicando ad esempio che può essere trovata in aggiornamenti precedenti a quello odierno, solitamente verranno indicati anche i rischi che a cui si può incorrere se non risolta. La seconda invece conterrà svariati link che ricondurranno l'utente alla documentazione spiegante la risoluzione della *vulnerabilità*, quest'ultima generalmente è corretta da colui che rilascia il pacchetto incriminato e solitamente viene semplicemente consigliato di aggiornare l'oggetto sotto analisi.

Per attivare questa feature sugli agent è necessario precedentemente inserire porzioni di codice XML all'interno del file di configurazione *ossec.conf* del Wazuh Manager.

```
<vulnerability-detector>
   <enabled>yes</enabled>
   <interval>5m</interval>
   <ignore_time>6h</ignore_time>
   <run_on_start>yes</run_on_start>

   <!-- Ubuntu OS vulnerabilities -->
   <provider name="canonical">
     <enabled>yes</enabled>
     <os>trusty</os>
     <os>xenial</os>
     <os>bionic</os>
     <os>focal</os>
     <update_interval>1h</update_interval>
   </provider>

   <!-- Debian OS vulnerabilities -->
   <provider name="debian">
```

```
    <enabled>yes</enabled>
    <os>stretch</os>
    <os>buster</os>
    <update_interval>1h</update_interval>
  </provider>

  <!-- RedHat OS vulnerabilities -->
  <provider name="redhat">
    <enabled>yes</enabled>
    <os>5</os>
    <os>6</os>
    <os>7</os>
    <os>8</os>
    <update_interval>1h</update_interval>
  </provider>

  <!-- Windows OS vulnerabilities -->
  <provider name="msu">
    <enabled>yes</enabled>
    <update_interval>1h</update_interval>
  </provider>

  <!-- Aggregate vulnerabilities -->
  <provider name="nvd">
    <enabled>yes</enabled>
    <update_from_year>2010</update_from_year>
    <update_interval>1h</update_interval>
  </provider>
</vulnerability-detector>
```

In questa porzioni di codice sono stati aggiunti tutti i vari sistemi e le varie distribuzioni che Wazuh supporta per la vulnerability detection, in modo tale da poter aggiungere qualunque macchina con qualunque sistema ed avere già presente questa funzionalità senza dover ulteriormente operare sul file di configurazione del Manager.

Per i vari agent invece si è dovuto aggiungere un ulteriore porzione di codice per attivare il plugin, permettendo quindi al Manager di analizzare l'host sulla quale è eseguito l'agente. Il codice proposto è già funzionante per ogni qualsivoglia sistema operativo.

```
<wodle name="syscollector">
        <disabled>no</disabled>
        <interval>1h</interval>
        <os>yes</os>
        <packages>yes</packages>
        <hotfixes>yes</hotfixes>
</wodle>
```

4.2.4 Logging e Integrity Monitoring

Una delle funzionalità più importanti di Wazuh è quella di logging che ci permette di avere in real-time informazioni derivanti dai vari host presenti all'interno del nostro sistema. Questa componente permette di ricevere log utilizzando file di testo e eventi Windows. Inoltre è possibile anche inviare log utilizzando il protocollo di rete Syslog, che risulta estremamente utile in caso di log di Firewall o di altri dispositivi.

Lo scopo di questo modulo è quello di poter identificare gli errori derivanti dal sistema o da applicazioni installate, poter riconoscere determinate configurazioni mal scritte, rilevare possibili tentativi di intrusione oppure verificare l'esistenza di violazione di policy o problemi di sicurezza.

A livello computazionale gli agenti richiedono poca memoria e CPU poiché il loro compito è semplicemente quello di inviare i vari eventi collezionati al manager. D'altro canto quest'ultimo potrebbe richiedere molte più risorse poiché ciò dipenderà da quanti EPS (Events per second) dovrà analizzare.

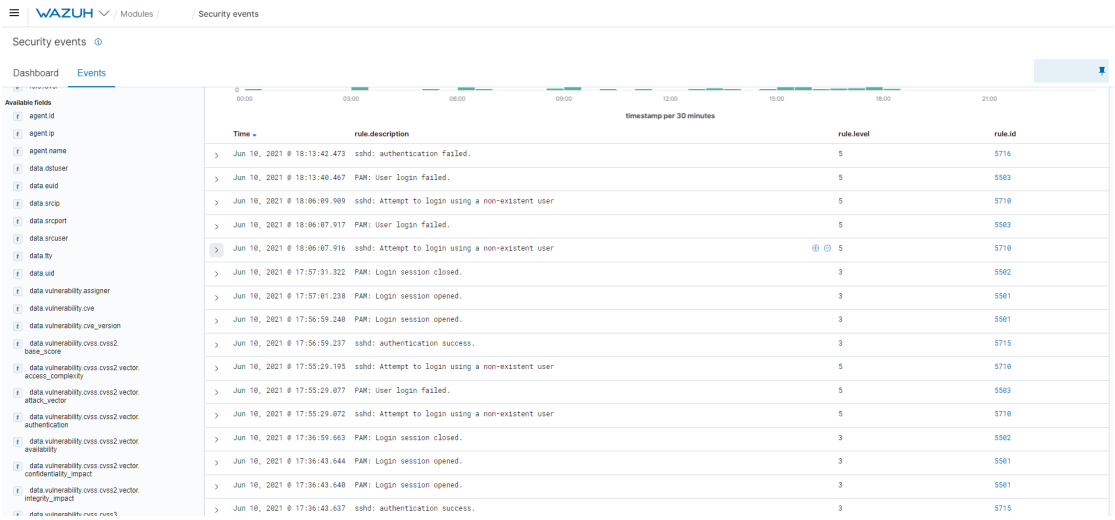

Figure 12: Dashboard per visualizzazione log.

Nella figura sovrastante si può vedere come i log sono diversificati in base al tipo di evento inviato. Inoltre Wazuh nella colonna *rule.level* ci indica anche quanto questo evento possa risultare pericoloso. I livelli arrivano fino al 12 ed è possibile anche inviare mail ad utenti od amministratori in caso di eventi superanti un determinato grado di pericolosità. In ognuno di questo log vengono fornite tutte le informazioni utili riguardanti l'evento, come ad esempio il nome dell'agente, l'ip dell'host monitorato,

l'ora alla quale è avvenuto e la descrizione completa del log. Ci sono anche casi dove l'evento è generato da IP esterni all'host ospitante l'agente, come nel caso delle connessioni *ssh*. In questi casi il log ci fornirà informazioni riguardanti l'ip che ha tentato di creare una connessione con la nostra macchina e queste solitamente comprendono IP che ha richiesto la connessione, ubicazione geografica dell'host richiedente e anche la tipologia di tattica e di tecnica catalogata da *Mitre*. Queste ultime ovviamente sono informazioni riguardanti perlopiù attacchi al nostro host ma in altri casi sono connessioni sicure verso studenti o utenti in remoto utilizzanti la macchina.

Un altro modulo legato sempre dai log è chiamato Integrity Monitoring Module e la tipologia di log inviati sono quelli riguardanti il cambiamento del filesystem . E' possibile vedere qualunque azione che viene compiuta su un qualsiasi file all'interno della macchina monitorata dando così informazioni estremamente utili in caso di attacco. La tipologia di questi log si differenzia in 3 tipi:

1. **modified:** In questa tipologia ci vengono fornite informazioni che riguardano quale modifica è stata apportata all'interno del file. Viene esplicato cosa è stato modificato, se la dimensione del file è variata e anche se l'hash relativo al file ha subito dei cambiamenti. Grazie al support di Mitre vengono fornite informazioni relative alla tipologia di tecniche e tattiche utilizzate per la determinata modifica, verrà inoltre fornito anche il timestamp relativo.

2. **added:** Nel caso di aggiunta di un file viene fornito il path nella quale è stato creato. Inoltre il log ci descrive anche che cosa è stato aggiunto (una chiave di registro, un file di testo etc...) e a quale orario l'operazione è stata eseguita.

3. **deleted:** Quando si ha un evento di eliminazione il log ci descriverà cosa è stato eliminato e in che percorso questo evento è stato rilevato. Anche in questa situazione ci verranno fornite le tipologie di tattiche e di tecniche che Mitre identifica sull'evento verificatosi. Verrà fornito inoltre anche il timestamp relativo.

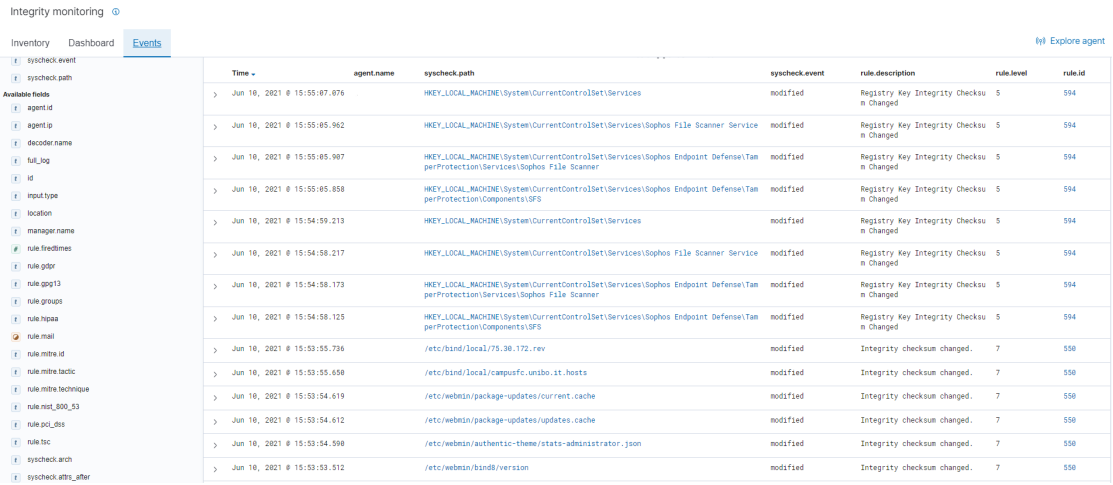

Figure 13: Dashboard per Integrity Monitoring.

La suddivisione in gruppi data dalla configurazione centralizzata risulta fondamentale in ognuna di queste situazioni permettendoci di differenziare quali log vogliamo che siano inviati al Manager dai vari agenti.

5 Test e risultati

In questo capitolo verranno trattate le minacce che sono state rivolte a qualcuno dei server monitorati. Soprattutto saranno mostrate situazioni di brute-force e alert particolari trovati durante l'analisi di log, con in aggiunta il numero di log derivanti da ogni tipo di server in nostro possesso.

5.1 Attacco Brute Force

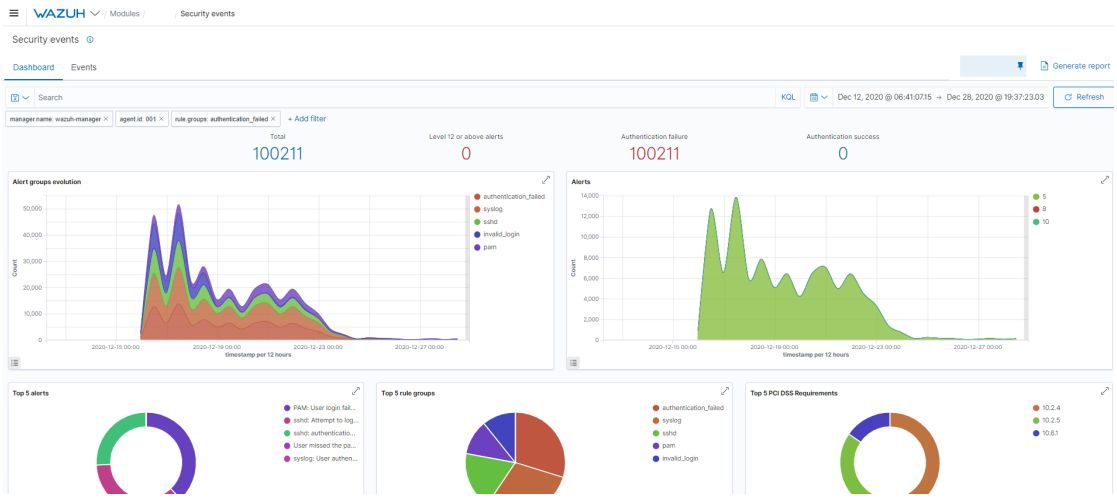

Figure 14: Grafico riguardante il brute force SSH.

Nell'immagine sovrastante è possibile osservare svariati picchi di attività. Questo grafico mostra la quantità di log mandata dall'agent al manager, solitamente questa si aggira intorno alle 1600 1700 unità giornaliere, mentre in questo caso il numero è arrivato a toccare i 130211 log nel giro di mezza giornata. Essendo log relativi a connessioni SSH derivanti tutti dall'esterno si è giunto alla conclusioni di essere sotto attacco di tipo Brute Force.

In primis si è pensato a fermare questa attività utilizzando il software Fail2Ban, che permette di scansionare i vari log e bannare gli ip responsabili di attività malevola o che hanno sbagliato troppi tentativi di login. Grazie a questo è stato possibile ridurre il numero di login ma non ad eliminarli in toto, si può notare infatti nel grafico come ci sia un decremento dell'attività ma non un completo shutdown. Per azzerare completamente il Brute Force è stato necessario inserire delle regole all'interno del firewall in grado di poter effettivamente chiudere ogni tentativo di connessione da paesi extra europei, visto che i tentativi derivavano per la maggior parte da questi.

Dopo questo procedimento i tentativi di accesso sono stati completamente fermati, facendo rimanere effettivamente solo quelli che interessavano il corpo studentesco o utenti affiliati. A volte si riscontra tutt'ora qualche attività di questo tipo ma di entità molto più piccola.

5.2 DNS Query Chace Denied

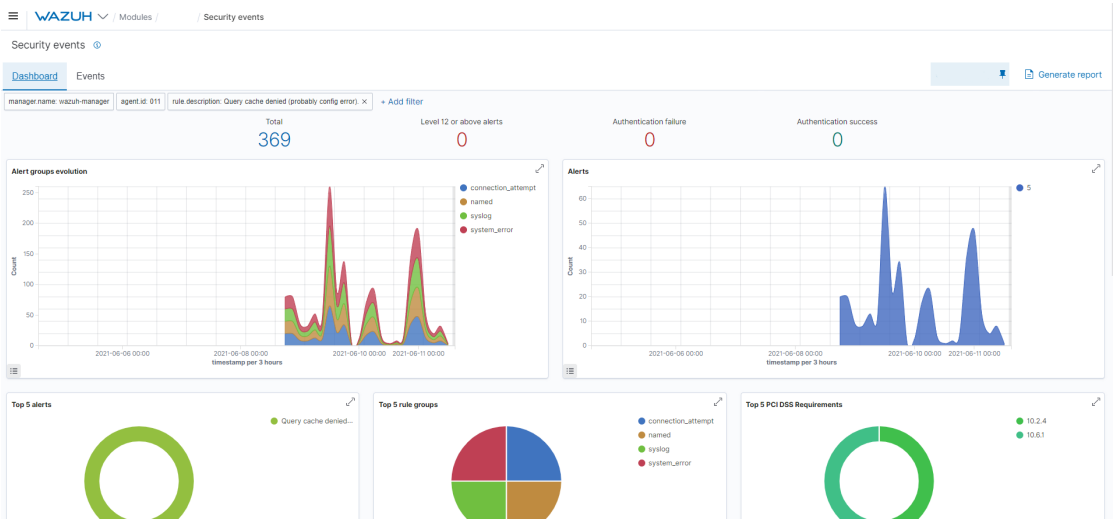

Figure 15: Grafico rigurdante richieste DNS.

In questo caso invece prendiamo in analisi il problemi delle Query Cache Denied. Si presenta quando il server DNS proprietario di unibo viene utilizzato per risolvere richieste DNS verso domini non di proprietà. Le query quindi vengono scartate e non vengono eseguite. Il server DNS potrebbe anche essere soggetto a DNS cache poisoning dove viene inserito un indirizzo corrotto all'interno della cache reindirizzando poi le richieste effettuate verranno reindirizzate verso la destinazione sbagliato (solitamente un server in possesso dell'attaccante).

5.3 Spam mail

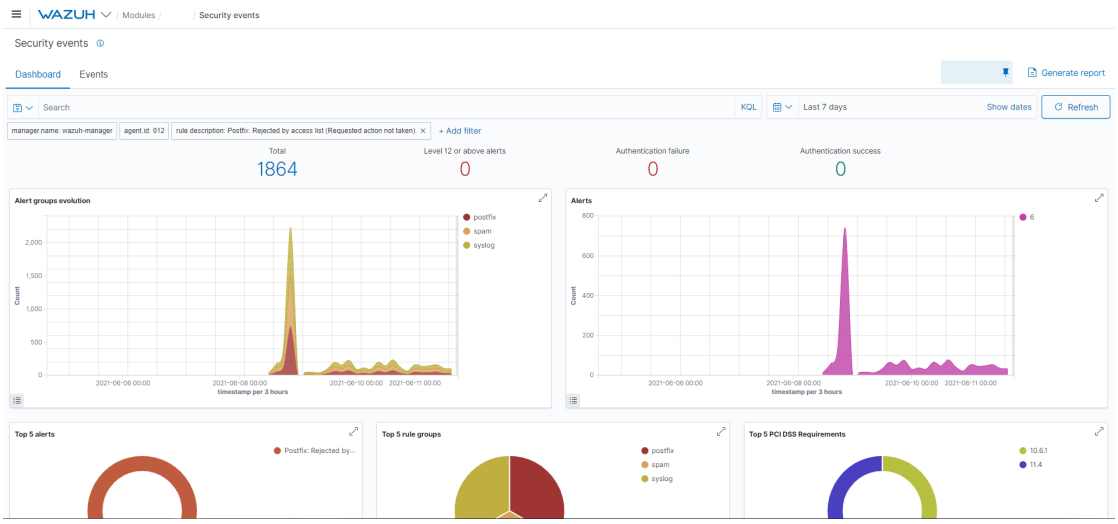

Figure 16: Grafico rigurdante spam di mail.

Lo spam di mail è un'altra situazione abbastanza comune. Si verifica quando un indirizzo mail non verificato prova ad inviare messaggi a profili interni ad Unibo. Un'altra situazione analoga ha luogo anche quando mail valide vengono inviate a profili non più esistenti. Wazuh inoltre è anche in grado di bloccare autonomamente gli IP invianti mail spam.

5.4 Log settimanali per tipi di server

In questa sezione verranno allegati grafici con informazioni relativi all'attività degli agenti Wazuh nell'arco di una settimana.

5.4.1 Dati per server SSH

Nel server SSH come si può notare abbiamo un grande numero di accessi, perlopiù falliti, se questi eventi aumentano in maniera sostanziale potrebbe essere conseguenza di un attacco Brute Force.

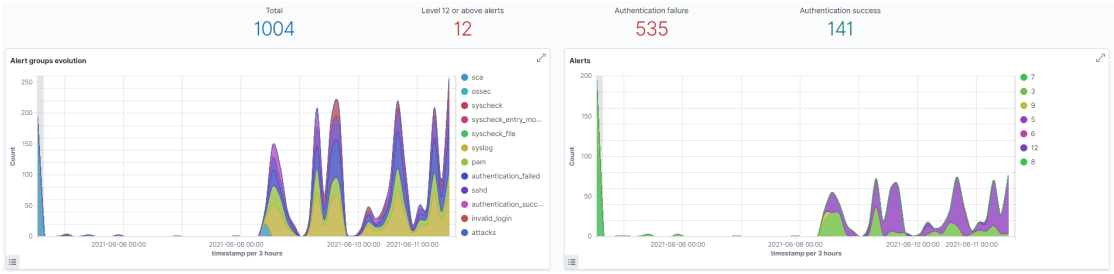

Figure 17: Grafico riguardante eventi SSH.

5.4.2 Dati per server DNS

In color ocra è possibile vedere tutte le richieste DNS verso il server in nostro possesso. Molte di queste vengono scartate per via di richieste verso domini non proprietari.

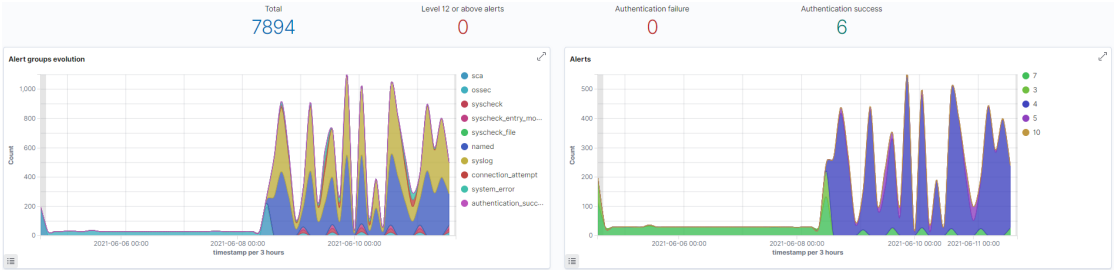

Figure 18: Grafico riguardante eventi DNS.

5.4.3 Dati per server Wordpress

Il server Wordpress riceve richieste di connessione di tipo PAM oppure genera errori di tipo 400. Nel caso di questi errori solitamente si tratta di interrogazioni da parte di IP esterni per capire se il server in nostro possesso possiede tipi specifici di vulnerabilità.

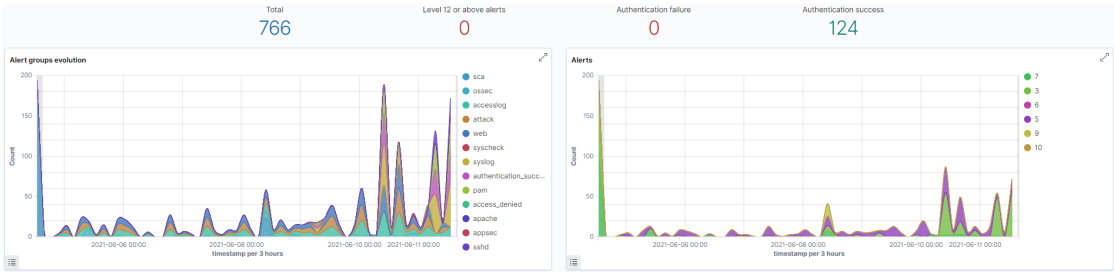

Figure 19: Grafico riguardante server Wordpress.

5.4.4 Dati per server mailer

Nel server mailer invece è subito visibile l'alto numero di autenticazioni fallite. Questo è dato solitamente dal inserimento errato delle credenziali di accesso. Il numero di eventi è anche incrementato da tutte le mail spam inviate agli utenti, questi IP vengono poi prontamente bloccati, ma ciononostante sono sempre numerose.

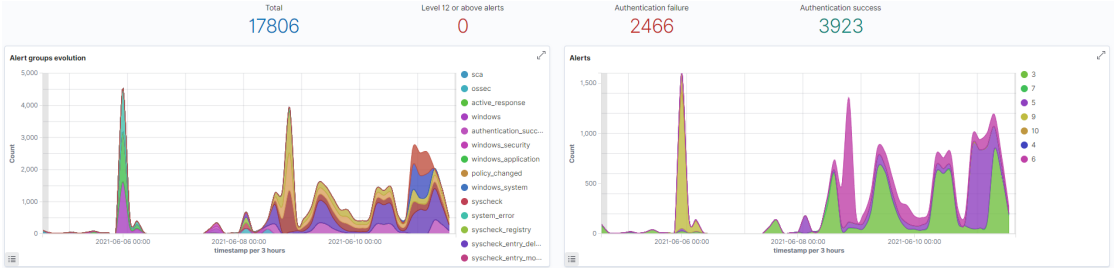

Figure 20: Grafico riguardante server Mailer.

6 Conclusioni e migliorie

6.1 Conclusioni

Attualmente gli agenti Wazuh sono stati installati su 16 server ritenuti importanti e sono controllati quotidianamente da chi di dovere. Questo permette di elevare il livello di sicurezza di questi server ancora di più, riuscendo quindi ad individuare eventuali attacchi di qualsivoglia genere. Ad esempio grazie a Wazuh è stato possibile rilevare attacchi diretti ai server SSH e attivare poi le contromisure necessarie sul firewall per chiudere le connessioni causanti anomalie. Nonostante questo possono essere segnalati anche falsi positivi che potrebbero causare allarmismi immotivati.
Per via della pandemia i vari agenti non sono stati sfruttati a pieno regime rendendo molto complesso capire il comportamento e le prestazioni del Wazuh Manager con il carico di lavoro generato in un periodo di grande intensità come quello delle lezioni.

Durante periodi di media attività il Wazuh Manager riceve circa 360.000 messaggi a settimana provenienti dai vari agenti. Durante il periodo di sessione estiva si è invece osservata un'impennata nel numero di pacchetti in ingresso arrivando addirittura a quasi 600.000 settimanali. Questo probabilmente è stato lo stress test migliore che abbiamo potuto sperimentare.

6.2 Possibili migliorie

Per rendere più efficiente il sistema si potrebbe pensare di decentralizzare il manager in più nodi, in modo da alleggerire il carico di lavoro della macchina preposta al lavoro di manager.
Inoltre è necessario anche adattare le regole inserite all'interno di Wazuh in modo tale da far registrare solo eventi effettivamente malevoli, eliminando quindi il problema dei falsi positivi.
Un'altra miglioria applicabile sarebbe quella di rendere automatica la risposta di Wazuh ogni qualvolta si verifichino determinati tipi di eventi. Questo quindi trasformerebbe Wazuh anche in un IPS (Intrusion Prevention System), agendo quindi in maniera autonoma per eliminare la minacce.

Per aumentare ulteriormente la sicurezza delle macchine con un agente Wazuh si potrebbe anche pensare di installare Security Onion. QUesto tool sfrutta svariati IDS per rilevare minacce provenienti dall'esterno ma, in caso di attacco riuscito non sarebbe in grado di fornirci informazioni riguardanti l'evento avvenuto. Ma in combinazione con Wazuh questa debolezza verrebbe meno, rendendo quindi il sistema ancora più robusto.

La parte di hardening può essere ulteriormente approfondita leggendo i vari SCA report redatti da Wazuh. A volte questi possono comprendere operazioni futili, non necessarie quindi a irrobustire la difesa della macchina. Un analisi più accurata dei report può quindi ottimizzare questa fase rendendola più efficace.

7 Ringraziamenti

Volevo ringraziare in primis il dottor Ciro Barbone, correlatore del progetto e grande punto di riferimento per me durante tutto lo svolgimento. Ringrazio poi il prof. Vittorio Ghini per avermi permesso di conoscere ambiti dell'informatica a me sconosciuti e alla quale ora sono profondamente interessato.

Infine voglio ringraziare enormemente la mia famiglia per avermi supportato in ogni modo in questo percorso e aiutandomi a non mollare anche nei momenti più difficili.

References

[AO 21] AO Kaspersky Lab. *Definitions*. https://www.kaspersky.com/resource-center/definitions. [Online]. 2021.

[Bar21] Barracuda Networks Inc. *Glossary*. https://www.barracuda.com/glossary/. [Online]. 2021.

[Clo21] Cloudflare Inc. *Learning Center*. https://www.cloudflare.com/learning/. [Online]. 2021.

[Doc21] Docker Inc. *Docker Docs*. https://docs.docker.com/. [Online]. 2021.

[Ela21] Elasticsearch B.V. *Elastic Stack and Product Documentation*. https://www.elastic.co/guide/i [Online]. 2021.

[Lin21] Linux Containers. *Introduction*. https://linuxcontainers.org/lxc/introduction/. [Online]. 2021.

[Nex21] Nexsys. *Digital Forensic*. https://www.nexsys.it/blog/digital-forensics-cos-e-l-analisi-forense/. [Online]. 2021.

[Ope16] Open Information Security Foundation. *Suricata User Guide*. https://suricata.readthedocs.io/e 6.0.3/. [Online]. 2016.

[Red21] Red Hat Inc. *What is SELinux?* https://www.redhat.com/it/topics/linux/what-is-selinux. [Online]. 2021.

[The21a] The Linux Foundation. *Overview*. https://kubernetes.io/docs/concepts/overview/. [Online]. 2021.

[The21b] The MITRE Corporation. *Common Vulnerability Enumeration*. https://cve.mitre.org/. [Online]. 2021.

[The21c] The MITRE Corporation. *Common Weakness Enumeration*. https://cwe.mitre.org/. [Online]. 2021.

[The21d] The MITRE Corporation. *MITRE ATTCK*. https://attack.mitre.org/. [Online]. 2021.

[Waz21] Wazuh Inc. *Getting started*. https://documentation.wazuh.com/current/getting-started/index.html. [Online]. 2021.