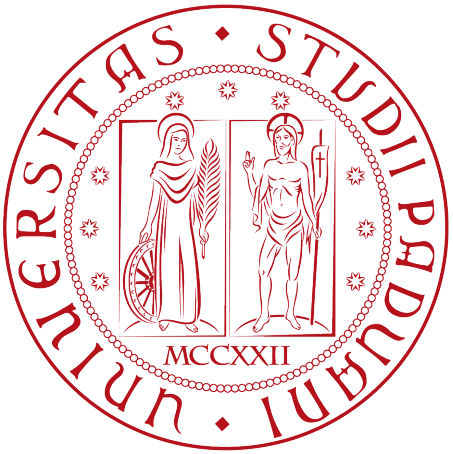
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Sistemi di Log Monitoring e Security Management

Tesi di laurea triennale

Relatore

Prof.ssa Ombretta Gaggi

Laureando

Gianluca Ferrari

Anno Accademico 2021-2022

Abstract

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Ferrari Gianluca presso l'azienda Athesys s.r.l. di Padova.
Lo scopo principale del progetto consiste nell'effettuare una ricerca sulle principali tecnologie sull'ambito di Log Monitoring e Security Information and Event Management (SIEM), dando priorità alle soluzioni opensource, per scegliere quelle più idonee da testare.
Successivamente è stato effettuato il deploy delle soluzioni scelte per testarne le funzionalità principali e valutare al meglio pregi e difetti.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine alla Prof.ssa Ombretta Gaggi, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Desidero infine ringraziare i miei amici per l'appoggio e l'incitamento offertomi.

Padova, Dicembre 2022 Gianluca Ferrari

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

In questo capitolo verranno riportate le informazioni riguardanti l'azienda ospitante ed una breve sintesi del progetto affrontato..

1.1 L'azienda

Figura 1.1: Logo Athesys s.r.l.

Il tirocinio è stato svolto presso l'azienda **Athesys s.r.l.** (logo in figura 1.1) di Padova.
L'azienda (nata nel 2010) si occupa principalmente di consulenza nei settori di Cloud, Database Management, Identity and Access Management e Sviluppo Software. Nel 2017 fonda la startup `Monokee`, dove è stato spostata la parte più incentrata sullo sviluppo, lasciando su `Athesys` la parte più consulenziale e di system integration.
Monokee offre un sistema di Identity and Access management ibrido, che gestisce le identità digitali sia in modo centralizzato che decentralizato.

1.2 Il progetto

Il progetto nasce dalla necessità di introdurre nelle tecnologie aziendali un sistema automatizzato di monitoraggio dei log ed un sistema di sicurezza Security Information and Event Management (SIEM).
In un file di log vengono registrate tutte le operazioni, in ordine cronologico, svolte nel normale utilizzo di un software, di un applicativo o più semplicemente di un computer. I file di log registrano anche le operazioni che un computer svolge in autonomia, comprendendo tutte le informazioni sul normale funzionamento della macchina e, soprattutto, le registrazioni di errori e problemi.
Un software di monitoraggio raccoglie log da diverse fonti (applicazioni, log di sistema, log di sicurezza) e li gestisce in un unico sistema centralizzato che permette di avere

una panoramica dell'intero ambiente.
Il monitoraggio del log permette dunque di identificare attività di interesse come accessi, anomalie, fallimenti e possibili minacce malware, garantendo una rapida identificazione dei problemi e di conseguenza una loro efficiente risoluzione.
Un sistema SIEM ha come obiettivo la raccolta centralizzata dei log e degli eventi generati da applicazioni e sistemi in rete, per consentire analisi di sicurezza in tempi brevi.
Permette di raccogliere, analizzare e correlare dati provenienti da diverse fonti, come ad esempio dispositivi di rete (router, switch), applicazioni, strumenti di sicurezza.
Il sistema SIEM effettua una correlazione tra i dati raccolti ed una serie di regole che descrivono comportamenti anomali o malevoli, generando di conseguenza alert che possono essere valutati per l'investigazione del problema e la sua risoluzione.
Lo scopo del progetto è dunque identificare le varie soluzioni opensource disponibili in ambito di log monitoring e SIEM, per scegliere le più adatte da installare e testare.
Lo stage è stato suddiviso in tre periodi principali, di seguito descritti.

- Primo periodo focalizzato sullo studio della documentazione ufficiale dei sistemi di monitoraggio dei log e SIEM, per scegliere qali tecnologie utilizzare nella parte pratica.
- Secondo periodo focalizzato sul deploy delle tecnologie scelte e sulla loro configurazione.
- Terzo periodo utilizzato per effettuare una ricerca sulla possibilità di integrare la soluzione SIEM scelta con degli strumenti di Machine Learning (ML) e con un Security Operation Center (SOC).

1.3 Organizzazione del testo

Il secondo capitolo descrive lo studio effettuato sulle varie soluzioni disponibili negli ambiti di monitoraggio dei log e SIEM. Al termine si mostrano i sistemi che sono stati scelti per essere installati e testati.

Il terzo capitolo approfondisce il lavoro svolto utilizzando il software di log monitoring Prometheus, la sua installazione e configurazione ed una panoramica delle possibità che esso offre.

Il quarto capitolo approfondisce il lavoro svolto utilizzando lo stack Elastic, la sua installazione e configurazione ed una panoramica delle possibità che esso offre.

Il quinto capitolo approfondisce il lavoro svolto utilizzando il sistema SIEM Wazuh, la sua installazione e configurazione, l'integrazione con softtware di intrusion detection, ed una panoramica delle possibilità che esso offre.

Il sesto capitolo descrive la possibile integrazione di strumenti di Machine Learning a supporto dei sistemi SIEM.

Il settimo capitolo descrive la possibile integrazione di un sistema SIEM in un Security Operation Center.

L'ottavo capitolo illustra le conclusioni relative allo stage.

Capitolo 2

Analisi delle tecnologie

In questo capitolo vengono analizzate le varie soluzioni di Log Monitoring e SIEM, illustrando le peculiarità di ogni prodotto.

2.1 Log Monitoring

Di seguito si valutano le tecnologie principali utizlizzate per monitoraggio e gestione dei log.

2.1.1 Prometheus

Prometheus è un software opensource di monitoraggio dei log sviluppato da Soundcloud in linguaggio Go.
Prometheus colleziona **metriche**, ovvero valori numerici che misurano caratteristiche di interesse (as esempio: utilizzo della CPU) associati ad un timestamp e ad un nome che identifica la metrica. Le metriche collezionate vengono salvate su un Time Series Database (TSDB).
Le metriche collezionate si suddividono in quattro tipologie:

- **Counter**: dato numerico il cui valore può solo aumentare od essere resettato a zero.
- **Gauge**: dato numerico il cui valore può aumentare o diminuire.
- **Histogram**: raccoglie osservazioni e le conta su bucket configurabili.
- **Summary**: offre le stesse caratteristiche degli *Histogram*, con in più la possibilità di calcolare quantili.

Le metriche raccolte possono essere ricercate nel sistema usando un apposito linguaggio di query detto Prometheus Query Language (PromQL). Prometheus è composto da uno stack di componenti che interagiscono tra loro:

- **Prometheus Server**: è il componente principale; raccoglie le metriche dai sistemi da monitorare e le salva nel TSDB. Gestisce inoltre le regole degli *alert* ed espone i dati raccolti in una dashboard.

- **Alertmanager**: componente che gestisce l'invio degli *alert* su vari sistemi configurabili (ad esempio: email, canali Slack).

- **esporter**: componente che si occupa della raccolta delle metriche e della loro esposizione al *Prometheus Server*. Esistono molti tipi di exporter, specializzati in un sistema specifico da monitorare (ad esempio: database, container, componenti hardware). Alcuni exporter sono sviluppati internamente al progetto Prometheus, altri sono sviluppati e mantenuti esternamente.

- **Client Libraries**: librerie che permettono di creare ed esporre metriche nelle applicazioni. Alcune librerie sono sviluppate internamente al progetto Prometheus (per i linguaggi Go, Java, Python, Ruby, Rust), ma ne esistono molte sviluppate da varie community.

Il database di Prometheus è pensato per una breve ritenzione delle metriche: per uno storage a lungo termine si consiglia l'utilizzo di altri database compatibili, come ad esempio *VictoriaMetrics* o M3DB.
Scalabilità: Prometheus offre un sistema nativo di scalabilità abbastanza limitato, utilizzando diversi nodi su cui gira un *Prometheus Server*. In caso di necessità di un sistema distribuito su larga scala, esistono dei software opensource esterni creati appositamente per questo: il più utilizzato è il software **Thanos**.

2.1.2 Elastic Stack

Lo stack Elastic è un insieme di tecnologie utilizzato per la raccolta e la visualizzazione di log. A differenza di Prometheus i log vengono salvati come documenti, permettendo in questo modo il monitoraggio di qualsiasi tipo di log e non solo quelli con valori numerici. Inizialmente rilasciato con una licenza *Apache 2.0*, ora è riasciato con una licenza Server Side Public License (SSPL).[1]
Elastic è inoltre disponibile in varie versioni a pagamento, che aggiungono caratteristiche non presenti nella versione base.
Lo stack Elastic è composto da quattro componenti:

- **Beats**: i Beats sono i *data shipper* utilizzati da Elastic per raccogliere i log di interesse. Sono diversificati in base al tipo di dato che devono monitorare; al momento sono disponibili: Filebeat, Metricbeat, Packetbeat, Winlogbeat, Auditbeat, Heartbeat, Functionbeat.

- **Logstash**: modulo che si occupa di ricevere i dati dai Beats e di effettuare operazioni di enrichment, parsing, filtering e vari tipi di normalizzazione e di rendere i dati trattati disponibili al modulo Elasticsearch. Logstash gira su Java Virtual Machine (JVM), ed è importante configurare l'ambiente per evitare problemi di prestazioni.

- **Elasticsearch**: motore di ricerca e database su cui sono salvati tutti i dati raccolti dai *data shipper*. Si basa sul database opensource *Apache Lucene*. I dati vengono divisi ed organizzati su indici logici. Anche Elasticsearch gira su JVM, ed anche in questo caso è consigliato configurare l'ambiente per prevenire problemi di prestazioni.

[1] Seacom.it - *Facciamo chiarezza sulla questione delle licenze Elastic*
https://www.seacom.it/facciamo-chiarezza-sulla-questione-delle-licenze-elastic/

- **Kibana**: strumento di visualizzazione dei dati. Può essere utilizzato con delle dashboard standard o con dashboard create dall'utente; permette inoltre di creare vari tipi di grafici e tabelle.

Elastic offre una collezione di librerie Client per integrarsi con vari linguaggi: al momento esistono librerie per Java, Javascript, Ruby, Go, Perl, Python, Rust e molte altre sviluppate da varie community.
Un sistema di alerting non è presente nella versione base di Elastic, ma è disponibile nelle versioni a pagamento. Tuttavia, esistono dei software opensource sviluppati da varie community che possono essere integrati con Elasticsearch per svolgere questa funzione.
Lo stack Elastic è stato pensato sin dall'inizio del suo sviluppo per essere un sistema scalabile, senza bisogno di software di terze parti; si può effettuare il deploy di un *cluster* di nodi Elasticsearch, facilmente configurabile ed estendibile in caso di necessità con l'aggiunta di altri nodi.

2.1.3 Graphite

Graphite è un sistema di monitaraggio che raccoglie **metriche** numeriche come *time series* e le salva su un TSDB. Le metriche possono essere poi visualizzate su vari tipi di grafici.
Graphite è strutturato in tre componenti principali:

- **Carbon**: è un servizio che riceve le metriche per renderle disponibili agli altri componenti del sistema.
- **Whisper**: è un TSDB utilizzato per salvare le informazioni raccolte.
- **Graphite Web UI**: interfaccia di visualizzazione dati; può generare diversi tipi di grafici per l'osservazione delle metriche raccolte.

Graphite richiede che il client da monitorare invii le metriche a *Carbon*. Esistono diversi metodi per inviare dati:

- **Protocolli**: Carbon supporta diversi tipi di protocolli come plaintext e Advanced Message Queuing Protocol (AMPQ).
- **Tools**: Carbon è compatibile con diversi tool che si occupano di raccogliere ed esporre dati dai client, come ad esempio *StatsD* o *collectd*.

Generalmente si considera la Graphite Web UI non eccellente nel creare dashboard personalizzate che diano una visuale completa dei sistemi monitorati. Viene integrata con sistemi esterni di visualizzazione dei dati come *Grafana*.
Graphite non dispone di un sistema di alerting nativo. Tuttavia è possibile l'integrazione con tool esterni che permettono di generare alert, come ad esempio *graphite-beacon* o *Moira*.

2.1.4 Sensu

Sensu è un sistema di monitoraggio dei log bassato su *agent* da installare nei sistemi da monitorare.
Sensu è free fino al limite di 100 nodi monitorati, ed offre soluzioni a pagamento per sistemi più estesi.

Gli agent osservano metriche su server e app di vario genere (AWS, Docker, Kubernetes, Prometheus, StatsD, OpenShift), e producono *events*. Gli *events* sono contenitori generici che Sensu usa per dare un contesto al dato; di seguito un esempio della struttura di un event:

```
{
  "metadata": {},
  "entity": {},
  "check": {},
  "metrics": {},
  "id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "timestamp": 1234567890
}
```

Sensu è provvisto di un database chiave-valore *etcd* per poter salvare sia i dati che la configurazione del sistema stesso; è possibile configurare il sistema per usare un cluster di nodi *etcd* o per salvare i dati su altri tipi di database esterni (ad esempio *PostgreSQL*).
Sensu è provvisto di un repository chiamato *Bonsai* ove sono disponibili gli agent per i diversi sistemi da monitorare, oltre che a vari tipi di plugin free o a pagamanento.
Il sistema può essere configurato impostando dei filtri per valutare eventi critici e lanciare degli alert al loro verificarsi.
È inoltre possibile integrare Sensu con sistemi esterni di visualizzazione dei dati, come ad esempio *Grafana* o *Kibana*.

2.1.5 Nagios Core

Nagios Core è un software opensource per il monitoraggio di computer e risorse di rete. La sua funzione base è quella di controllare nodi, reti e servizi specificati, avvertendo con degli alert quando questi non garantiscono il loro servizio o quando ritornano attivi.
Esiste la versione a pagamento denominata Nagios XI.
La versione Core offre funzionalità limitate rispetto alla sua controparte a pagamento.
La soluzione Nagios Core è strutturata in tre componenti principali:

- **Plugins**: sono gli agenti da installare nei sistemi da monitorare: esistono numerosi tipi di plugin specifici per ogni tipologia di sistema.
- **Process Scheduler**: controlla i *Plugins* ad intervalli regolari ed agisce in base al tipo di evento rilevato; ad esempio può essere configurato per inviare alert o eseguire script.
- **Nagios GUI**: interfaccia grafica del sistema.

Nagios Core non utilizza un database per salvare i dati; è possibile tuttavia utilizzare il plugin *NDOUtils* per esportare i dati in un database Mysql.

2.1.6 Icinga

Icinga è un sistema di monitoraggio opensource nato nel 2009 come fork di Nagios.
Icinga può esere utilizzato per monitorare la disponibilità di host e servizi, per verificare se sono attivi o no: possono essere monitorati servizi di rete (come HTTP, SMTP), stampanti, routers, sensori ed in genere qualsiasi sistema accessibile in rete.
Inoltre Icinga dispone di *agent* da installare nei sistemi da monitorare (ad esempio:

Icinga Agent).
Per lo stoccaggio dei dati, Icinga utilizza un sistema formato da diversi componenti:

- **icingadb**: invia i dati collezionati dal sistema ad un server Redis.
- **database**: Ichinaga conserva i dati collezionati in un database, tipicamente Mysql o PostreSQL.
- **Icinga DB daemon**: servizio che si occupa di sincronizzare i dati tra il database ed il server Redis.
- **Icinga DB Web**: modulo che si connette sia al database che al server Redis per far visualizzare i dati.

Icinga può essere integrato con molti altri sistemi; di seguito alcuni esempi di integrazioni utili:

- export dei dati raccolti a *Graphite*.
- export dei dati raccolti ad *Elasticsearch* (utilizzando il plugin *Ichingabeat*).
- visualizzazione dei dati su *Grafana*.
- utilizzo dei plugin di *Nagios*.

2.1.7 Scelta dei sistemi da installare

Valutando le varie caratteristiche dei sistemi considerati, si è deciso di procedere con l'effettivo deploy di **Prometheus** e di **Elastic Stack**. Si è considerata principalmente la possibilità di effettuare una scrittura dei dati su un database esterno, la possibilità di integrazione con altri sistemi e soprattutto la chiarezza della documentazione per quanto riguarda l'installazione e la configurazione del software. Di seguito una tabella comparativa che illustra i pro e contro dei vari sistemi.

	DB interno	DB remoto	Sistema Alert	Integrazione altri sistemi	Documenti chiari	Scalabilità	Licenza
Prometheus	✓	✓	✓	✓	✓	✓	opensouce
Elastic Stack	✓	✗[1]	✓	✓	✓	✓	SSPL
Graphite	✓	✗[2]	✗[3]	✓	✓	✓	opensource
Sensu[4]	✓	✓	✓	✓	✓	✓[5]	opensource commercial
Icinga	✓	✓	✓	✓	✗	✓	opensource
Nagios Core	✗	✓[6]	✓	✗	✗	✗	opensource

Tabella 2.1: Tabella Comparativa

[1] No DB esterno ma si aumenta la capacità aumentando i nodi del sistema.
[2] No DB esterno, ma si può usare un DB alternativo a quello standard.
[3] Si può aggiungere con Moira-alerts
[4] Free fino a 100 nodi.
[5] Sensu etcd cluster
[6] con plugin NDOUtils.

2.2 SIEM

Di seguito si valutano le tecnologie principali utilizzate come sistema SIEM.

2.2.1 OSSEC

OSSEC è un sistema SIEM che prevede un'opzione opensource (OSSEC) ed una a pagamento (Atomic OSSEC).
Il sistema OSSEC è composto principalmente da due componenti:

- **Manager**: controlla tutti i dati ricevuti dai sistemi da monitorare, gestisce il database, gestisce tutte le opzioni di configurazione. Di default gli agenti comunicano con il Manager tramite la porta 1514/udp.
- **Agent**: programmi installati nei sistemi da monitorare, che raccolgono dati e li inviano al *Manager*. Il monitoraggio può essere real-time o periodico, in base al tipo di informazione da raccogliere. Sono sviluppati per avere un consumo estremamente ridotto su memoria e CPU.

Su sistemi in cui non è possibile installare un agent (ad esempio: firewall, router) è possibile avere un monitoraggio *agentless*.
Utilizza database MySQL o PostgreSQL.
È provvisto di un sistema di alerting customizzabile.
La versione opensource offre molte meno funzionalità di quella Atomic: non è possibile ad esempio fare scan di vulnerabilità, configurare active response, e molte altre caratteristiche sono assenti o limitate.

2.2.2 Wazuh

Wazuh è un sistema opensource che combina capacità SIEM e eXtended Detection and Response (XDR). È basato su OSSEC. È disponibile una versione a pagamento denominata *Wazuh Cloud*, che semplifica il deploy e la configurazione.
Wazuh è strutturato dai seguenti componenti:

- **Wazuh-manager**: è il componente che si occupa di ricevere i dati dagli endpoint da monitorare, per analizzarli e cercare indicatori di compromissione del sistema od eventuali vulnerabilità riscontrate. Si occupa inoltre di gestire gli *agent* installati sugli endpoint.
- **Wazuh-indexer**: motore di ricerca e di analisi. Salva e gestisce gli *alert* generati dal Wazih-manger, offrendo la possibilità di ricerca ed analisi su di essi.
- **Wazuh-dashboard**: sistema di visualizzazione dei dati.
- **Agent**: gli agent vegono installati negli endpoint da monitorare.

Su sistemi in cui non è possibile installare un agent (ad esempio: firewall, router) è possibile avere un monitoraggio *agentless*.
Il protocollo di Wazuh usa crittografia AES con blocchi da 128 bits e chiave a 256 bits (con crittografia Blowfish opzionale). Il manager usa *FileBeat* (di Elastic) per inviare dati ed alert all'indexer usado crittografia TLS su porta 9200/TCP.
Dopo che l' indexer ha indicizzato i dati, la dashboard li visualizza.
Gli eventi vengono salvati sul server Wazuh in formato JSON oppure in plain-text

(formato .log), e vengono giornalmente compressi e firmati (MD5, SHA1, SHA256).
Wazuh può essere usato per soddisfare requisiti di *compliance* (PCI DSS, HI-PAA), monitorare servizi IaaS (AWS, Azure), macchine virtuali e container.
Offre la possibilità di esportare gli alert in un database esterno (MySQL o PostgeSQL) e può essere integrato con Elasticsearch.
È possibile configurare Wazuh per inoltrare gli alert di interesse tramite vari canali (ad esempio via email o Slack).
La scalabilità si ottiene con il deploy di un *cluster* di nodi, editando nella configurazione quale è il nodo Master e quali i nodi Worker.

2.2.3 AlienVault OSSIM

AlienVault OSSIM è la versione opensource del software AlienVault USM.
Analizza log da diverse fonti, quali database, syslog, WMI, ed il traffico di rete.
OSSIM utilizza la seguente terminologia per identificare gli endpoint da monitorare:

- **Asset**: ogni dispositivo dotato di un indirizzo IP.
- **Data Source**: ogni *asset* che crea ed invia log.

OSSIM effettua un'analisi degli indici di compromissione tramite l'identificazione degli asset di interesse e l'analisi delle porte comunemente usate, oltre che a delle scansioni effettuate utilizzando il software OpenVAS e l'identificazione di anomalie nel traffico di rete.
OSSIM è strutturato in tre componenti:

- **Server**: elabora gli eventi ed applica i modelli per trovare anomalie.
- **Sensor**: raccoglie i log e li elabora in eventi da inviare al Server.
- **Logger**: salva, comprime e firma i log.

Nei sistemi ove non è possibile installare un agent, fornisce monitoraggio *agentless* per router, switch, firewall e dispositivi di rete.
Offre la possibilità di inviare alert di interesse (ad esempio via email).

2.2.4 Prelude OSS

Prelude OSS è la versione opensource di Prelude SIEM.
È strutturato in sei componenti:

- **Prelude Manager**: server che accetta le connessioni dai sensori o da altre istanze Manager, e salva gli alert in un sistema deciso dall'utente (database , log file...). È provvisto di plugin di filtraggio dei log e di reporting (ad esempio: invio mail con descrizione di un alert).
- **Libprelude**: libreria che garantisce una connessione sicura tra i sensori ed il Manager.
- **LibpreludeDB**: fornisce un'astrazione sul tipo usato per salvare gli alert, rendendo l'accesso al database indipendente dal suo formato/tipo. Il database deve essere installato a parte. Prelude è compatibile con MySQL, PostgreSQL, Sqlite.

- **Prelude-LML**: permette di analizzare i log ricevuti per scovare attività sospette e di generare alert. È configurabile e compatibile con molti altri sistemi (ad esempio: AuditD, Samhain, Suricata).
- **Prelude-Correlator**: motore di correlazione basato su regole scritto in Python. Se trova una correlazione, genera un alert.
- **Prewikka**: interfaccia grafica usata per visualizzare gli alert generati da Prelude.

La versione opensource è da considerarsi adatta al monitoraggio di sistemi ridotti e per testing.

2.2.5 Security Onion

Security Onion è una soluzione SIEM opensource con capacità di monitoraggio di sicurezza, analisi delle minacce e gestione dei log.
Si integra con diversi sistemi opensouce di monitoraggio, come Suricata, Stenograph e Strelka, offrendo una visione di insieme di possibili anomalie ed indicatori di compromissione.
Può inoltre monitorare gli endpoint di interesse utilizzando gli *agent* di altri sistemi, ad esepio quelli di Wazuh ed i Beats di Elastic.
La visualizzazione e l'analisi dei dati viene gestita su un'interfaccia grafica denominata **Security Onion Console**.
La Security Onion Console è provvista delle seguenti interfacce:

- **Alerts**: visualizza tutti gli alert generati. È possibile raggruppare gli alert usando regole definite dagli utenti.
- **Dashboards**: contiene un set di dashboard predefinite per visualizzare i dati.
- **Hunt**: interfaccia specifica per operazioni di *Threat Hunting*.
- **Cases**: si possono aggregare eventi e misurazioni che potrebbero indicare una compromissione in un *Caso* a cui sarà assegnato nome, descrizione e potenziale livello di minaccia.
- **CyberChef**: tool che permette di manipolare i dati effettuando svariati i tipi di conversioni sia semplici che complesse, operazioni di codifica e decodifica, operazioni di calcolo.
- **Playbook**: permette di creare un *Detection Playbook*, ovvero una raccolta di singoli *Play*; un Play descrive gli aspetti di una strategia di detection da utilizzare. Indica cosa si vuole rilevare, il contesto in cui operare e le azioni previste per rimediare al problema riscontrato.

Security Onion prevede inoltre la possibilità di visualizzare i dati utilizzando strumenti esterni, come Grafana o Kibana.
Può essere configurato per inviare alert via email.

2.2.6 Scelta dei sistemi da installare

Valutando le caratteristiche dei sistemi considerati, si è deciso di procedere con l'effettivo deploy di **Wazuh**. Esso è infatti l'unico sistema a non essere troppo limitato nella sua versione opensource rispetto a quella a pagamento, ed a non essere troppo complesso da gestire per le finalità del progetto (come ad esempio è Security Onion). Inoltre presenta una documentazione molto chiara e completa.

Capitolo 3

Prometheus

In questo capitolo si illustrano i test effettuati utilizzando il software Prometheus.

3.1 Installazione Prometheus

Per effettuare il deploy, configurazione e testing di Prometheus si è diciso di usare delle macchine virtuali che montano un sistema operativo Debian.
Per facilitare le operazioni, è stato editato un file `.service` per avviare Prometheus come servizio.
All'avvio, Prometheus è configurato per monitorare il sistema su cui è installato.
La configurazione di Prometheus viene effettuata editando il file `prometheus.yml`; a questo possono essere aggiunti degli ulteriori file di configurazione (sempre in formato `yml`) che definiscono particolari regole definite dall'utente.
Esiste un utile tool denominato `promtool` che può essere usato per testare se i file di configurazione sono corretti o se presentano degli errori; il comando per verificare che il file di configurazione sia privo di errori è il seguente:

```
./promtool check config prometheus.yml
```

Una volta avviato il software, la dashboard di Prometheus è raggiungibile collegandosi con un qualsiasi browser all'inditizzo:

```
http://localhost:9090
```

3.2 Installazione Exporter

Per poter monitorare gli endpoint, Prometheus utilizza degli `exporter` che vengono installati su di essi: gli exporter si occupano di analizzare il sistema su cui sono installati, trasformando le informazioni in *metriche* leggibili da Prometheus.
Si è scelto di utillizare `node_exporter`: è uno degli exporter più ulizzati ed espone i dati relativi allo stato di un sistema (ad esempio: utilizzo CPU, memoria libera).
`Node_exporter` è stato installato su tre macchine virtuali Debian, e proprio come per Prometheus è stato avviato come servizio dopo aver editato un file `.service`.
Di default questo exporter utilizza la porta 9100 per esporre le metriche; si è scelto di non modificare questa configurazione.

Prometheus deve essere configurato per conoscere gli endpoint da monitorare: per far questo è necessario editare il file `prometheus.yml`; questa è la configurazione utilizzata:

```
scrape configs:
  - job name: dolly
static configs:
  - targets: ['xxx.xxx.xxx.xxx:9100']
```

dove si indica su `job name` il nome con cui l'endpoint verrà etichettato, e su `targets` l'indirizzo ip dell'endpoint ed il numero di porta sul quale sono esposte le metriche. Nella figura 3.1 si può vedere la schermata principale di Prometheus in cui vengono mostrati gli endpoint monitorati.

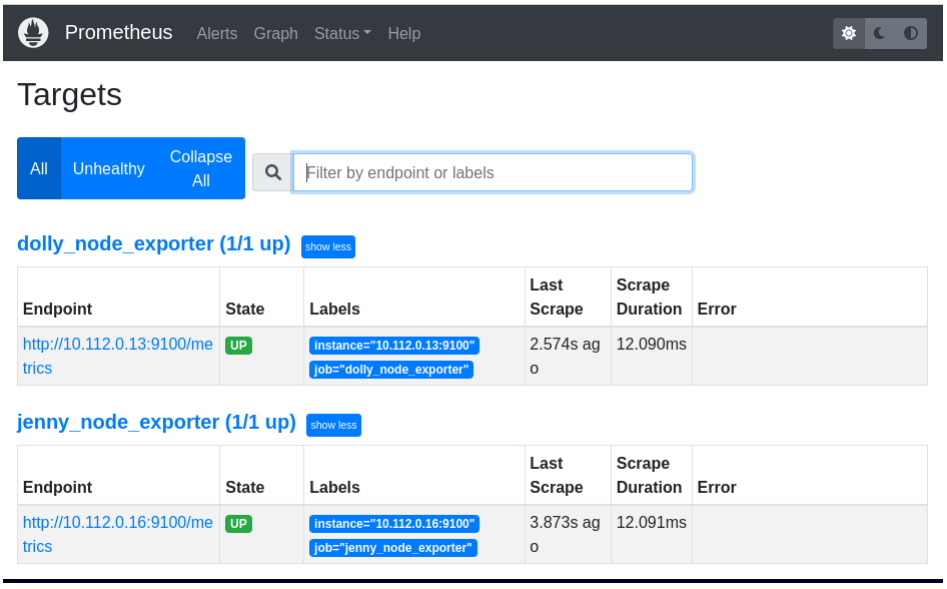


Figura 3.1: Prometheus: endpoint monitorati

Normalmente `node_exporter` espone le metriche come plain-text, senza alcun tipo di protezione. Effettuando un'analisi del traffico di rete con **Wireshark** è possibile notare che le metriche sono ben visibili, come mostrato in figura 3.2.

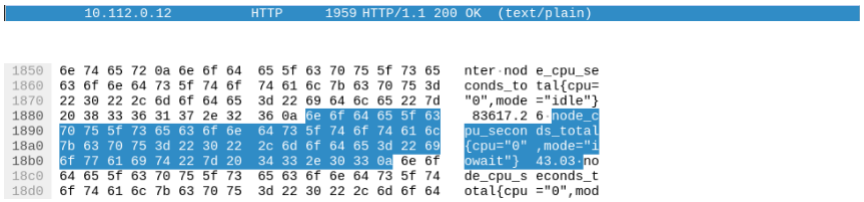


Figura 3.2: Prometheus: metriche visibili con Wireshark

Per ovviare a questo problema `node_exporter` supporta il protocollo TLS, configurabile in questo modo:

- utilizzare un certificato valido, o generarne uno (ad esempio con *openssl*).

- configurare `node_exporter` editando la posizione del certificato e della chiave:

```
tls_server_config:
   cert_file: node_exporter.crt
   key_file: node_exporter.key
```

- configurare Prometheus per utilizzare il protocollo TLS nella connessone con uno specifico endpoint:

```
- job_name: tiffany
  scrape_interval: 5s
  scheme: https
  tls_config:
    ca_file: node_exporter.crt
  static_configs:
    - targets: ['xxx.xxx.xxx.xxx:9100']
```

Una volta abilitata questa configurazione, eseguendo la stessa analisi con Wireshark non è più possibile vedere in chiaro le metriche esposte, come dimostra la figura 3.3.

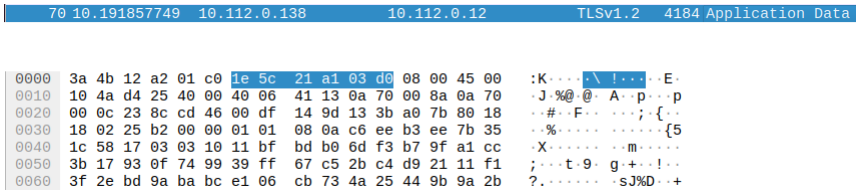


Figura 3.3: Prometheus: utilizzo del protocollo TLS

3.3 Integrare Grafana

Grafana può essere facilmete integrata con Prometheus, come sostituzione della sua dashboard nativa.
Dopo aver effettuato l'installazione, la dashboard di Grafana è accessibile all'indirizzo:

`http://Server_IP:3000/`

L'integrazione con Prometheus è già predisposta su Grafana: basta impostare Prometheus come *data source*, indicando solo l'URL a cui quest'ultimo è disponibile.
Ora che Grafana usa Prometheus come sorgente di dati, è possibile personalizzare la dashboard inserendo grafici e strumenti in una configurazione scelta dall'utente.
È inoltre possibile importare delle dashborad già configurate dalla community, semplicemente utilzzando un ID ad esse associato.
Iin questo caso si è scelto di importare una dashboard specifica per visualizzare le metriche di un `node_exporter` (ID 1860).
In figura 3.4 si può vedere come si presenta la dashboard di Grafana.

Figura 3.4: Grafana: dashboard per node_exporter

3.4 Creare un alert

Prometheus può essere configurato per generare un alert quando si verificano determinate condizioni impostate dall'utente.
Si è deciso di testare questa funzionalità verificando quando un endpoint non è più raggiungibile.
Per ottenere questo risultato si può impostare un controllo utilizzando il linguaggio di query di Prometheus (PromQL): in questo caso basta la semplice espressione `up==0`.
È necessario creare un file dove viene editata la regola che genererà l'alert: tale file può essere verificato con lo strumento `promtool` per controllarne la correttezza.
Di seguito il codice che genera un alert se un endpoint risulta inattivo per più di un minuto:

```
groups:
- name: node_is_down
  rules:
    - alert: node_is_down
      expr: up == 0
     for: 1m
     labels:
       severity: warning
    annotations:
    summary: Node(s) are down
```

La regola ha un nome ed un livello di gravità.
Si è testata la regola spegnendo una macchina virtuale tra quelle monitorate: dopo un minuto si può vedere nella dashboard di Prometheus che l'alert è stato generato, come si può osservare nella figura 3.5.

Figura 3.5: Prometheus: alert nella dashboard

3.5 Inviare un alert via email

Per poter inoltrare un alert, Prometheus dispone di un modulo (da installare a parte) denominato **Alertmanager**.
Si è deciso di testare la funzionalità inviado gli alert generati via email.
Per questo sono stati installati nella stessa macchina virtuale di Prometheus il modulo Alertmanager ed il software `Postfix`, che permette di inoltrare le email.
È stata editata la configurazione che permette a Prometheus di interagire con Alertmanager (di default utilizza la porta 9093):

```
alerting:
  alertmanagers:
    - static_configs:
      - targets:
        - localhost:9093
```

In seguito si è editato il file di configurazione di Alertmanager (anch'esso un file `.yml`) nel seguente modo:

```
route:
    group_by: ['alertname']
    receiver: smtp-local
receivers:
- name: 'smtp-local'
  email_configs:
  - to: 'PrometheusAlerts@mail.com'
    from: 'localhost@bunnies.com'
    require_tls: false
    smarthost: localhost:25
    send_resolved: true
```

Questa configurazione permette di inoltrare mail usando `Postfix` ad una casella di posta appositamente creata per i test. In figura 3.6 si può vedere che la mail viene effettivamente ricevuta nella casella apposita quando si verifica l'alert precedentemente configurato.

Figura 3.6: Prometheus: mail ricevuta in seguito ad un alert

3.6 Utilizzo di un database esterno

Prometheus offre la possibilità di inoltrare le metriche in un database esterno: solitamente questa opzione viene utilizzata per avere uno storage a lungo termine delle metriche.
Si è scelto di utilizzare **VictoriaMetrics** come database esterno compatibile con Prometheus.
Tale funzionalità è possibile editando la voce `remote_write` presente nel file di configurazione di Prometheus.
Per testare la scrittura su database esterno si è installato VictoriaMetrics su una macchina virtuale; di default, VictoriaMetrics utilizza la porta 8428.
È nescessario configurare Prometheus per comunicare con il database esterno: in questo caso basta configurare la voce `remote_write` indicando l'indirizzo e la porta della macchina dove è installato VictoriaMetrics.

```
remote_write:
- url: http://xxx.xxx.xxx.xxx:8428/api/v1/write
```

VictoriaMetrics è provvisto di una dashboard su cui visualizzare i dati ricevuti, come si può vedere in figura 3.7.

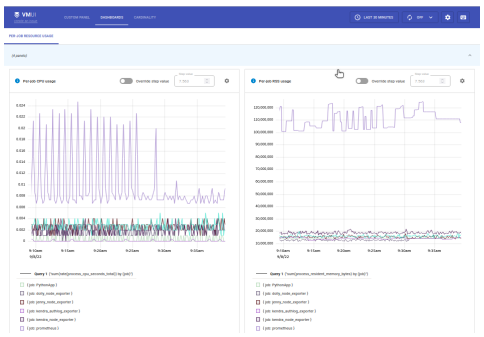


Figura 3.7: VictoriaMetrics: dashboard

Capitolo 4

Elastic Stack

In questo capitolo si illustrano i test effettuati utilizzando i software dello stack Elastic.

4.1 Installazione Elasticsearch e Kibana

Tipicamente lo stack Elastic è costituito da quattro componenti: Beats, Logstash, Elasticsearch, Kibana.
Per le finalità del progetto non vi era necessità di effettuare operazioni particolari sui log, quindi si è deciso di non installare il modulo Logstash.
Elasticsearch e Kibana sono stati installati nella stessa macchina virtuale.
Durante il processo di installazione, ElasticSearch crea degli *user* da poter utilizzare per connettersi agli altri componenti dello stack; per comodità è stato usato lo user `elastic`, che possiede privilegi di amministratore e può essere usato per qualsiasi scopo.
La password da usare con tale utente viene stampata a video durante l'installazione.
Inoltre, al termine dell'installazione verrà generato e stampato a schermo un *token* da utilizzare per configurare la connessione tra ElasticSearch e Kibana.
Elasticsearch dispone di tool per modificare questi elementi creati in automatico.
Per modificare la password generata in automatico dello user `elastic` basta utilizzare il tool apposito:

```
./elasticsearch-reset-password -v -u elastic
```

Con questo strumento è possibile sia generare automaticamente una nuova password (che verrà stampata a video), sia inserirne una scelta dall'utente.
Per generare un nuovo token da usare con Kibana è possibile usare il tool apposito:

```
./elasticsearch-create-enrollment-token -s kibana
```

Anche in questo caso il nuovo token verrà stampato a video.
Elasticsearch gira su JVM ed è importante configurare lo *heap-size* per permettere all'applicazione di funzionare correttamente. La dimensione dello *heap* consigliata per il corretto funzionamento di Elasticsearch è la metà della RAM.
Per impostare il corretto valore di *heap-size* si è creato un file in formato `.option` nel folder `/etc/elasticsearch/jvm.options.d` editato nel seguente modo:

```
-Xms1g
-Xmx1g
```

Si è così impostata la dimensione ad 1 Gb, considerando che la macchina virtuale aveva 3Gb di RAM.
Si è notato che una errata configurazione, ad esempio con valori troppo elevati o provando a tralasciare completamente la configurazione dello *heap*, causava continui crash dell'applicazione.
Al termine della configurazione di Elasticsearch si è povveduto ad installare Kibana.
Sia Elasticsearch che Kibana sono stati utilizzati con la loro configurazione di default.
Kibana utilizza la porta 5601, mentre Elasticsearch la porta 9200.
Per connettere Kibana ad Elasticsearch è sufficiente collegarsi con un qualsiasi browser all'indirizzo `http://SERVER_IP:5601` ed inserire il *token* precedentemente generato.

4.2 Beats

Per testare lo stack Elastic sono stati installati su due macchine virtuali i seguenti *Beats*:

- Metricbeat: utilizzato per visualizzare *metriche* dell'endpoint. È simile al `node_exporter` usato da Prometheus.
- Filebeat: utilizzato per raccogliere qualsisi tipo di file di log.
- Packetbeat: utilizzato per analizzare il traffico di rete.

I Beats si configurano editando un file `.yml`. La configurazione è uguale per tutti i Beats installati.
È necessario indicare l'indirizzo e la porta in cui trovare Elasticsearch, ed inserire user e password per autenticasrsi; in questo caso si è utilizzato l'utente *elastic*. È inoltre necessario fornire un *fingerprint* di un certificato: Elasticsearch genera in automatico una serie di certificati durante la fase di installazione.
Il certificato da usare in questo caso si trova su `/etc/elasticsearch/certs/http_ca.crt`.
Si può generare un *fingerprint* da questo certificato usando il comando:

```
openssl x509 -fingerprint -sha256 -in http_ca.crt
```

Nel caso non si volesse usare il certificato generato in fase di installazione, si può crearne uno nuovo utilizzando il tool di Elasticsearch denominato `elasticsearch-certutil`.
In figura 4.1 un esempio di configurazione di un Beats per connettersi correttamente ad ElasticSearch.
Ogni Beats è provvisto di vari *moduli*, che sono specializzati nel monitorare e leggere i log di una specifica tecnologia (ad esempio AWS, Docker, Kubernetes).
Per visualizzare una lista dei moduli disponibili, e verificare quali sono abilitati, basta eseguire il comando (in questo esempio si mostra per Metricbeat):

```
metricbeat modules list
```

Per abilitare un modulo, basta eseguire:

```
metricbeat modules enable <nome-del-modulo>
```

Dopo aver abilitato un modulo, può essere necessario editare il suo relativo file di configurazione, se quella di default non dovesse essere sufficiente; i file di configurazione sono nella cartella denominata `modules.d`.
Una volta terminata la configurazione e la scelta dei moduli, è possibile testare la presenza di eventuali errori utilizzando il comando (in questo esempio si mostra per Metricbeat):

```
# ========================== Outputs ============================
# ------------------- Elasticsearch Output --------------------
output.elasticsearch:
# Array of hosts to connect to.
hosts: ["xxx.xxx.xxx.xxx:9200"]
# Protocol - either ‘http‘ or ‘https‘.
protocol: "https"
# Authentication credentials
username: "elastic"
password: "..."
ssl:
enabled: true
ca_trusted_fingerprint: "0FA05BDA970..."
```

Figura 4.1: Beats: configurazione

```
./metricbeat test config -e
```

Se il test va a buon fine e non mostra errori, si può completare il setup eseguendo il comando:

```
./metricbeat setup -e
```

Accedendo a Kibana è possibile vedere i dati che arrivano dai Beats.
Kibana permette di creare delle dashboard personalizzate per visualizzare i dati sotto forma di grafici o tabelle; esistono anche delle dashboard preconfigurate.
In figura 4.2 è possibile osservare una schermata di Kabana che utilizza una dashboard preconfigurata per visualizzare i dati provenienti da Metricbeat.

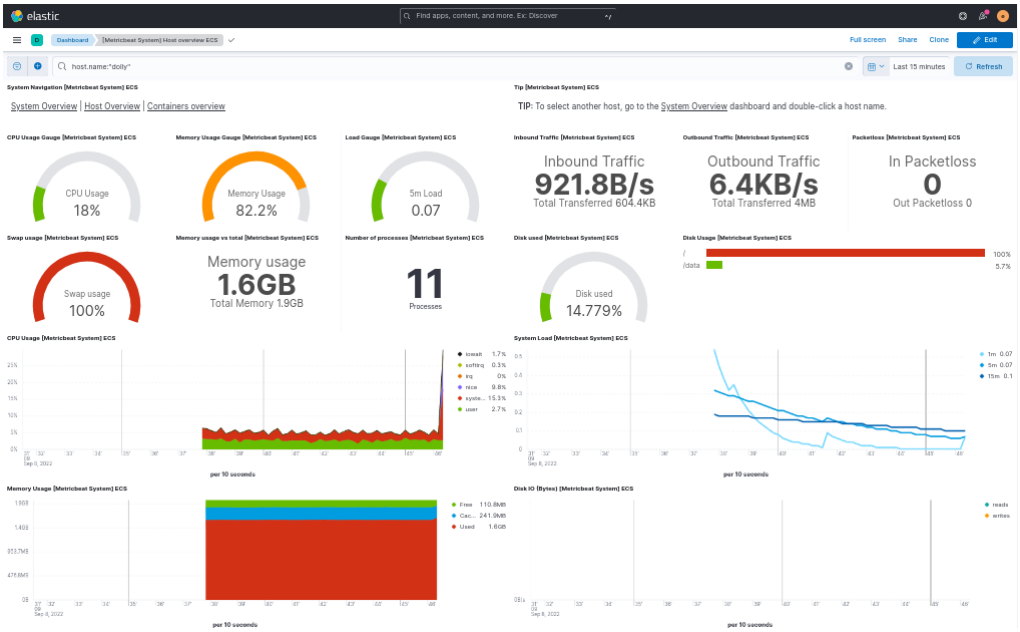


Figura 4.2: Kibana: metricbeat dashboard

4.3 Creare un alert con Elastalert2

Lo stack Elastic offre la possibilità di configurare delle regole per generare degli alert in caso si verifichi qualche evento di interesse; tramite dei *connettori* è possibile inviare gli alert con diverse modalità (ad esempio: via email, su Slack, su MSTeams). Quasi tutti i *connettori* sono disponibili unicamente nelle versioni a pagamento.
La versione base dello stack Elastic offre unicamente due opzioni per inoltrare un alert:

- scrivere gli alert su un file di log.
- salvare gli alert su un *indice* di Elasticsearch.

Per poter gestire gli alert esistono alcune soluzioni opensource che si integrano con Elasticsearch.
Si è scelto di utilizzare *Elastalert2* per inoltrare gli alert via email, utilizzando *Postfix*.
Elastalert2 è un software opensource nato come fork del progetto *elastalert* (ora non più mantenuto), scritto in python.
Dopo aver clonato il progetto dalla sua *repo* ufficiale, si procede all'installazione con:

```
python setup.py install
```

Elastalert2 si configura editando un file denominato `config.yaml`. Deve essere configurato con le informazioni necessarie per connettersi ad Elasticsearch:

```
es_host: xxx.xxx.xxx.xxx
es_port: 9200
es_username: elastic
es_password: Qw4t...
```

Completata la configurazione, bisogna creare un *indice* su Elasticsearch, su cui verranno salvati gli alert, usando il comando:

```
elastalert-create-index
```

Si è deciso di testare il sistema creando una regola che genera un alert nel caso si rilevi una connessione SSH fallita.
La regola, visibile in figura 4.3, è stata editata in un file che è stato chiamato `ssh.yaml` (è considerata buona norma avere un file specifico per ogni regola).
La regola può essere testata con il seguente comando:

```
elastalert-test-rule --config /ssh.yaml
```

Per verificare che il sistema di alerting funzioni, si è eseguita una connessione SSH su uno degli endpoint monitorati, inserendo una password errata.
Si è riscontrato che effettivamente l'alert viene generato ed inviato via email, come si può vedere in figura 4.4.

```
name: SSH login (ElastAlert 3.0.1) - 2
type: frequency
num_events: 1
filter:
  - query:
    query_string:
    query: "event.type:authentication_failure"
    index: filebeat-*
query_key:
  - source.ip
include:
- host.hostname
- user.name
- source.ip
alert:
- "email"
  email: "alertmailtest@testmail.com"
  email_from_field: "alert.elastic"
  email_add_domain: "@test.com"
  alert_text_type: alert_text_only
```

Figura 4.3: Elastalert2: struttura della regola

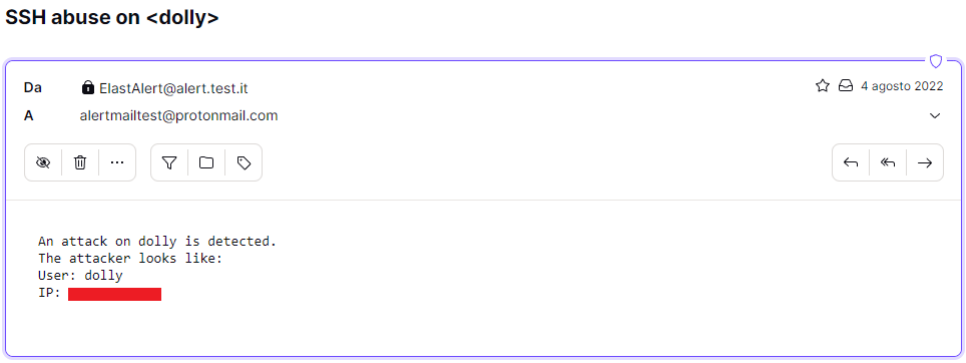


Figura 4.4: Elastalert2: email ricevuta

Capitolo 5

Wazuh

In questo capitolo si illustrano i test effettuati utilizzando il software Wazuh.

5.1 Installazione Wazuh ed agent

Wazuh offre diverse possibilità per installare i suoi componenti: è possibile installarli singolarmente oppure utilizzare uno script `.sh` per l'installazione automatizzata.
È inoltre possibile effettuare un'installazione alternativa dove si utilizza Elasticsearch come database e Kibana come dashboard: per questa installazione vengono forniti dei file di configurazione che servono a connettere i vari componenti.
Si è scelto di installare Wazuh su una macchina virtuale Ubuntu 20.04 utilizzando lo script di installazione.
Essendo tutti i componenti installati nella stessa macchina, non è stata necessaria alcuna configurazione.
Completata l'installazione, verrà stampato a schermo l'utente e la password da usare per poter accedere al sistema:

```
INFO: You can access the web interface https://<wazuh-dashboard-ip>
    User: admin
    Password: <ADMIN_PASSWORD>
INFO: Installation finished.
```

Wazuh è pronto per ricevere dati dai suoi *agent*: di default ascolta sulla porta 1514.
Si è deciso di testare il sistema installando l'agent di Wazuh su tre macchine virtuali.
L'installazione degli agent è estremamente semplice: basta aggiungere il repository di Wazuh e poi eseguire il comando:

```
WAZUH_MANAGER="xxx.xxx.xxx.xxx" apt-get install wazuh-agent
```

Come si può notare è sufficiente indicare all'agent solo l'indirizzo IP su cui può trovare il Wazuh-manager.
Completata la configurazione, la dashboard di Wazuh è accessibile all'indirizzo: `https://<wazuh-dashboard-ip>`.
Nella figura 5.1 è possibile osservare la schermata principale di Wazuh.

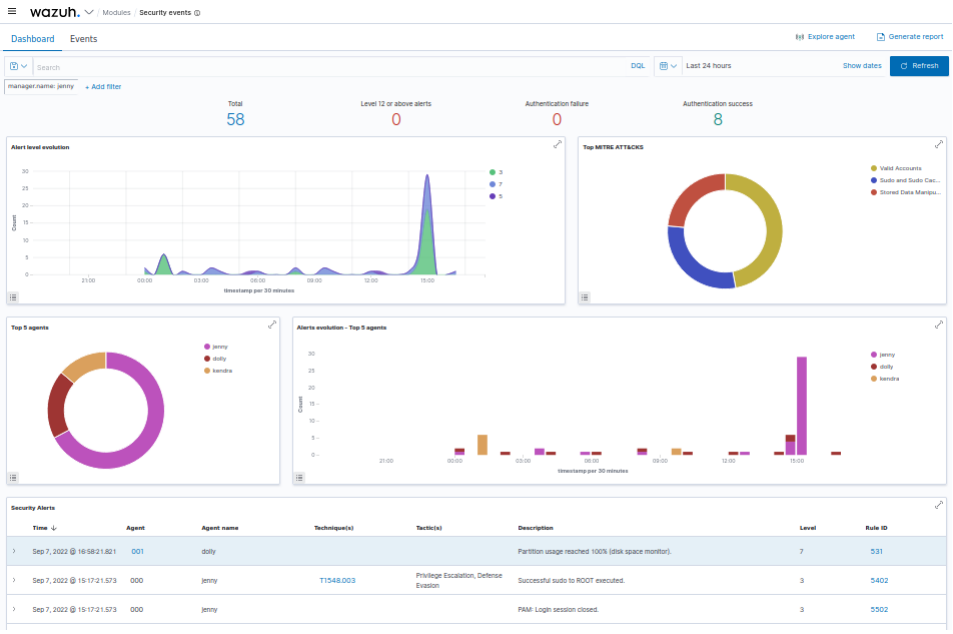


Figura 5.1: Wazuh: overview della dashboard

5.2 Integrazione con Suricata

Suricata è un software Intrusion Detection System (IDS) opensource che può essere facilmente integrato con Wazuh.
Suricata usa set di regole per monitorare il traffico di rete, e attiva degli avvisi quando si verificano eventi sospetti.
Le regole utilizzate possono essere definite dall'utente, oppure importate da diverse fonti, come ad esempio il set di regole di *Snort* o quello di *Emerging Threats*.
Suricata è stato installato su ognuna delle macchine monitorate da Wazuh.
Si è deciso di utilizzare il set di regole *Emerging Threats*: per renderlo utilizzabile è necessario scaricare le regole dal sito ufficiale, ed editare il file di configurazione di Suricata indicando il path corretto dove sono salvate le regole.

```
default-rule-path: /etc/suricata/rules
rule-files:
- "*.rules"
```

Suricata è attivo e scrive i suoi log su un file `.json` che di default si trova su `/var/log/suricata/eve.json`.
È necessario configurare l'agent di Wazuh per leggere ed inviare al sistema i log prodotti da Suricata: per far questo basta editare il file di configurazione dell'agent aggiungendo:

```
<localfile>
    <log_format>syslog</log_format>
    <location>/var/log/suricata/eve.json</location>
</localfile>
```

Gli alert generati da Suricata sono visibili nella dashboard di Wazuh; è possibile raggrupparli utilizzando il filtro `rule.groups:suricata`.

5.3 Impostare alert via email

Wazuh comprende già la funzionalità di forwarding degli alert, senza bisogno di componenti esterni da integrare.
Per abilitare l'invio di email, è sufficiente editare l'apposita sezione nel file di configurazione di Wazuh.
Anche in questo caso si è scelto di utilizzare *Postfix*.
Di seguito la configurazione di Wazuh (in questo caso si inviano email solo se l'alert è almeno di livello 10):

```
<ossec_config>
    <global>
        <email_notification>yes</email_notification>
        <email_to>mymail@test.com</email_to>
        <smtp_server>localhost</smtp_server>
        <email_from>wazuh@test.com</email_from>
    </global>

    <alerts>
       <email_alert_level>10</email_alert_level>
    </alerts>
</ossec_config>
```

5.4 Verifica del funzionamento del sistema

Si è provato a testare il sistema simulando un attacco *brute-force* SSH, utilizzando il software **Hydra**.
Per questo sono stati creati due file con un elenco di username ed uno di password (*user.txt* e *pass.txt*).
Si è lanciato Hydra simulando un *brute-force* su uno degli endpoint monitorati, lanciando il comando:

```
hydra -L user.txt -P pass.txt ssh://xxx.xxx.xxx.xxx
```

Come risultato, nella dashboard si nota immediatamente l'aumento del conteggio di login falliti, come si può notare nella figura 5.2.

Inoltre si osserva che il sistema invia una email per avvisare dell'accaduto, come si può vedere in figura 5.3.

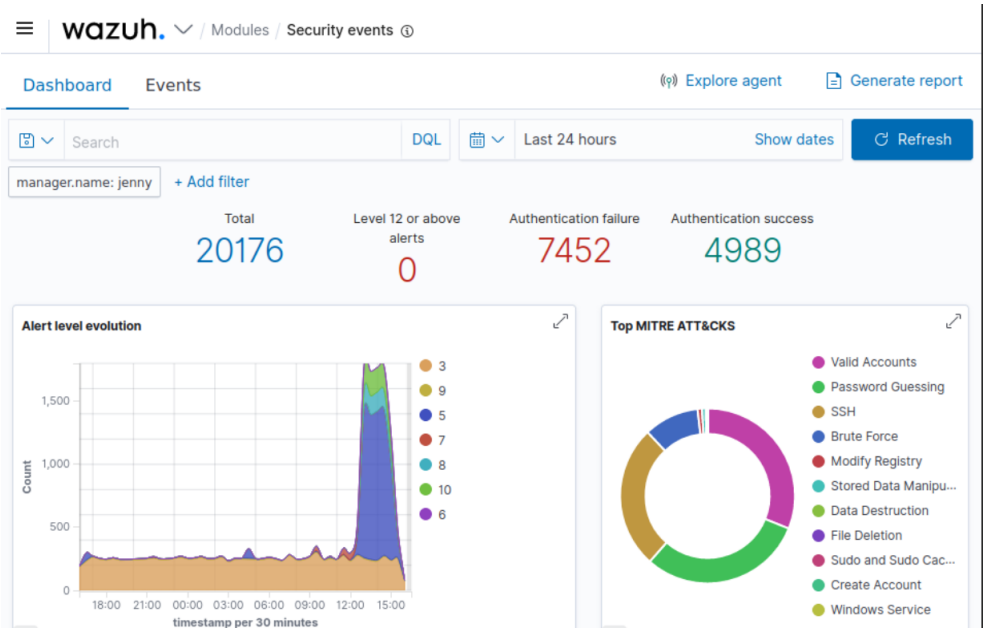


Figura 5.2: Wazuh: tentativo di brute-force

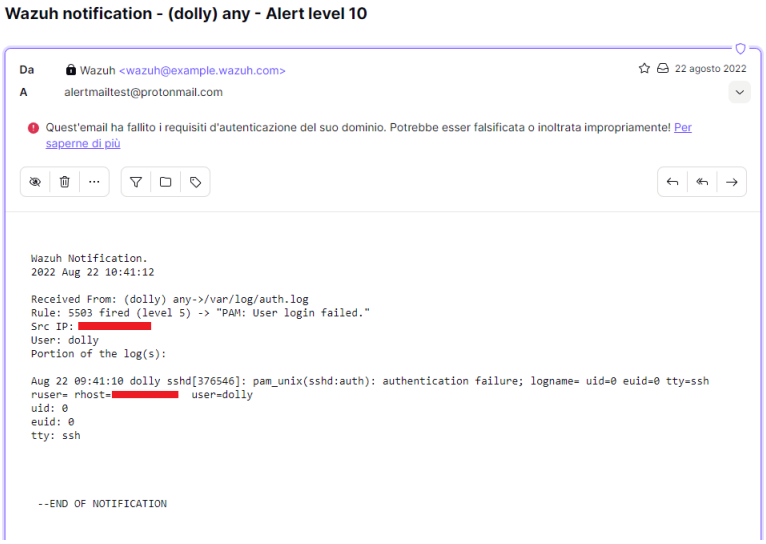


Figura 5.3: Wazuh: email in caso di alert

5.5 Creare custom rule e custom decoder

Wazuh può essere modificato e personalizzato creando nuovi *decoder* e *rules*.
Un *decoder* è in sostanza un modulo che si occupa di leggere i log e di estrarne le informazioni utili; a questo può essere assegnata una *regola* che genera un alert se le

informazioni raccolte dal decoder sono tali da poter essere considerate rischiose.
Si è deciso di testare l'utilizzo di decoder e regole custom in questo modo:

- si suppone di dover monitorare un log con la seguente struttura:

```
Test: log da testare
```

- si suppone di voler generare un alert se si si presenta questo log.

Per prima cosa è necessario creare un decoder che faccia il parsing del log; i decoder definiti da utente possono eessere editati su `/var/ossec/etc/decoders/local_decoder.xml`.
Ecco il decoder creato per il log di interesse:

```
<decoder name="test">
  <prematch>^Test:</prematch>
</decoder>

<decoder name="test_child">
  <parent>test</parent>
  <regex offset="after_parent">^ log da (\.+)</regex>
  <order>azione</order>
</decoder>
```

In questo modo il log viene controllato e viene estratta come informazione il termine *monitorare*, associato alla parola chiave *azione*: questo servirà per definire la regola che genererà l'alert.
Le regole definite da utente possono essere editate su
`/var/ossec/etc/decoders/local_rules.xml`.
Wazuh offre un tool per testare le regole ed i decoder custom, in modo tale da verificarne la correttezza prima di usarli effettivamente nel sistema; il tool si chiama `wazuh-logtest`, ed utilizzandolo per verificare il decoder creato si ottiene il seguente output:

```
    **Phanse 1: Completed pre-decoding
             full event: 'Test: log da testare'
    **Phase 2: Completed decoding.
            name: 'test'
            action: 'testare'
```

Dopo aver testato il decoder, si può procedere alla creazione della regola.

```
    <group name="myrule">
      <rule id="100025" level="5">
        <decoded_as>test</decoded_as>
        <field name="azione">testare</field>
        <description>Semplice regola di testing: parola chiave: $(azione)
        </description>
      </rule>
  </group>
```

Questa regola controlla che il campo *azione* definito nel decoder abbia il valore *testare*.
Se questo succede, viene creato un alert di livello 5.
Testando la regola con `wazuh-logtest` si ottine un output aggiuntivo.

```
**Phase 3: Completed filtering (rules)
  id: '100025'
  level: '5'
  description 'Semplice regola di testing: parola chiave: testare'
  gropus: '['test_rule']'
  firedtimes: '1'
  mail: 'False'
```

Si è deciso di effettuare un ulteriore test senza definire un decoder, ma solo una regola: l'idea è di creare una *custom rule* per monitorare l'evento di numerosi casi di login falliti in un determinato lasso di tempo (da poter valutare simulando un brute-force SSH).
Tipicamente un brute-force di questo tipo genera un alert usando la regola 5710: si è deciso di creare la *custom rule* per generare un alert nel caso il sistema identifichi un id 5710 per 5 volte in un intervallo di tempo di 200 secondi.
Di seguito viene mostrata la regola creata.

```
<group name="sshbruteforce">
  <rule name="100026" frequency="5" timeframe="200">
    <if_matched_sid>5710</if_matched_sid>
    <same_source_ip />
    <description>Possible ssh brute force attack</description>
  </rule>
</group>
```

Si è effettuato un test eseguendo un brute-force con Hydra su una delle macchine monitorate.
Si può osservare in figura 5.4 che la regola appena scritta ha generato un alert visibile sulla dashboard di Wazuh.

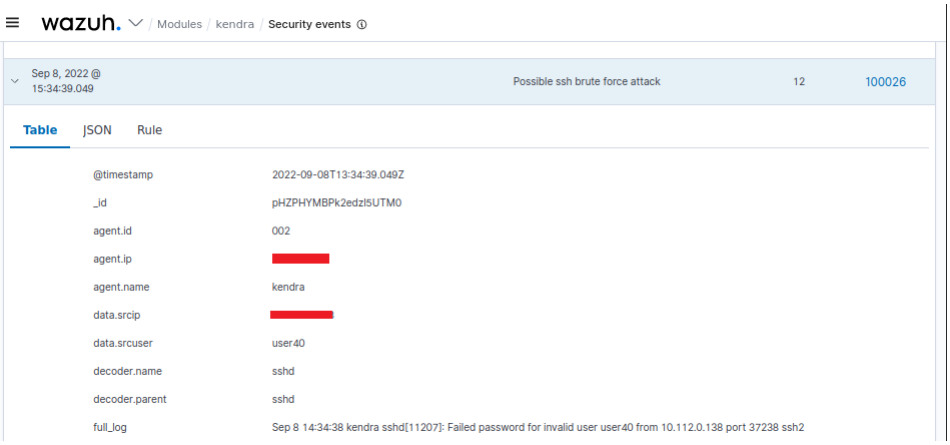


Figura 5.4: Wazuh: alert su regola definita da utente

Capitolo 6

Machine Learning

In questo capitolo si illustra la ricerca svolta per una integrazione di strumenti di Machine Learning nei sitemi di monitoraggio e sicurezza.

6.1 Orange3

Si è deciso per la parte finale del progetto di effettuare una breve ricerca sull'utilizzo di algoritmi di ML da applicare ai log di sicurezza generati dai sistemi di monitoraggio.
Per questo scopo si è deciso di utilizzare il software *Orange3*.
Orange è un pacchetto software open source di programmazione visiva basato su componenti per la visualizzazione dei dati, l'apprendimento automatico, il data mining e l'analisi dei dati.
È formato da diversi componenti detti *widget* che vengono usati per la visualizzazione dei dati, la selezione di sottoinsiemi, la valutazione empirica degli algoritmi di apprendimento e la modellazione predittiva.
Presenta un'interfaccia canvas su cui l'utente posiziona i *widget* e crea un flusso di lavoro di analisi dei dati.
Orange3 è stato installato su una macchina virtuale utilizzando **Miniconda**
Si è utilizzato per i test il *dataset* **CIC-IDS-2017**.
Tale dataset consiste in dati relativi a flussi di rete con assegnate delle *label*, che descrivono una rete in caso di normale funzionamento (`BENIGN`) o se sotto attacco (**Dos Goldeneye**, **Dos Hulk**, **Dos Slowloris**, **Heartbleed**).
Tali dati sono da usare in un sistema di ML da applicare nel campo di IDS (Intrusion Detection System).

6.2 Widget Test and Score

Questo test è stato effettuato riducendo il dataset con il *widget* **Data Sampler**, per ottenere un subset con proporzione fissa al 30% del dataset iniziale.
Test and score è un widget che valuta diversi algoritmi di classificazione, e mostra i risultati in una tabella.
In questo caso sono stati utilizzati i seguenti classificatori:

- Random Forest
- Tree
- Neural Network
- k-nearest neighbors (kNN)
- support-vector machines (SVM)

I risultati ottenuti sono riportati di seguito.

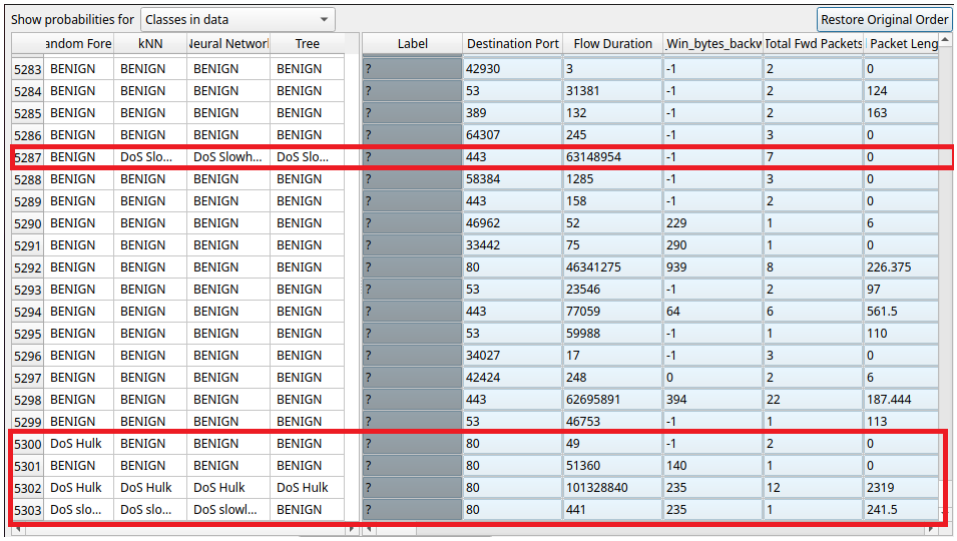
Model	Area Under The Curve	Classification accuracy	Precision
Random Forest	0.993	0.968	0.970
Tree	0.976	0.954	0.935
Neural Network	0.985	0.945	0.948
kNN	0.840	0.877	0.901
SVM	0.585	0.043	0.230

Tabella 6.1: Test and Score

Si può notare che il modello con i risultati migliori risulta essere **Random forest**.

6.3 Widget Prediction

Il widget **Prediction** riceve in input vari modelli ed un dataset privo di *labels* su cui effettuare le predizioni.
Per questo test è stato ricavato un subset cui sono state eliminate le *labels*. I modelli utilizzati sono gli stessi del test effettuato con *Test and score*.
I risultati ottenuti sono visibili in figura 6.1.
Nella figura sono stati evidenziati alcuni casi in cui i modelli effettuano previsioni errate: è possibili tuttavia notare che il classificatore **Random Forest** effettua la previsione corretta.



Show probabilities for Classes in data — Restore Original Order

	andom Fore	kNN	Neural Networ	Tree	Label	Destination Port	Flow Duration	Win_bytes_backv	Total Fwd Packets	Packet Leng
5283	BENIGN	BENIGN	BENIGN	BENIGN	?	42930	3	-1	2	0
5284	BENIGN	BENIGN	BENIGN	BENIGN	?	53	31381	-1	2	124
5285	BENIGN	BENIGN	BENIGN	BENIGN	?	389	132	-1	2	163
5286	BENIGN	BENIGN	BENIGN	BENIGN	?	64307	245	-1	3	0
5287	BENIGN	DoS Slo...	DoS Slowh...	DoS Slo...	?	443	63148954	-1	7	0
5288	BENIGN	BENIGN	BENIGN	BENIGN	?	58384	1285	-1	3	0
5289	BENIGN	BENIGN	BENIGN	BENIGN	?	443	158	-1	2	0
5290	BENIGN	BENIGN	BENIGN	BENIGN	?	46962	52	229	1	6
5291	BENIGN	BENIGN	BENIGN	BENIGN	?	33442	75	290	1	0
5292	BENIGN	BENIGN	BENIGN	BENIGN	?	80	46341275	939	8	226.375
5293	BENIGN	BENIGN	BENIGN	BENIGN	?	53	23546	-1	2	97
5294	BENIGN	BENIGN	BENIGN	BENIGN	?	443	77059	64	6	561.5
5295	BENIGN	BENIGN	BENIGN	BENIGN	?	53	59988	-1	1	110
5296	BENIGN	BENIGN	BENIGN	BENIGN	?	34027	17	-1	3	0
5297	BENIGN	BENIGN	BENIGN	BENIGN	?	42424	248	0	2	6
5298	BENIGN	BENIGN	BENIGN	BENIGN	?	443	62695891	394	22	187.444
5299	BENIGN	BENIGN	BENIGN	BENIGN	?	53	46753	-1	1	113
5300	DoS Hulk	BENIGN	BENIGN	BENIGN	?	80	49	-1	2	0
5301	BENIGN	BENIGN	BENIGN	BENIGN	?	80	51360	140	1	0
5302	DoS Hulk	DoS Hulk	DoS Hulk	DoS Hulk	?	80	101328840	235	12	2319
5303	DoS slo...	DoS slo...	DoS slowl...	BENIGN	?	80	441	235	1	241.5

Figura 6.1: Orange: Predictions

Capitolo 7

Security Operation Center

In questo capitolo si illustra la ricerca svolta su deploy e configurazione di un sistema SOC.

7.1 Componenti di un sistema SOC

Come conclusione del progetto, si è deciso di valutare l'integrazione di un sistema SIEM in un SOC completo.
Lo stack di tecnologie scelto è il seguente:

- **Wazuh**: il sistema SIEM da cui ricevere gli alert di sicurezza.
- **The Hive**: piattaforma che permette di centralizzare l'analisi di ogni evento relativo alla sicurezza e la conseguente strategia di risposta.
- **Cortex**: sistema di analisi di vari tipi di *osservabili* (come indirizzi IP, files o hashes).
- **MISP**: piattaforma opensource di intelligence sulle minacce. Il progetto sviluppa utilità e documentazione per un'intelligence sulle minacce più efficace, condividendo gli indicatori di compromissione.

7.2 The Hive

The Hive è una piattaforma di *security incident response* che gestisce gli eventi di sicurezza ricavati da varie tecnologie di monitoraggio (in questo caso Wazuh) in un unico *hub* da cui è possibile creare dei *case* da assegnare agli analisti registrati nel sistema.
È predisposto per integrarsi con Cortex e MISP e permette di effettuare analisi sugli alert e di configurare strategie di risposta alle minacce rilevate.
The Hive gira su JVM ed utilizza Elasticsearch come database. È possibile sostituire Elasticsearch con il database *Apache Cassandra*.
Una volta effettuata l'installazione (in queto caso su una macchina virtuale) è necessario creare una **Organizzazione** e degli **utenti** cui assegnare ruoli predefiniti dal sistema.
In questo caso è stato creato un utente col ruolo di *amministratore* per gestire e configurare il sistema, ed un utente *analista*.

Per ricevere eventi di sicurezza, è necessario integrare The Hive con Wazuh.
Per questo si configura Wazuh editando uno script da inserire nel path: `/var/ossec/integrations/`.
The Hive offre uno script python che permette di automatizzare l'operazione.
Completata questa operazione, si può accedere al sistema collegandosi con un qualsiasi browser all'indirizzo `localhost:9000`; The Hive riceve gli alert e li mostra nella sua schermata principale.

7.3 Cortex

Cortex rappresenta il motore di *intelligence* del sistema SOC. Effettua analisi sugli eventi di sicurezza e può essere configurato con *active response* su quest'ultimi.
Per fare questo Cortex permette di installare diversi *analizers* e *responders* sviluppati dalla comminity o di terze parte, che possono in questo modo essere utilizzati contemporaneamente da un unico *hub* centrale.
Per questo test si è scelto di non utilizzare moduli *responders*, ma solo degli strumenti di analisi.
In figura 7.1 è possibile vedere una schermata di Cortex con la lista di *analyzers* disponibili.

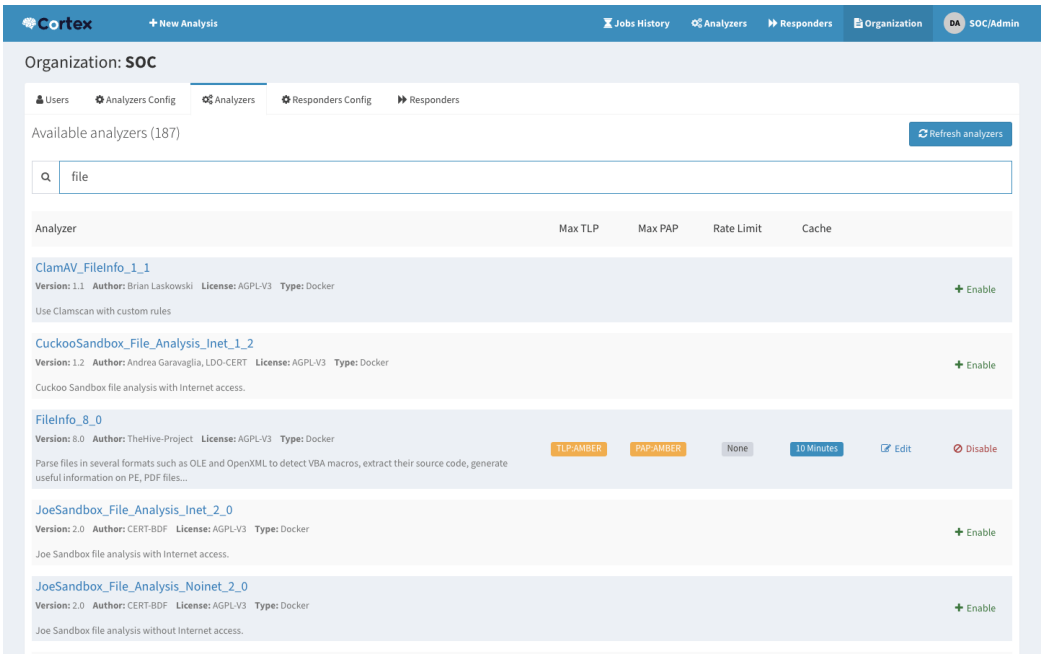


Figura 7.1: Cortex: analyzers disponibili

Si è deciso di abilitare i seguenti *analyzers*:

- **AbuseIPDB**: progetto che contiene una *blacklist* di indirizzi IP considerati collegati ad attività malevole.
- **Virustotal**: progetto che permette l'analisi gratuita di files e URLs per scovarne virus o malware all'interno.

Per integrare Cortex e The Hive è necessario editare il file di configurazione di The Hive indicando indirizzo IP cui trovare il server in cui è presente Cortex.
Completata la configurazione , è possibile utilizzare gli *analyzers* direttamente da The Hive, come si può federe in figura 7.2.

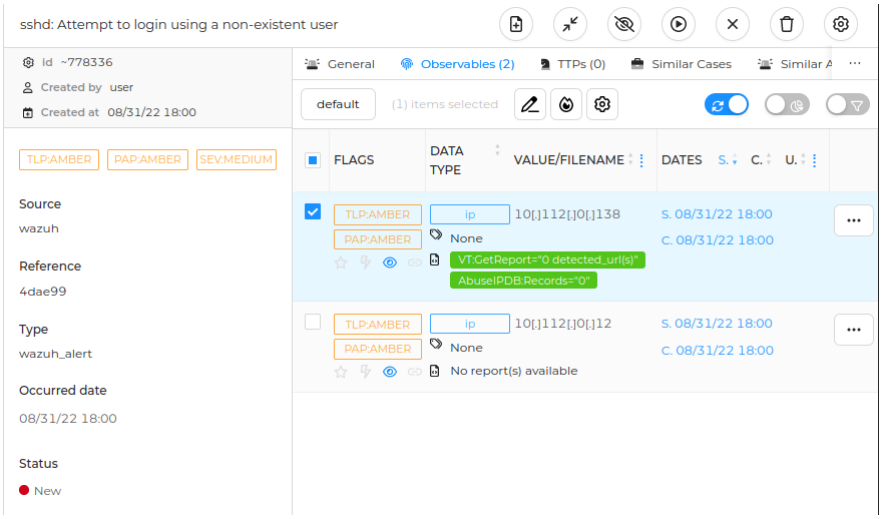


Figura 7.2: The Hive: analyzers su log di Wazuh

7.4 MISP

MISP è una piattaforma che contiene documenti e metadati relativi a diversi indicatori di compromissione.
Integrando MISP nel SOC si rende possibile consultare tutte le risorse rese disponibili da MISP direttamente da The Hive; inoltre è possibile inviare dati a MISP nel caso si sospetti un nuovo tipo di minaccia non ancora presente nel sistema.
Una volta installato, MISP è raggiungibile tramite qualsiasi browser all'indirizzo: `https://localhost:443`.
Gli indicatori di compromissione utilizzabili con MISP sono da importare selezionando dei contenitori detti *feeds*: ogni feed importato viene automaticamente aggiornato dal sistema ad intervalli di tempo regolari e configurabili.
Per utilizzare MISP nel sistema, è necessario configurare l'integrazione sia con Cortex che con The Hive.

- **Integrazione con The Hive**: si ottiene editando il file di configurazione di The Hive (*application.conf*) nella sezione appositamente dedicata a MISP.
- **Integrazione con Cortex**: generare dal pannello `Administration -> List Auth Keys` di MISP una chiave; su Cortex abilitare l'*analyzer* textttMISP_2_1 usando la chiave precedentemente generata.

È ora possibilie analizzare gli alert ricavati da Wazuh con l'*analyzer* di MISP.

7.5 Testare il sistema con attacco Shellshock

Shellshock, noto anche come Bashdoor, è una famiglia di bug di sicurezza nella shell Bash, il primo dei quali è stato divulgato il 24 settembre 2014.
Questo attacco permette di eseguire comandi ed ottenere accessi non autorizzati a servizi internet (come ad esempio web server) che usano Bash per processare le richieste.
Wazuh riconosce questo tipo di attacco con la regola 31168 (*Shellshock attack detected*).

Si è deciso ti testare il sistema SOC effettuando un attacco di tipo Shellshock su un endpoint monitorato da Wazuh.
Sull'endpoint scelto per l'attacco è stato installato un web server `nginx`.
Successivamente è stato configurato l'agent di Wazuh per leggere i log generati dal web server, aggiungendo la voce:

```
<localfile>
    <log_format>syslog</log_format>
    <location>/var/log/nginx/access.log</location>
</localfile>
```

Si è effettuato l'attacco lanciando il comando:

```
curl -H "User-Agent: () { :; }; /bin/cat /etc/passwd" <WEBSERVER-IP>
```

L'attacco viene bloccato dal web server, essendo ormai stato risolto con numerose patch.
Viene comunque tracciato nei file di log letti dall'agent di Wazuh.
Il tentativo di attacco viene immediatamente registrato come alert da Wazuh, come visibile in figura 7.3.

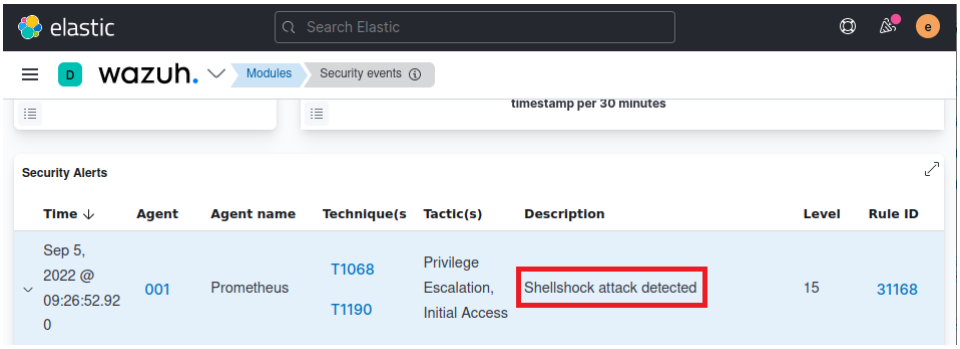


Figura 7.3: Wazuh: shellshock attack detected

L'attacco rilevato da Wazuh viene propagato a The Hive.
Da qui l'alert potrà essere gestito effettuando sugli osservabili ad esso relativi le verifiche possibli dagli *analyzers* Configurati con Cortex (come visibile in figura 7.4).
Da qui potrà inoltre essere creato un *case* da assegnare ad uno o più *user* registrati nel sistema.

Inoltre è possibile visualizzare tutti i dati relativi all'attacco individuato presenti su MISP.

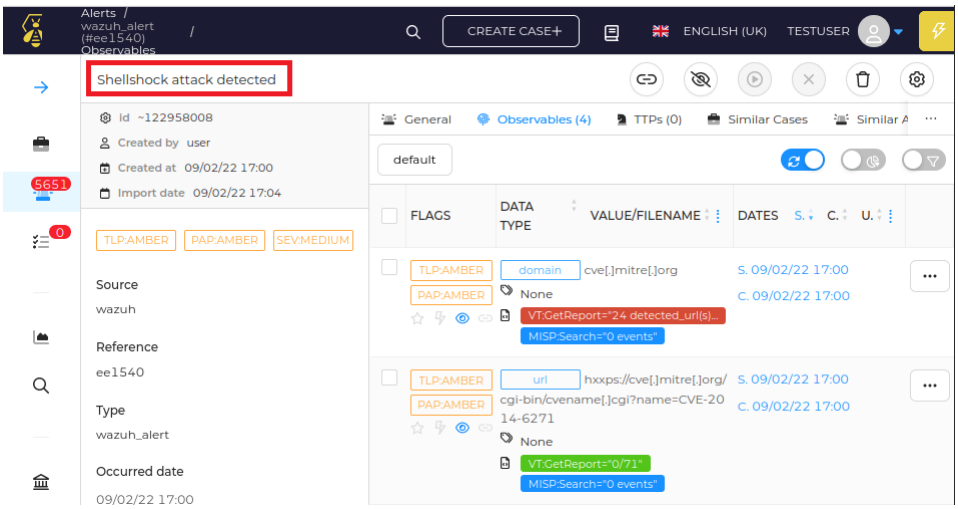


Figura 7.4: The Hive: shellshock attack detected

Capitolo 8

Conclusioni

In questo capitolo si illustrano le conclusioni relative allo stage.

8.1 Riassunto del lavoro svolto

Per completare questo progetto è stato svolto un lavoro di ricerca sul funzionamento delle tecnologie rientranti negli ambiti di log monitoring e Siem, valutando i vari prodotti disponibili per poi scegliere quali testare, in base alle caratteristiche descritte nella documentazione ufficiale.
Dei sistemi scelti è stato quindi eseguito il deploy e la configurazione, testando l'integrazione con vari software e le capacità dei sistemi.
Nella parte conclusiva del progetto è stata svolta una ricerca sulla possibilità di utilizzare strumenti di Machine Learning come aiuto ai software di sicurezza, valutando quali classificatori effettuino le predizioni migliori. Infine si è testata la possibilità di integrare un sistema SIEM in un Security Operation Center, scegliendo quali tecnologie utilizzare ed effettuando dei test sul funzionanento del sistema.

8.2 Conoscenze acquisite

Nel corso dello stage sono state acquisite le seguenti nozioni:

- principali tecnologie opensource di log monitoring: si sono apprese le diverse strategie utilizzate da diversi sistemi per la gestione ed il monitoraggio dei log, le possibilità che tali sitemi offrono e l'importanza di utilizzarli in ambito aziendale. Si è inoltre acquisita maggiore conoscenza sulla struttura e sulle informazioni contenute nei diversi tipi di log e come queste informazioni possono essere utilizzate.

- principali sistemi SIEM opensouce: si sono apprese le diverse tecnologie in ambito SIEM, e le possibili features che questi sistemi rendono disponibili.

- Machine Learning applicato alla sicurezza: si è visto come un sistema di ML può essere utilizzato con dati relativi alla sicurezza, per avere un'analisi aggiuntiva sugli alert che potrebbe indicare degli indici di compromissione ignotati o evidenziare dei possibili falsi positivi.

- sistema SOC: si sono apprese le conoscenze necessarie ad integrare un SIEM in un centro operativo per la sicurezza informatica, e come questo può essere utilizzato per riconoscere ed analizzare rischi e possibili anomalie nel sistema.

8.3 Valutazione personale

Il progetto di stage offerto da Athesys si è svolto al meglio.
Il tutor Mattia Zago e l'azienda sono stati disponibili per qualsiasi necessità o chiarimento.
L'azienda ha offero libertà di scelta per quello che riguarda le tecnologie da testare.
Tutti gli obiettivi iniziali previsti dal progetto sono stati portati a compimento. Lo stage mi ha permesso di acquisire nuove competenze e di agire indipendentemente per quello che riguarda la soluzione di problemi e la ricerca del modo migliore di configurare i sistemi utilizzati.

Acronimi e abbreviazioni

AMPQ Advanced Message Queuing Protocol. 5

IDS Intrusion Detection System. 24, 29

JVM Java Virtual Machine. 4, 17, 33

kNN k-nearest neighbors. 30

ML Machine Learning. 2, 29, 39

PromQL Prometheus Query Language. 3

SIEM Security Information and Event Management. 1, 2, 8, 10, 33, 40, 43

SOC Security Operation Center. 2, 33–36, 40

SSPL Server Side Public License. 4

SVM support-vector machines. 30

TSDB Time Series Database. 3, 5

XDR eXtended Detection and Response. 8

Glossario

AMPQ *AMPQ* (Advanced Message Queuing Protocol) è uno standard aperto che definisce un protocollo a livello applicativo per il message-oriented middleware. AMQP è definito in modo tale da garantire funzionalità di messaggistica, accodamento, routing, affidabilità e sicurezza. . 41

IDS *IDS* (Intrusion Detection System) è un dispositivo software o hardware utilizzato per identificare accessi non autorizzati ai computer o alle reti locali. . 41

JVM *JVM* (Java Virtual MAchine) è una macchina informatica astratta - virtuale appunto - che consente ad un computer di eseguire un programma Java. . 41

kNN *kNN* (k-nearest neighbors) è un algoritmo utilizzato nel riconoscimento di pattern per la classificazione di oggetti basandosi sulle caratteristiche degli oggetti vicini a quello considerato. . 41

ML in informatica con il termine *ML* (Machine Learing) si indica una sottocategoria dell'intelligenza artificiale che si riferisce al processo tramite il quale i computer sviluppano il riconoscimento dei pattern, o la capacità di apprendere continuamente ed effettuare previsioni utilizzando i dati.. 41

PromQL *PromQl* (Prometheus Query Language) è un linguaggio di query funzionale nativo della tecnologia Prometheus, che permette di selezionare ed aggregare metriche.. 41

SIEM in informatica con il termine *SIEM* (Security information and event management) si indica una tecnologia che unisce la gestione delle informazioni di sicurezza (SIM) e la gestione degli eventi di sicurezza (SEM) in un unico sistema di gestione della sicurezza. 41

SOC in informatica con il termine *SOC* (Security Operation Center) si indica un'entità che fornisce un servizio di rilevamento degli incidenti, osservando gli eventi tecnici nelle reti e nei sistemi, e può anche essere responsabile della risposta e della gestione degli incidenti.. 41

SSPL *SSPLl* (Server Side Public License) è una licenza software proprietaria/disponibile che stabilisce in modo chiaro ed esplicito i termini per la distribuzione del programma concesso in licenza come servizio di terze parti. . 41

SVM *SVM* (support-vector machines) è un modello di apprendimento supervisionato associato ad algoritmi di apprendimento per la regressione e la classificazione. . 41

TSDB in informatica con il termine *TSDB* (Time Series Database) si intende un sistema software ottimizzato per l'archiviazione e l'elaborazione di serie temporali tramite coppie di tempo e valore associate.. 41

XDR *XDR* (eXtended Detection and Response) è un sistema che raccoglie e correla automaticamente i dati tra più livelli di sicurezza: email, endpoint, server, workload in cloud e rete. Ciò permette di rilevare più velocemente le minacce e di migliorare i tempi di indagine e di risposta attraverso l'analisi della sicurezza. . 41

Bibliografia

Alien Vault Ossim. URL: https://cybersecurity.att.com/documentation/usm-appliance/initial-setup/ossim-installation.htm

Cortex. URL: https://docs.thehive-project.org/cortex/

Elastalert2. URL: https://elastalert2.readthedocs.io/en/latest/

Elastic Stack. URL: https://www.elastic.co/guide/index.html

* *Elasticsearch*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
* *Filebeat*. URL: https://www.elastic.co/guide/en/beats/filebeat/current/index.html
* *Kibana*. URL: https://www.elastic.co/guide/en/kibana/current/index.html
* *Licenza Elastic*. URL:https://www.seacom.it/facciamo-chiarezza-sulla-questione-delle-licenze-elastic/
* *Logstash*. URL:https://www.elastic.co/guide/en/logstash/current/index.html
* *Metricbeat*. URL: https://www.elastic.co/guide/en/beats/metricbeat/current/index.html
* *Pachetbeat*. URL: https://www.elastic.co/guide/en/beats/filebeat/Packetbeat/index.html

Grafana. URL: https://grafana.com/docs/grafana/latest/

Graphite. URL: https://graphite.readthedocs.io/en/latest/index.html

Icinga. URL: https://icinga.com/docs/icinga-2/latest/doc/01-about/

MISP. URL: https://www.misp-project.org/documentation/

Nagios Core. URL: https://exchange.nagios.org/directory/Documentation/Nagios-Core-Documentation

Orange. URL: https://orangedatamining.com/docs/

Ossec. URL: https://www.ossec.net/docs/

Prometheus. URL: https://prometheus.io/

* *Alertmanager*. URL: https://prometheus.io/docs/alerting/latest/alertmanager/

* *Node exporter*. URL: https://prometheus.io/docs/guides/node-exporter/

* *Node exporter TLS*. URL: https://inuits.eu/blog/prometheus-tls/

The Hive. URL: https://docs.thehive-project.org/thehive/

Security OnionOnion. URL: https://docs.securityonion.net/en/2.3/

Sensu. URL: https://docs.sensu.io/sensu-go/latest/

Suricata. URL: https://suricata.readthedocs.io/en/suricata-6.0.9/

Victoria Metrics. URL: https://docs.victoriametrics.com/

Wazuh. URL: https://documentation.wazuh.com/current/index.html