SAPIENZA
UNIVERSITÀ DI ROMA

Progettazione e sviluppo di una piattaforma Web per la generazione visuale di query per tecnologie SIEM

Facoltà di Ingegneria dell'informazione, informatica e statistica presso UNITELMA

Corso di Laurea in Informatica

Candidato

Alessio Giovannini
Matricola 1700191

Responsabile

Prof. Angelo Monti

Co-responsabile

Alessio Dalla Piazza

Anno Accademico 2022/2023

Progettazione e sviluppo di una piattaforma Web per la generazione visuale di query per tecnologie SIEM

Relazione di Tirocinio. Sapienza – Università di Roma

Questa relazione è stata composta con LaTeX e la classe Sapthesis.

Versione: 30 gennaio 2023

Email dell'autore: giovannini.1700191@studenti.uniroma1.it

Sommario

L'oggetto di questo tirocinio è la progettazione e lo sviluppo di una piattaforma web per supportare gli analisti nei SOC o i ricercatori nella creazione di query specifiche per SIEM. La relazione si compone di 4 parti fondamentali:

- l'introduzione allo **scenario di riferimento** e alla necessità del lavoro svolto;
- l'**analisi dei requisiti**, volta a migliorare la modellazione delle funzionalità della piattaforma;
- la **progettazione del sistema** e della sua architettura;
- i dettagli dello sviluppo e delle **fasi di implementazione**.

Indice

Capitolo 1

Introduzione

1.1 Tecnologie SIEM e il loro ruolo nella sicurezza difensiva

Un **Security Information and Event Management (SIEM)** è una soluzione software il cui scopo è quello di essere di supporto per la rilevazione, analisi e risposta alle minacce alla sicurezza prima che tali possano provocare danni. Un SIEM è composto dalla combinazione di due soluzioni:

- Il **Security Information Management (SIM)** il quale si occupa di raccogliere, monitorare e analizzare i dati relativi alla sicurezza dai log dei computer e da varie altre fonti di dati;
- Il **Security Event Management (SEM)** il quale si occupa di registrare e valutare gli eventi generati dai sistemi monitorati supportando gli amministratori di sicurezza nell'analisi, regolazione e gestione dell'architettura, delle politiche e delle procedure di sicurezza delle informazioni.

Le tecnologie SIEM vengono spesso adottate all'interno dei SOC, in singolo o combinati con altre tecnologie affini come EDR, IDS e molte altre.

L'utilizzo di tecnologie SIEM si dimostra cruciale per la prevenzione e la risposta agli incidenti di Cybersecurity che colpiscono con sempre più frequenza le organizzazioni.

Cos'è un SOC? Un **Security Operation Center (SOC)** rappresenta un'unità, che può essere centralizzata internamente all'organizzazione o esterna ad essa, che si occupa di tenere in sicurezza la rete dell'azienda presso cui opera, monitorando e rispondendo agli incidenti di Cybersecurity. Un SOC si occupa di raccogliere i log da tutte le infrastrutture IT dell'organizzazione da diverse tipologie di dispositivi presenti all'interno della rete. All'interno di un SOC sussistono diversi ruoli con diversi gradi di responsabilità.

- Gli **Analisti di 1° Livello** i quali si occupano dell'analisi dei dati raccolti e della valutazione riguardo gli avvisi di sicurezza generati dalla tecnologia impiegata. Dopo un'attenta valutazione, gli analisti di 1° livello passeranno di dati che richiedono necessità di analisi più specifiche agli analisti di 2° livello;

- Gli **Analisti di 2° Livello**, i quali si occupano di performare indagini più approfondite sui dati che vengono passati dagli analisti di 1° livello per capire il grado di minaccia e identificare possibili soluzioni. Qualora il livello della minaccia si dimostri troppo complesso da risolvere per le competenze possedute dagli analisti di 2° livello quest'ultimi richiedono l'intervento degli analisti di 3° livello;
- Gli **Analisti di 3° Livello** i quali si occupano di gestire gli incidenti di gravità elevata. Alcuni compiti che un'analista di tale livello potrebbe dover svolgere sono il reverse engineering di un binario sospetto o l'analisi di un malware. Gli analisti di tale livello possono inoltre performare *Vulnerability Assestment* all'interno dell'organizzazione per ricercare possibili falle nella sicurezza della rete aziendale;
- I **SOC Manager** i quali si occupano di supervisionare e organizzare la struttura di un SOC e le varie operazioni che ne conseguono.

1.2 Le licenze software

Qualsiasi prodotto software esistente, salvo rari casi, è sempre vincolato da delle licenze di utilizzo, ovvero una serie di autorizzazioni che vengono concesse all'utente per quanto riguarda l'utilizzo, la modifica e l'eventuale ridistribuzione del software in questione con le relative risorse associate ad esso.

Esistono diverse tipologie di licenze adottabili nell'ambito software, da quelle che consentono un utilizzo a fini commerciali a licenze più permissive che permettono la copia, modifica e ridistribuzione del codice sorgente.

Un esempio di licenza è la licenza **Mozilla Public License 2.0** [8] la quale viene catalogata come licenza copyleft e i punti cardini di tale licenza si basano sul fatto di mettere a disposizione il codice sorgente dei file licenziati, e le modifiche dei file stessi, con la medesima licenza (o, in alcuni casi, con una delle licenze *GNU*). Inoltre le note sul copyright e sulla licenza devono essere preservate. Ai contributori è fornita una esplicita concessione sui brevetti. Comunque, un lavoro basato sull'opera in oggetto può essere distribuito con una licenza diversa, anche proprietaria e anche senza il codice sorgente dei file aggiunti.

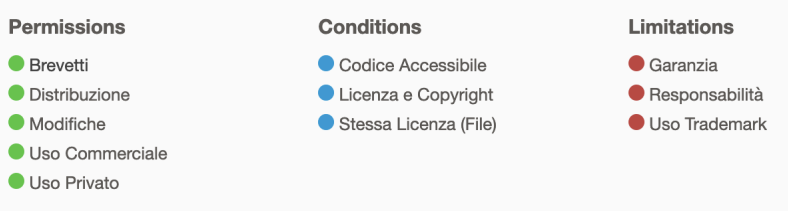

Figura 1.1. Permessi, condizioni e limitazioni specifici per la MPL 2.0

Nel progetto oggetto di tale relazione sviluppato si è adottata la **GNU General Public License V 3.0**.

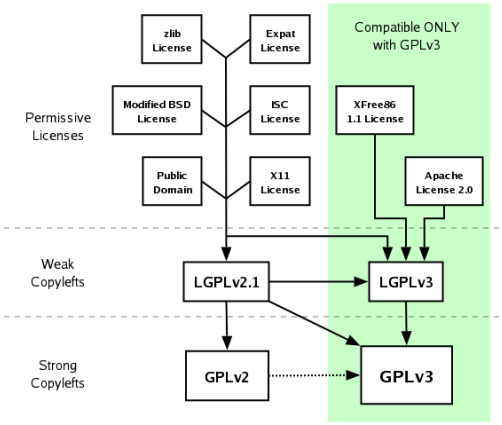

Figura 1.2. Confronto tra le varie license

La GNU General Public License V 3.0 [4] è una licenza copyleft la quale stabilisce l'obbligo di di mettere a disposizione l'intero codice sorgente dell'opera e delle sue modifiche le quali possono includere anche opere più estese che usano quella originaria (come nel caso del progetto oggetti di tale tesi) con la medesima licenza. Le note sul copyright e sulla licenza devono essere preservate e, inoltre, ai contributori è fornita una esplicita concessione sui brevetti.

1.3 La necessità della piattaforma

Il settore della Cybersecurity è ormai saturo di strumenti volti a migliorare il lavoro di operatori sia lato Red Team che lato Blue Team, partendo da strumenti utilizzabili

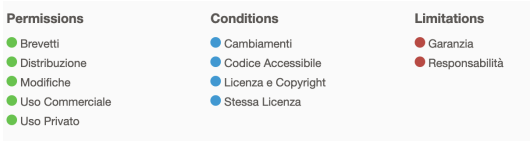

Figura 1.3. GNU Public License version 3

da riga di comando a veri e propri software ad alta complessità che fanno affidamento sull'ormai sempre più diffusa Intelligenza Artificiale modellata per gli specifici usi. Un esempio di soluzione software ad alta complessità è **Wazuh**[16], una soluzione EDR che tra le molteplici funzionalità include proprio l'utilizzo dell'Intelligenza Artificiale, al fine di automatizzare e migliorare la detection di possibili minacce alla sicurezza.

Salvo rare eccezioni, la quasi totalità di strumenti con funzionalità avanzate presenti sul mercato presentano costi, talvolta non indifferenti, per le organizzazioni che vogliono adottarle.

Esistono però anche strumenti liberamente accessibili che supportano gli analisti e i ricercatori nel loro lavoro quotidiano, un esempio concreto è il tool **sigma**[9], liberamente scaricabile da GitHub.

La problematica intriseca della maggiorparte di tali strumenti è che richiedono una linea di apprendimento mediamente ripida e competenze che non tutti gli addetti lavori potrebbero possedere, per questo motivo nasce l'esigenza di trasformare un tool come **sigma**[9] in una Web Application con un interfaccia visuale, esonerando quindi l'analista dal dover configurare manualmente un ambiente di lavoro e imparare ad usare le tecnologie necessarie per l'utilizzo dello strumento.

1.4 La tecnologia su cui si basa il progetto

L'applicativo web sviluppato si basa su uno strumento esistente di nome **sigma**[9], uno software scritto in **Python**[18] utilizzabile esclusivamente da riga di comando.

sigma si presenta come un formato di firma generico e aperto che consente di descrivere i log degli eventi rilevanti in modo semplice. Più in dettaglio, prendendo in input una regola definita dall'utente genera una query utilizzabile sulla tecnologia SIEM specifica, esonerando l'analista dal dover conoscere le varie sintassi di ogni soluzione esistente.

Oltre la conoscenza della riga di comando per usare a pieno sigma è necessario anche essere confidente con il linguaggio **YAML**[2] utilizzato per scrivere le regole che verranno poi convertite da sigma per ottenere la query finale specifica per la tecnologia SIEM utilizzata.

Capitolo 2

Analisi dei requisiti

2.1 Requisiti della piattaforma

Inizialmente sono stati raccolti ed analizzati i requisiti principali della piattaforma, così da poterne definire ulteriori funzionalità e guidare le successive fasi dello sviluppo.

Gli attori[1] del sistema hanno requisiti di base che gli permettono di accedere solamente al front-end dell'applicazione e sono in grado di eseguire i seguenti compiti:

- **Creazione della regola**: tramite l'interfaccia visuale fornita, l'utente può scrivere la regola che verrà poi tramutata in query dall'applicativo;
- **Caricamento della regola**: L'utente può caricare una regola già creata da file;
- **Salvataggio della regola**: L'utente può richiedere il download sotto forma di file YAML della regola creata con l'editor visuale;
- **Generazione della query**: scegliendo la tecnologia SIEM desiderata, con una delle configurazioni disponibili e una regola ben strutturata, l'utente può ottenere la query SIEM specifica in risposta dall'applicativo.

Generazione delle regole L'utente ha quindi due modalità con le quali sottomettere una regola all'applicazione: generarla manualmente tramite l'editor visuale o caricarne localmente una già pronta da file. L'editor possiede inoltre una serie di funzionalità per aiutare l'utente nella strutturazione della regola stessa, difatti rende impossibile cancellare i campi obbligatori nella regola garantendo di fatto la corretta semantica della stessa e, inoltre, fornisce suggerimenti di inserimento a seconda del campo selezionato dall'utente durante la fase di editing.

Richiesta di generazione della query Affinchè risulti possibile ottenere la query SIEM-specific desiderata è necessario strutturare correttamente la regola che si vuole sottoporre all'applicativo. Tale regola infatti deve avere dei campi non arbitrari, i quali vengono specificati sia nella documentazione del progetto stesso che in quella di sigma.

[1] Coloro che interagiscono con il sistema

Dettaglio e accettazione/rifiuto regola Il sistema, dopo un'opportuna verifica, può decidere se scartare o accettare la richiesta di generazione della query; Tale richiesta viene valutata analizzando la correttezza sintattica della regola sottomessa al sistema. Oltre a ciò i controlli vengono anche eseguiti sugli altri campi necessari per la creazione della richiesta, in assenza di uno dei campi obbligatori l'applicativo non eseguirà alcuna routine.

Generazione della regola per il download L'utente può decidere, tramite l'apposita opzione visuale, di effettuare il download della regola scritta tramite l'editor visuale fornito dall'applicativo; tale regola verrà formattata e scaricata sotto forma di file YAML.

2.2 Raffinamento e funzionalità aggiuntive

I requisiti appena elencati rappresentano le funzionalità essenziali per permettere la creazione e gestione delle regole scritte, ovvero lo scopo principale della piattaforma. È però necessario tenere anche conto dei dati relativi alle risorse e alle loro versioni; è quindi seguita una fase di raffinamento dei requisiti e delle funzionalità.

Integrazione L'applicativo è stato integrato all'interno di una piattaforma di e-learning ove sono presenti dei laboratori virtuali creati da `CYS4`. L'applicativo viene distribuito tramite container **Docker**[6] e si avvia all'interno di ogni istanza del laboratorio, questo fa si che ogni utente abbia accesso esclusivo all'applicativo e per questo motivo non si è reso necessario ne lo sviluppo di un sistema di registrazione e di autenticazione ne l'implementazione di un eventuale database a supporto dell'infrastruttura.

Capitolo 3

Progettazione

3.1 Architettura

Il progetto è basato su una architettura monolitica single-tier.

Un'applicazione monolitica viene, by design, distribuita come modulo singolo ma può comunque interagire con altri servizi o database durante l'esecuzione delle operazioni. Una delle caratteristiche di questo tipo di applicazioni è la possibilità di scalare la stessa in maniera orizzontale, ossia duplicare l'applicazione stessa su più server o macchine virtuali. L'organizzazione dell'applicazione avviene attraverso la suddivisione della stessa in più livelli logici.

I livelli logici L'organizzazione di un applicazione attraverso livelli logici consiste nella suddivisione della stessa in base al tipo di responsabilità e competenza delle varie funzionalità implementate. Utilizzando questa suddivisione si garantisce una migliore manutenzione e sviluppo della codebase oltre a consentire il riuso del codice osservando il principio **DRY (Don't Repeat Yourself)**.

In un'architettura a livelli, le applicazioni possono imporre restrizioni su quali livelli possono comunicare con altri livelli. Le operazioni di modifica o sostituzione di un livello, di default, non impattano sugli altri livelli essendo ognuno di essi indipendenti gli uni dagli altri, evitando così le modifiche sull'intero applicativo.

I livelli, e l'incapsulamento, facilitano la sostituzione delle funzionalità all'interno dell'applicazione. Un esempio potrebbe essere quello di un'applicazione che inizialmente faccia uso di un prorio database per il salvataggio di dati, e in seguito si decida di effettuare un cambio di strategia di salvataggio permanente basata sul cloud o su un'altra tecnologia; Questa operazione appena descritta si renderebbe possibile a patto che l'implementazione della persistenza all'interno di un livello logico sia stata incapsulata correttamente.

I livelli dell'applicazione rendono semplice la sostituzione di parti dell'applicativo a fine di eseguire test su specifiche componenti; invece di scrivere test che operano sul livello dei dati reali o dell'interfaccia utente dell'applicazione, questi livelli possono essere sostituiti in fase di test con implementazioni che emulano quelle originali e che restituiscono risposte note alle richieste. Utilizzando questo tipo di approccio si garantisce una maggiore facilità nello sviluppare ed eseguire i suddeti test.

I livelli rappresentano la separazione logica all'interno dell'applicazione. Nel caso in cui la logica dell'applicazione sia fisicamente distribuita in server o processi separati, queste destinazioni di distribuzione fisiche separate sono denominate livelli. È possibile, e piuttosto comune, che un'applicazione a più livelli sia distribuita in un singolo livello.

Prima di parlare dell'architettura single-tier è necessario espandere il concetto di *"tier"*. I tier possono essere visti come dei layer astratti dell'applicativo in questione; tali layer sono suddivisi in 3 parti:

- **Presentation Layer (User Interface)**, informalmente conosciuto come `Livello Client` o `Interfaccia Utente`. La funzione di questo livello è quella di organizzare e mostrare i risultati forniti dall'applicativo tramite un'interfaccia che l'utente che utilizza l'applicativo può comprendere;
- **Application Layer (Backend Login)**. è il livello che si occupa della logica dell'applicativo (denominata "business logic") e ha lo scopo di processare i dati tra il **Presentation Layer** e il **Data Layer**, prendendo i dati dal **Data Layer** e inviandoli al **Presentation Layer** elaborandoli a seconda dei requisiti di quest'ultimo;
- **Data Layer (Database)** è il layer che si occupa della conservazione delle informazioni e si occupa di mandare quest'ultima all' **Application Layer**. Il Data Layer si connette al database e in seguito performa le azioni richieste, denominate **CRUD Operations**[7]

In un'applicatione basata su architettura single-tier la logica del backend, database e frontend sono implementate sulla stessa macchina. Un esempio di applicazione che adotta un architettura single-tier è la suite di *Microsoft Office*.
Infine è importante considerare i pro e i contro che un'architettura single-tier porta con se:

- Pros:
  - Assenza di latenza nella rete dato che l'intero applicativo si trova su una singola macchina;
  - Reperibilità delle risorse assicurata dato che si trovano sulla stessa macchina e quindi non sono necessarie richieste a database locati su server remoti;
  - Minore manutenzione in termini di sicurezza dato che tutte le componenti si trovano su una singola macchina.
- Contro:
  - Singolo punto di rottura centrale: essendo l'intero servizio presente su una sola macchina in caso di malfuzionamenti l'intero applicativo smetterà di funzionare se l'applicativo non è duplicato su macchine *"mirror"*;
  - Performance possibilmente ridotte trattandosi di una struttura monolitica;
  - Scalabilità ridotta;

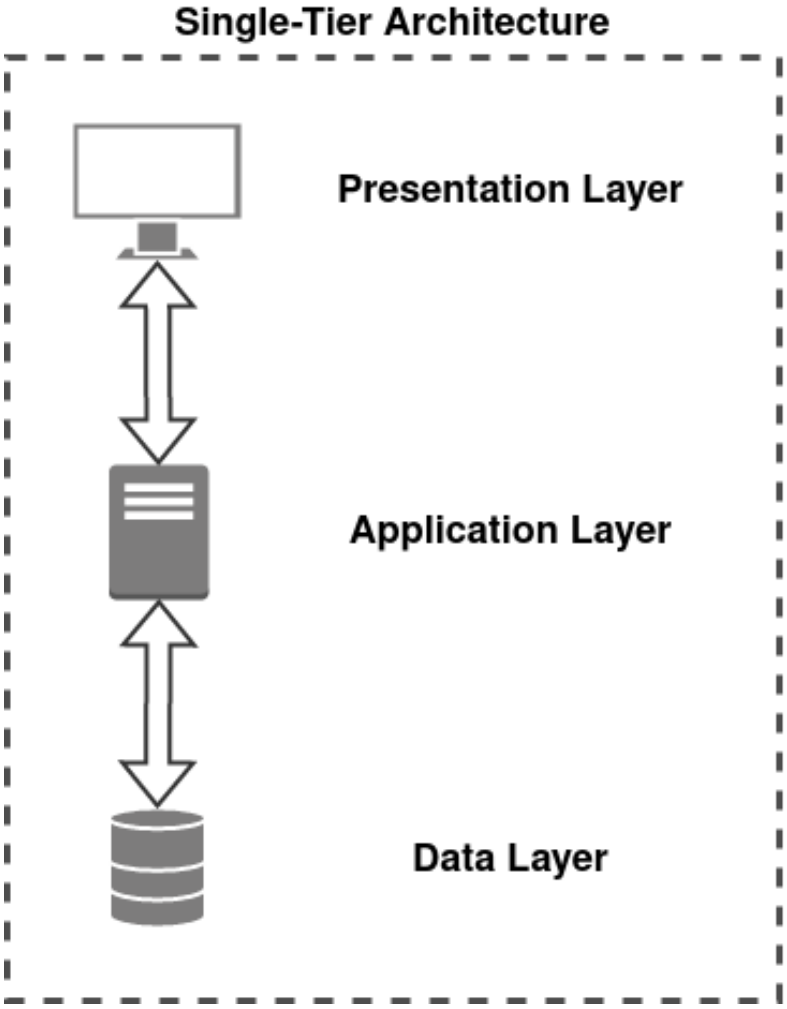

Figura 3.1. Selezione delle risorse con l'architettura single-tier

– Applicativo non sotto il controllo dello sviluppatore, difatti in un architettura del genere l'applicativo è generalmente pacchettizzato, caricato su una piattaforma di distribuzione e infine scaricato dall'utente;

– L'applicativo ha più probabilità di essere sottoposto a reverse engineering.

Approccio basato su Stream L'approccio utilizzato è quello basato sull'utilizzo di **stream di dati** per passare l'input generato dall'utente al backend dell'applicativo. Difatti la versione originale di **sigma**[9] richiedeva il salvataggio di un file locale contenente l'input dell'utente che veniva poi passato a **sigma** per la generazione della query desiderata. Attraverso l'utilizzo dello stream di dati invece si evita il salvataggio del suddetto input limitando le operazioni di I/O, limitandosi all'elaborazione esclusivamente in memoria primaria.

3.2 Generazione delle regole in formato YAML

Come accennato in precedenza **sigma**[9] per funzionare deve prendere come input un file il quale deve contenere una regola ben strutturata in formato YAML. Tale regola deve essere strutturata con precisi campi ben specificati all'interno della documentazione[10] di sigma stesso. Una delle limitazioni del formato richiesto per sigma è che non tutti i campi possono essere usati globalmente in quanto potrebbero per certe tecnologie portare ad errori durante la fase di generazione della query.
Segue un esempio di regola in formato YAML per sigma correttamente strutturata:

```yaml
title: Suspicious DNS Query with B64 Encoded String
id: 4153a907-2451-4e4f-a578-c52bb6881432
status: experimental
description: Detects suspicious DNS queries using base64 encoding
author: Florian Roth
date: 2018/05/10
modified: 2021/08/09
references:
    - https://github.com/krmaxwell/dns-exfiltration
logsource:
    category: dns
detection:
    selection:
        query|contains: '==.'
    condition: selection
falsepositives:
    - Unknown
level: medium
tags:
    - attack.exfiltration
    - attack.t1048.003
    - attack.command_and_control
    - attack.t1071.004
```

Snippet 3.1 Detection di connessioni DNS sospette con codifica base64

3.3 Pacchettizzazione e distribuzione cross-platform

Al fine di garantire la massima portabilità tra diversi sistemi operativi e gestire le dipendenze richieste per il funzionamento del programma senza intaccare l'integrità del sistema stesso si è deciso adottare la tecnologia denominata **Docker**[6].

3.3.1 Cos'è Docker

Docker è una piattaforma open con la quale è possibile lo sviluppo, distribuzione ed esecuzione di applicazioni. Docker consente di separare le applicazioni dall'infrastruttura, in modo da poter distribuire rapidamente il software. Con Docker è possibile gestire l'infrastruttura nello stesso modo in cui si gestiscono le applicazioni. Sfruttando le metodologie di Docker per il deploy, il testing e la distribuzione del codice, è possibile ridurre in modo significativo il ritardo tra la scrittura del codice stesso e la sua esecuzione in produzione.
Docker fa utilizzo di un'architettura client-server. Il client Docker parla con il demone Docker, che si occupa di costruire, eseguire e distribuire i contenitori Docker. Il client e il demone Docker possono essere eseguiti sullo stesso sistema, oppure è possibile collegare un client Docker a un demone Docker remoto. Il client e il **demone Docker** comunicano utilizzando un'**API REST**, tramite socket UNIX o un'interfaccia di rete. Un altro client Docker è Docker Compose, che consente di lavorare con applicazioni costituite da un insieme di container.

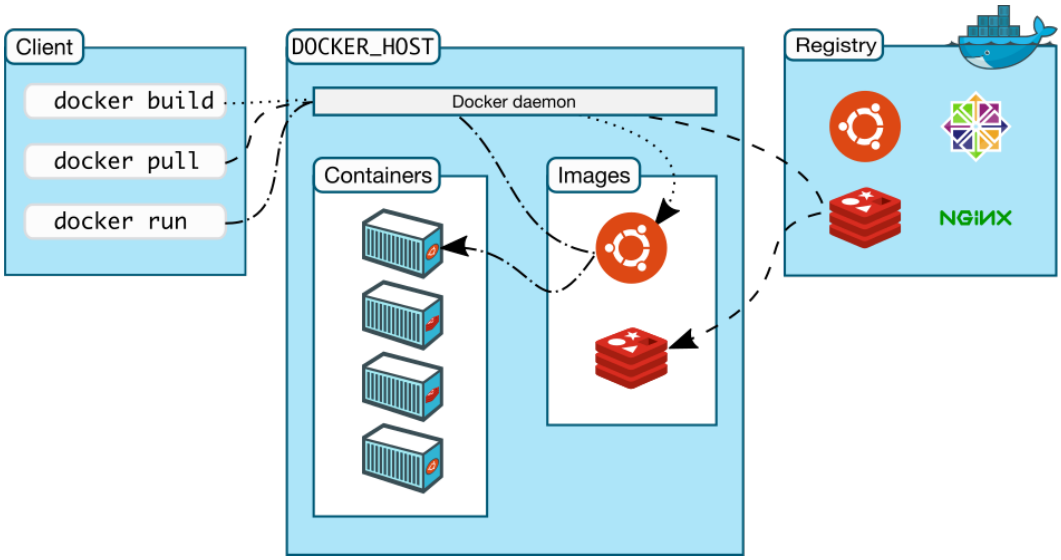

Figura 3.2. Architettura di Docker

Capitolo 4

Implementazione

4.1 Data tier

Per lo sviluppo di tale progetto non si è reso necessario l'utilizzo di una tecnologia Database in quanto grazie all'utilizzo degli **stream** tutta la procedura avviene in memoria, senza la necessità di salvare l'input dell'utente.

4.1.1 Cosa sono gli stream

Uno stream è un array i cui elementi possono essere utilizzati in parallelo, in modo simile al calcolo SIMD. Nella programmazione tramite stream, i dati vengono raccolti dalla memoria, elaborati e infine dispersi dallo stream alla memoria. La latenza della memoria viene quindi ridotta al minimo accedendo ai dati in blocchi, in modo simile al funzionamento della memoria cache.

4.2 Application tier

In questo modulo viene gestita la parte logica ed applicativa della piattaforma, tra cui la gestione del **server HTTP** scritto in **Flask**[14], il passaggio dei dati alla versione modificata di sigma e i vincoli derivanti dai requisiti per la creazione della query; come ad esempio la possibilità di richiedere la conversione per una data regola a patto che parametri fondamentali come la regola **YAML**[2], scelta della tecnologia SIEM e della configurazione ad essa associata siano definiti all'interno della richiesta.

4.2.1 Il framework Flask

Il linguaggio utilizzato è **Python** accoppiato a **Flask**, un micro-framework per lo sviluppo di applicazioni web, scritto in Python, basato sullo strumento **Werkzeug WSGI** e con il motore di template **Jinja2**[15].

Server HTTP in Flask Creare un server HTTP in Flask risulta essere un'operazione estremamente semplice grazie alla natura del framework stesso. Flask grazie ad una funzionalità chiamata *"routing"* permette la definizione di percorsi raggiungibili rendendo disponibili le varie risorse con sole poche righe di codice. (v. 4.1)

Nell'attuale progetto un singolo file in Python si occupa della gestione del **server web** ed esegue lato backend controlli sui campi inviati dall'utente garantendo la correttezza degli stessi prima di effettuare la chiamata al motore di sigma modificato presente su un diverso file. Il seguente listato mostra il codice scritto per la gestione del server web appena descritto:

```
from flask import Flask, render_template, url_for, request
import custom_sigmac

# The name of Flask Application
sigma_app = Flask(__name__)

#
    ------------------------------------------------------------------------


# ROUTES FOR REACHABLE PAGES

# Homepage
@sigma_app.route("/", methods=["GET"])
@sigma_app.route("/homepage", methods=["GET"])
@sigma_app.route("/index", methods=["GET"])
def homepage():
    return render_template("index.html")

# Documentation page
@sigma_app.route("/doc", methods=["GET"])
@sigma_app.route("/documentation", methods=["GET"])
def doc():
    return render_template("documentation.html")

#
    ------------------------------------------------------------------------


# ROUTES FOR ERROR HANDLING

# 404 error handler
@sigma_app.errorhandler(404)
def error_handler(error):
    return render_template("404.html")

#
    ------------------------------------------------------------------------


# ROUTES FOR BACKEND FUNCTIONALITIES (NO REACHABLE HTML PAGES)
@sigma_app.route("/convert", methods=["POST"])
def sigmaconverter():

    srule = str(request.form.get('srule'))           # the sigma
    rule generated by front-end
    target  =  request.form.get('target')       # this field go
    with `-t` option
    config = request.form.get('config')         # this field go
    with `-c` option
```

```
43      # Check for missing parameters that are required for sigma
        basic usage
44      if(srule==None or target==None or config==None):
45          missing_params = []
46          error = "<h1>The following parameter/s missing:\n"
47          if srule==None:
48              error += "Sigma Rule\n"
49          if target==None:
50              error += "Target\n"
51          if config==None:
52              error += "Configuration\n"
53          error += "</h1>"

55          return error

57      result = custom_sigmac.CustomSigmac([str(srule)], target, [
        config+".yml"]).convert()
58      print("RESULT: ",result)
59      return result #DEBUG


62  #
        ----------------------------------------------------------------------


64  # Use this for DEVELOPMENT DEPLOYMENT
65  if __name__ == "__main__":
66      sigma_app.run(debug=True) # Use this command for DEVELOPMENT
        DEPLOYMENT

68  # Use the next commands for PRODUCTION DEPLOYMENT
69  #   from waitress import serve
70  #   serve(sigma_app, host="0.0.0.0", port=8080)
```

Snippet 4.1 Richieste del server Flask al motore modificato di `sigma`

4.2.2 Approccio basato su Stream

L'approccio utilizzato è quello basato sull'utilizzo di **stream di dati** per passare l'input generato dall'utente al backend dell'applicativo. Difatti la versione originale di `sigma` richiedeva il salvataggio di un file locale contenente l'input dell'utente che veniva poi passato a `sigma` per la generazione della query desiderata. Attraverso l'utilizzo dello stream di dati invece si evita il salvataggio del suddetto input azzerando le operazioni di I/O precedentemente coinvolte nel processo, limitandosi dunque esclusivamente all'elaborazione in memoria primaria.

Per implementare l'approccio su stream si è fatto ricorso al modulo `io`[11] presente nella **Standard Library** di Python. Il modulo `io` permette la gestione di vari tipi di I/O.
Esistono tre rappresentazioni principali di I/O:

- I/O testuale;
- I/O binario;

- I/O grezzo.

Indipendentemente dalla sua categoria, ogni oggetto di tipo stream avrà anche varie caratteristiche come i permessi che possono essere *read-only*, *write-only* oppure di *read-write*. È possibile inoltre interagire con gli stream tramite un **accesso casuale arbitrario** (cercando in avanti o indietro in qualsiasi posizione), o solo un **accesso sequenziale** (ad esempio nel caso di un socket o di una pipe).

È importante considerare che gli stream sono sensibili al tipo di dati che trattano. Per esempio, dare un oggetto `str` al metodo `write()` di uno stream binario solleverà un'eccezione di tipo `TypeError`; lo stesso vale per un oggetto bytes al metodo `write()` di uno stream di testo.

```
...
f = io.StringIO(str(sigmafile))
parser = SigmaCollectionParser(f, sigmaconfigs, rulefilter, "
    sigmafile")
backend.setYmlFileName(str(sigmafile))
results = parser.generate(backend)
...
```

Snippet 4.2 Gestione degli stream tramite il modulo `io`

4.2.3 Core modificato di `sigma`

Come anticipato, si è reso necessario modificare parte della parte principale di `sigma` in modo da poterlo adattare al progetto esistente. Il motore è stato modificato al fine di accettare i dati inviati tramite web form sotto forma di stream di dati.

```
def convert(self):
        cwd = os.path.dirname(__file__)
        scm = SigmaConfigurationManager(paths=[str(cwd+"/static/
    libs/sigma/tools/config")])

        logger = logging.getLogger(__name__)
        if self.debug:    # pragma: no cover
            logging.basicConfig(filename='logs/custom_sigmac.log',
    filemode='a', level=logging.DEBUG)
            logger.setLevel(logging.DEBUG)

        if self.target is None:
            print("No target selected, select one with -t/--target
    ")
            sys.exit(self.ERR_NO_TARGET)

        logger.debug("* Target selected %s" % (self.target))

        rulefilter = None
        if self.filter:
            try:
                rulefilter = SigmaRuleFilter(self.filter)
            except SigmaRuleFilterParseException as e:
                print("Parse error in Sigma rule filter expression
    : %s" %
```

```
22                      str(e), file=sys.stderr)
23                  sys.exit(self.ERR_RULE_FILTER_PARSING)

25          sigmaconfigs = SigmaConfigurationChain()
26          backend_class = backends.getBackend(self.target)

28          if self.config is None:
29              if backend_class.config_required and not self.
    shoot_yourself_in_the_foot:
30                  logger.debug("Configuration for backend not
    provvided, EXITING...")
31                  return "<p>The backend you want to use usually
    requires a configuration to generate valid results. Please
    provide one with --config/-c.\n Please read Documentation for
    the available choices for this backend <p>"
32              if backend_class.default_config is not None:
33                  self.config = backend_class.default_config

35          if self.config:
36              order = 0
37              for conf_name in self.config:
38                  try:
39                      sigmaconfig = scm.get(conf_name[:-4]) # The
    slicing for string is necessary to remove .yml extension
40                      if sigmaconfig.order is not None:
41                          if sigmaconfig.order <= order and not self.
    shoot_yourself_in_the_foot:
42                              logger.debug(f"Configuration for
    backend not provvided, Error: {self.ERR_CONFIG_ORDER}, EXITING
    ...")
43                              return "<p>The configurations were
    provided in the wrong order (order key check in config file)\n
    Please read Documentation for more information about tools use<
    p>"
44                          order = sigmaconfig.order

46                      try:
47                          if self.target not in sigmaconfig.config["
    backends"]:
48                              logger.debug(f"Configuration for
    backend not valid, Error: {self.ERR_CONFIG_ORDER}, EXITING...")
49                              return f"<p>The configuration '{
    conf_name}' is not valid for backend '{self.target}'.\n Please
    read Documentation for more information about valid choices<p>"

51                      except KeyError:
52                          pass

54                      sigmaconfigs.append(sigmaconfig) # Append the
    object created in configuration.py (SigmaConfiguration)

56                  except OSError as e:
57                      logger.debug(f"Failed to open Sigma
    configuration file {conf_name}: {str(e)}, Error: {self.
    ERR_OPEN_CONFIG_FILE}, EXITING...")
58                      return f"<p>Impossible load Sigma
    configuration file.\n check the logs<p>"
```

```
59                 except (yaml.parser.ParserError, yaml.scanner.
    ScannerError) as e:
60                     logger.debug(f"Failed to load Sigma
    configuration: {conf_name}; the file {str(e)} is no valid YAML,
     Error: {self.CONFIG_INVALID_YAML}, EXITING...")
61                     return f"<p>Ops, something gone wrong with
    sigma. Impossible process request becaus sigma rule format is
    invalid\n Please read Documentation and if the error persist
    open an issue<p>"
62                 except SigmaConfigParseError as e:
63                     logger.debug(f"Failed to load Sigma
    configuration: {conf_name}; parsing error: {str(e)}, Error: {
    self.ERR_CONFIG_PARSING}, EXITING...")
64                     return f"<p>Ops, something gone wrong with
    sigma. Impossible process request\n Please read Documentation
    and if the error persist open an issue<p>"

66         if self.output_fields:
67             if self.output_format:
68                 output_fields_rejected = [field for field in self.
    output_fields.split(
69                     ",") if field not in self.allowed_fields]  #
    Not allowed fields
70                 if output_fields_rejected:
71                     fields_rej = ", ".join(output_fields_rejected)
72                     logger.debug(f"Fields not allowed: {fields_rej
    }; Error: {self.ERR_OUTPUT_FORMAT}, EXITING...")
73                     return f"<p>Ops, These fields are not allowed
    (check help for allow field list)t\n Please read Documentation
    and if the error persist open an issue<p>"

75                 else:
76                     output_fields_filtered = [field for field in
    self.output_fields.split(
77                         ",") if field in self.allowed_fields]  #
    Keep only allowed fields
78             else:
79                 logger.debug(f"The '--output-fields' or '-of'
    arguments must be used with '--output-format' or '-oF' equal to
     'json' or 'yaml': {fields_rej}; Error: {self.ERR_OUTPUT_FORMAT
    }, EXITING...")
80                 return f"<p>The '--output-fields' or '-of'
    arguments must be used with '--output-format' or '-oF' equal to
     'json' or 'yaml'\n Please read Documentation and if the error
    persist open an issue</p>"

82         backend_options = BackendOptions(
83             self.backend_option, self.backend_config)
84         backend = backend_class(sigmaconfigs, backend_options)

86         filename_ext = self.output_extention
87         filename = self.output
88         fileprefix = None
89         if filename:
90             if filename_ext:
91                 if filename_ext[0] == '.':
92                     pass
```

```
93                  else:
94                      filename_ext = '.' + filename_ext
95              else:
96                  filename_ext = '.rule'
97
98              if filename[-1:] in ['_', '/', '\\']:
99                  fileprefix = filename
100             else:
101                 try:
102                     out = open(filename, "w", encoding='utf-8')
103                 except (IOError, OSError) as e:
104                     logger.debug(f"Failed to open output file: {
    filename}: {str(e)}; Error: {self.ERR_OUTPUT}, EXITING...")
105                     return f"<p>Failed to open output file</p>"
106         else:
107             out = sys.stdout
108
109         error = 0
110         output_array = []
111         result = backend.initialize()      #Initialize
112         if result:
113             print(result, file=out)
114
115         for sigmafile in self.get_inputs(self.rules, self.recurse)
    :
116             logger.debug("* Processing Sigma input %s" % (
    sigmafile))
117             success = True
118             try:
119                 f = io.StringIO(str(sigmafile))
120                 parser = SigmaCollectionParser(f, sigmaconfigs,
    rulefilter, "sigmafile")
121                 backend.setYmlFileName(str(sigmafile))
122                 results = parser.generate(backend)
123
124                 nb_result = len(list(copy.deepcopy(results)))
125                 inc_filenane = None if nb_result < 2 else 0
126
127                 newline_separator = '\0' if self.print0 else '\n'
128                 results = list(results) # Since results is an
    iterator and used twice we convert it a list
129                 for result in results:
130                     if not fileprefix == None and not inc_filenane
     == None:  # yml action
131
132                         try:
133                             filename = fileprefix + str(sigmafile.
    name)
134                             filename = filename.replace(
135                                 '.yml', '_' + str(inc_filenane) +
    filename_ext)
136                             inc_filenane += 1
137                             out = open(filename, "w", encoding='
    utf-8')
138                         except (IOError, OSError) as e:
139                             logger.debug(f"Failed to open output
    file: {filename}: {str(e)}; Error: {self.ERR_OUTPUT}, EXITING
```

```
...")
140                             return f"<p>Failed to open output file
</p>"

142                     elif not fileprefix == None and inc_filenane =
= None:  # a simple yml
143                         try:
144                             filename = fileprefix + str(sigmafile.
name)
145                             filename = filename.replace('.yml',
filename_ext)
146                             out = open(filename, "w", encoding='
utf-8')
147                         except (IOError, OSError) as e:
148                             logger.debug(f"Failed to open output
file: {filename}: {str(e)}; Error: {self.ERR_OUTPUT}, EXITING
...")
149                             return f"<p>Failed to open output file
</p>"

151                     if not self.output_fields:
152                         return result
153                         print(result, file=out, end=
newline_separator)

155                 if self.output_fields:  # Handle output fields
156                     output = {}
157                     f.seek(0)
158                     docs = yaml.load_all(f, Loader=yaml.FullLoader
)
159                     for doc in docs:
160                         for k, v in doc.items():
161                             if k in output_fields_filtered:
162                                 output[k] = v
163                     output['rule'] = [result for result in results
]
164                     if len(output['rule']) > 0:  # avoid printing
empty rules
165                         if "filename" in output_fields_filtered:
166                             output['filename'] = str(sigmafile.
name)
167                         output_array.append(output)

169                 if nb_result == 0:  # backend get only 1 output
170                     if not fileprefix == None:  # want a prefix
anyway
171                         try:
172                             filename = "%s%s_mono_output%s" % (
173                                 fileprefix, self.target,
filename_ext)
174                             out = open(filename, "w", encoding='
utf-8')
175                             fileprefix = None  # no need to open
the same file many time
176                         except (IOError, OSError) as e:
177                             logger.debug(f"Failed to open output
file: {filename}: {str(e)}; Error: {self.ERR_OUTPUT}, \n {e.
```

```
    strerror} \n EXITING...")
178                             return f"Failed to open output file"

180             except OSError as e:
181                 logger.debug(f"* Convertion Sigma input sigmafile
    FAILURE; Error: {self.ERR_OPEN_SIGMA_RULE},\n {e.strerror} \n
    EXITING...")
182                 success = False
183                 return f"Failed to convert sigma rule. Error: {
    self.ERR_OPEN_SIGMA_RULE}"

185             except (yaml.parser.ParserError, yaml.scanner.
    ScannerError) as e:
186                 logger.debug(f"Error: Sigma file sigmafile is no
    valid YAML: {str(e)}; Error: {self.ERR_INVALID_YAML}, \n {e.
    strerror} \n EXITING...")
187                 success = False
188                 if not self.defer_abort:
189                     return f"Failed to convert sigma rule
    sigmafile"

191             except (SigmaParseError, SigmaCollectionParseError) as
     e:
192                 logger.debug(f"* Convertion Sigma input sigmafile
    FAILURE; Error: {self.ERR_SIGMA_PARSING}, EXITING...")
193                 if not self.defer_abort:
194                     return f"* Convertion Sigma input sigmafile
    FAILURE; Error: {self.ERR_SIGMA_PARSING}, EXITING..."

196             except NotSupportedError as e:
197                 logger.debug(f"Error: The Sigma rule requires a
    feature that is not supported by the target system: {str(e)}")
198                 success = False
199                 if not self.ignore_backend_errors:
200                     if not self.defer_abort:
201                         return f"<p>Error: The Sigma rule requires
     a feature that is not supported by the target system: {self.
    ERR_NOT_SUPPORTED}</p>"
202             except BackendError as e:
203                 logger.debug(f"Error: Backend error in sigmafile:
    {str(e)}")
204                 success = False
205                 if not self.ignore_backend_errors and not self.
    defer_abort:
206                     return f"<p>Error: Backend error in sigmafile.
     Error: {self.ERR_BACKEND}</p>"

208             except (NotImplementedError, TypeError) as e:
209                 logger.debug(f"An unsupported feature is required
    for this Sigma rule sigmafile: {str(e)} ")
210                 # traceback.print_exc()
211                 logger.debug("* Convertion Sigma input sigmafile
    FAILURE" )
212                 success = False
213                 if not self.ignore_backend_errors and not self.
    defer_abort:
214                     return f"<p>* Convertion Sigma input sigmafile
```

```
        FAILURE, Error: {self.ERR_NOT_IMPLEMENTED}</p>"

216             except PartialMatchError as e:
217                 logger.debug("Error: Partial field match error: %s
    " %
218                     str(e), file=sys.stderr)
219                 logger.debug("* Convertion Sigma input sigmafile
    FAILURE" )
220                 success = False
221                 if not self.ignore_backend_errors and not self.
    defer_abort:
222                     return f"{self.ERR_PARTIAL_FIELD_MATCH}"
223             except FullMatchError as e:
224                 print("Error: Full field match error", file=sys.
    stderr)
225                 logger.debug("* Convertion Sigma input sigmafile
    FAILURE")
226                 success = False
227                 if not self.ignore_backend_errors and not self.
    defer_abort:
228                     return f"<p>* Convertion Sigma input sigmafile
     FAILURE, Error: {self.ERR_FULL_FIELD_MATCH}</p>"
229             finally:
230                 try:
231                     f.close()
232                 except:
233                     pass

235             if success:
236                 logger.debug("* Convertion Sigma input sigmafile
    SUCCESS")

238         result = backend.finalize()
239         if result:
240             print("PRINT THE RESULT ",result)
241             print(result, file=out)

243         if self.output_fields:
244             if self.output_format == 'json':
245                 print(json.dumps(output_array, indent=4,
246                     ensure_ascii=False), file=out)

248         out.close()
249         sys.exit(error)
```

Snippet 4.3 Motore di `sigma` modificato

4.3 Presentation tier

In questo modulo è stata implementata la **UI (User Interface)**, ovvero il mezzo di comunicazione tra l'utente finale e i moduli precedenti. In questo caso si tratta di pagine web che presentano le informazioni e guidano l'utente nell'utilizzo delle funzionalità della piattaforma; tramite l'utilizzo del **Presentation Tier** vengono

inviate le richieste HTTP inviate all'Application tier il quale comunica le conseguenti risposte di conferma o di errore.

4.3.1 Approccio basato su creazione dinamica della regola tramite Javascript

L'utilizzo del linguaggio **Javascript**[1] si è reso necessario per rendere il frontend dell'applicativo dinamico a fronte dei cambiamenti apportati alla pagina durante la generazione della regola tramite l'editor, oltre alla gestione dei diversi controlli e funzionalità dell'applicativo.

Vanilla JS La scelta di Vanilla JS è guidata dal fatto di mantenere la semplicità nell'ambito dello sviluppo web; difatti ad oggi vi è un largo utilizzo di Framework Javascript molto performanti, ognuno con le proprie peculiarità, ma che portano spesso con se funzionalità non necessarie per lo sviluppo di un dato progetto. Oltre a ciò tali Framework portano con se molte dipendenze e spesso la complessità di tali progetti può creare bug o incompatibilità con dei moduli Javascript custom scritti per l'applicativo.

Tra i motivi principali per utilizzare VanillaJS troviamo quindi:

- Completo controllo sul funzionamento del codice che si scrive;
- Assenza di funzionalità non necessarie per l'applicativo;
- Massima compatibilità con tutti i motori Javascript distribuiti tra i vari browser;
- Assenza di conflitti tra componenti black-box.

4.3.2 Richieste AJAX

Per effettuare lo scambio di dati con l'Application tier, e quindi le richieste alle API HTTP, è stato adottato un approccio asincrono tramite richieste **AJAX (Asynchronous JavaScript and XML)**[3]. Questo significa che la richiesta non sarà bloccante per l'interfaccia e che i dati saranno aggiornati nel momento in cui arriverà la risposta dal server, senza esplicito ricaricamento della pagina web. Nello specifico ogni componente gestisce e mostra un proprio stato di attesa, così che al completamento delle richieste effettuate si aggiorni dinamicamente in base al responso del server.

4.3.3 Core di sigma modificato

Per permettere il corretto funzionamento dell'applicativo si è resa necessaria la modifica del core del progetto **sigma**[9], difatti tale progetto nasce per un utilizzo tramite **Command Line Interface (CLI)**. Per ridurre la superficie di attacco in ambito della sicurezza si è evitato di utilizzare approcci basati su syscall, come ad esempio tramite l'utilizzo dei moduli `os`[12] o `submodule`[13] offerti da Python.

4.3.4 Le pagine realizzate

Per quanto riguarda lo stile delle pagine e dei widget è stato utilizzato **BULMA**[17] un Framework CSS Javascript-free. La scelta di tale framework ha permesso di consentire la massima compatibilità sul progetto esistente senza creare conflitti con il codice Javascript custom scritto per funzionaltà presenti all'interno del front-end dell'applicativo. Inoltre, è possibile estendere il framework stesso modificando il file **SASS**[5] associato omettendo tutte quelle componenti non necessarie per il progetto. SASS è un'estensione del classico CSS, per definire gli stili più specifici.

Editor delle regole

L'editor delle regole si presenta con una struttura HTML di base con campi obbligatori per la regola non eliminabili dall'utente. Grazie poi all'utilizzo di Javascript l'utente può ampliare la struttura della regola stessa definendo campi custom opzionali aumentando la complessità della struttura ad albero.

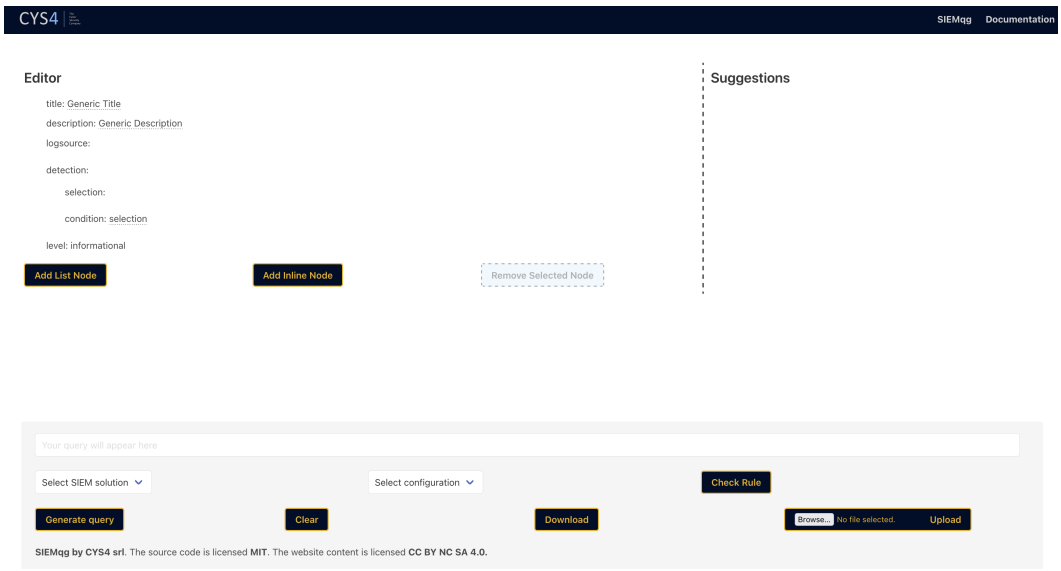

Figura 4.1. Pagina principale - Editor delle regole

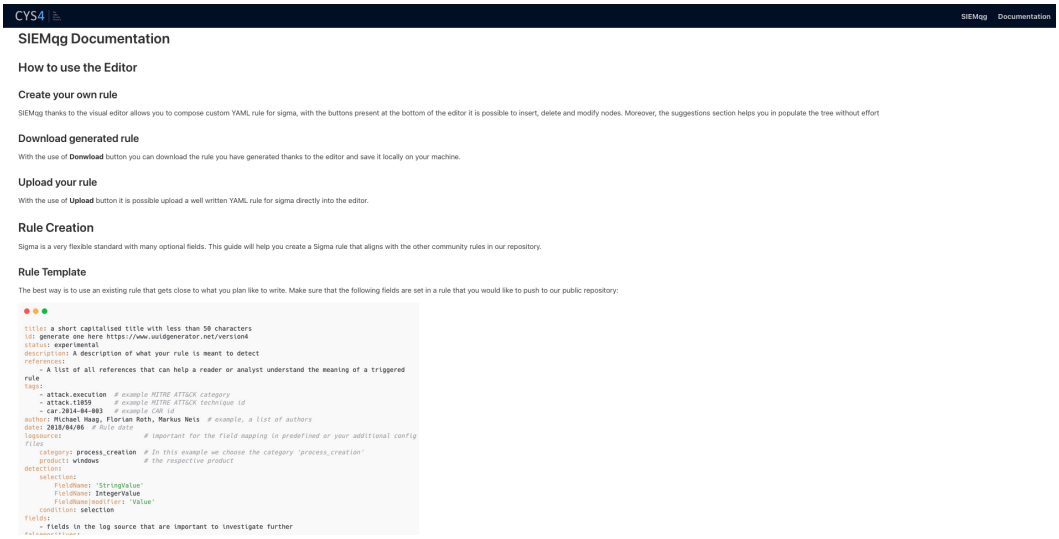

Figura 4.2. Pagina di documentazione per l'utilizzo dell'applicativo

Selezione della tecnologia SIEM e configurazione della stessa

Al fine di generare una corretta query `sigma` necessita di conoscere la tecnologia SIEM in uso e la relativa configurazione ad essa associata. Ogni tecnologia SIEM possiede multiple scelte in termini di configurazione riguardante il motore di ingest come ad esempio per `sysmon`, `powershell` e molte altre. È importante notare che differenti configurazioni possono portare la generazione di query sintatticamente differenti sulla stessa tecnologia SIEM. Nonostante ciò la differenza che si presenta è solamente sintattica in quanto ogni query generata e semanticamente coerente tra le varie configurazioni o tecnologie SIEM.

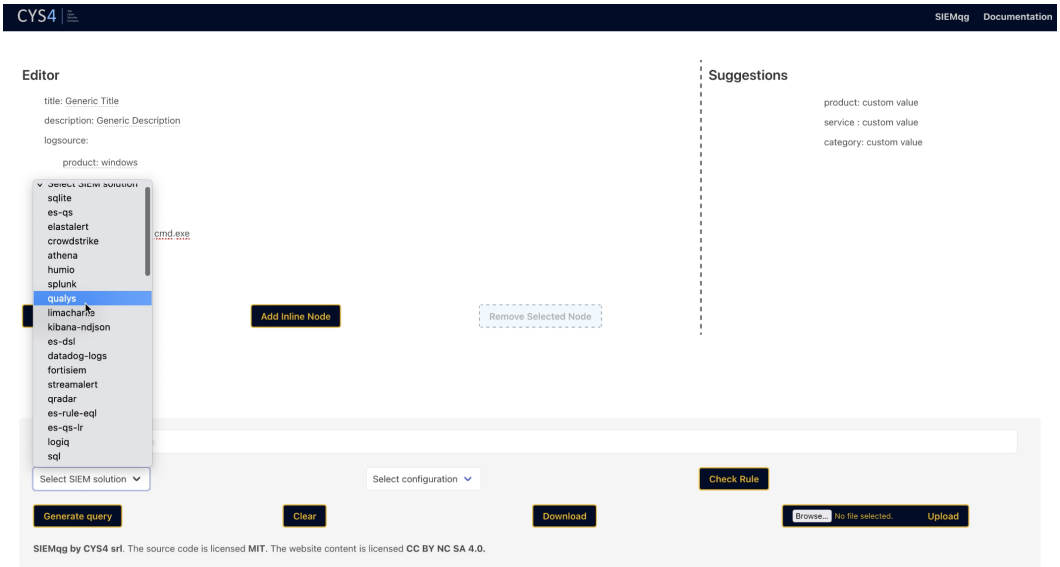

Figura 4.3. Scelta della tecnologia SIEM

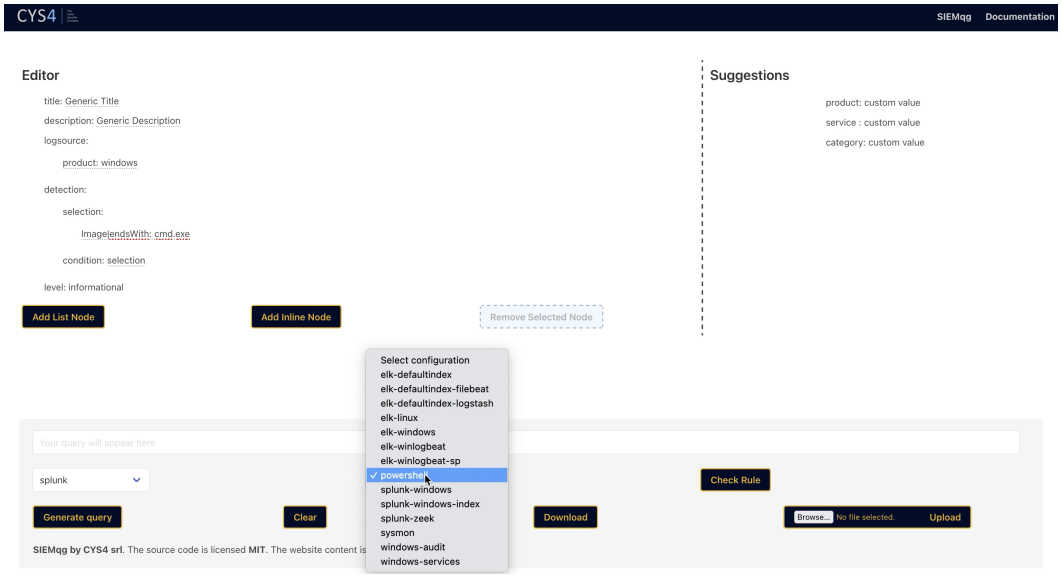

Figura 4.4. Scelta della della configurazione per la tecnologia SIEM

4.3.5 Upload della regola YAML

Tramite l'utilizzo della funzionalità di Upload della regola il sistema apre un prompt che permette la selezione di un file sul PC locale della regola YAML. Quello che accade è che la regola caricata dopo un opportuna parsing viene caricata e sovrascrive la struttura HTML nell'editor presente.

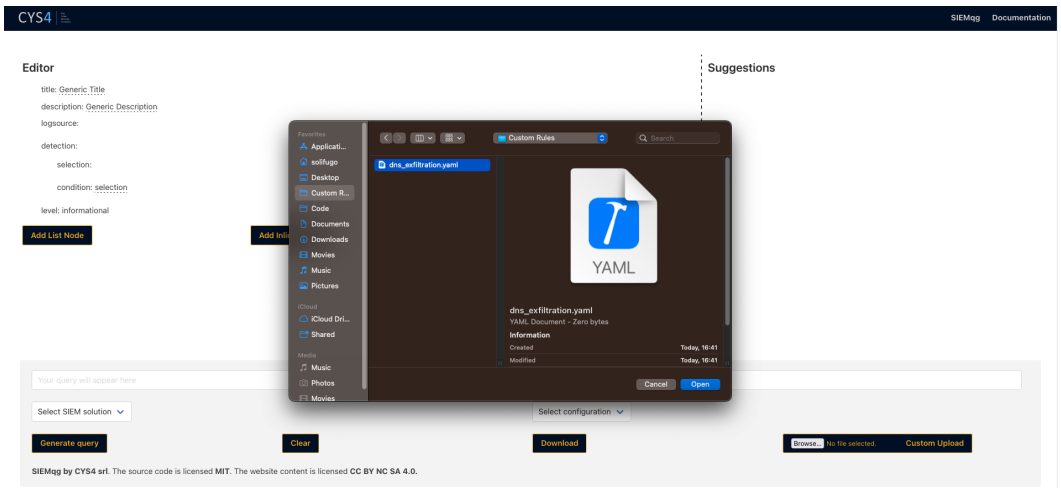

Figura 4.5. Caricamento di una regola da filesystem locale

4.3.6 Donwload della regola YAML

Tramite l'utilizzo della funzionalità di Download della regola fruibile tramite un apposito elemento HTML di tipo button con un azione Javascript collegata, l'utente

può effetture il download della regola presente nell'editor. La funzione Javascript scritta si occupa di effettuare un parsing della regola presente nell'editor scrivendola in un file in formato YAML che verrà poi scaricato automaticamente.

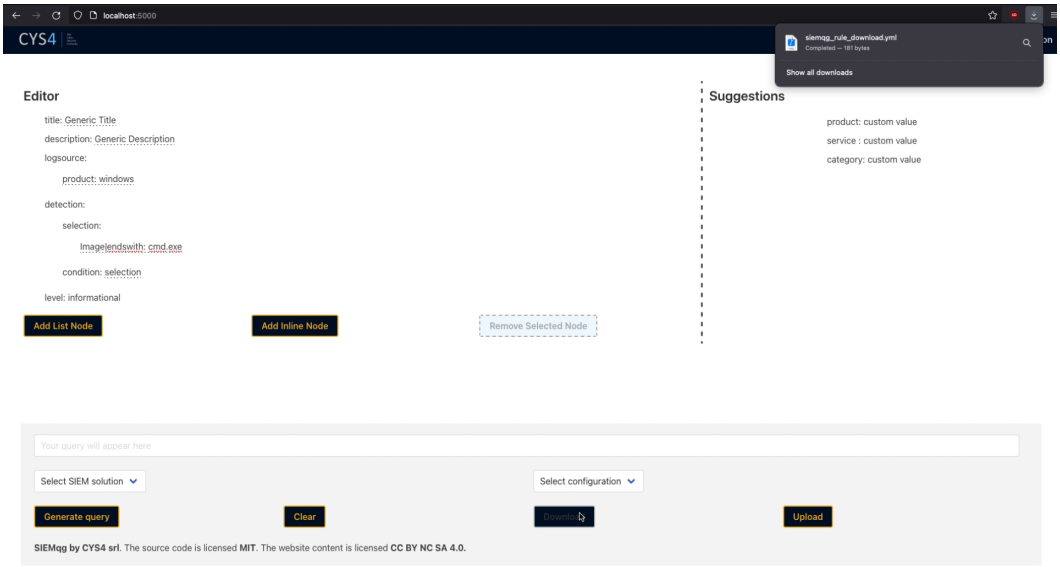

Figura 4.6. Della della configurazione per la tecnologia SIEM

Gestione delle richieste effettuate

Le richieste per la generazione di una query vengono effettuate tramite chiamate asincrone utilizzando AJAX, in questo modo data la natura non bloccante delle stesse, l'utente potrà continuare ad utilizzare liberamente l'applicativo. Una volta che la query sarà stata prodotta la pagina del frontend aggiornerà esclusivamente la porzione di pagina responsabile ad ottenere la query ritornata dal backend.

```
1  //Send the proper header information along with the request
2  http.setRequestHeader('Content-type', 'application/x-www-form-
       urlencoded');

4  http.onreadystatechange = function() {//Call a function when the
       state changes.
5    if(http.readyState == 4   http.status == 200) {
6      document.getElementById("qresult").value = http.responseText;
7    }
8    else if(http.readyState == 4   http.status == 500){
9      alert("The configuration selected is not currently implemented
        correctly in sigma. Please use different configuration...");
10   }
11 }
12 http.send(params);
13 })
```

Snippet 4.4 Porzione di codice responsabile delle chiamate AJAX al backend

4.4 Definizione del Dockerfile per il progetto

Come accennato in precedenza si è scelto l'utilizzo della tecnologia dei container Docker. A questo scopo si è resa necessaria la scrittura di un Dockerfile per il progetto stesso. Un `Dockerfile` è un documento di testo che contiene tutti i comandi che un utente potrebbe richiamare alla riga di comando per assemblare un'immagine. Di seguito viene listato il suddetto file.

```
1 FROM python:3.8.15-bullseye

3 COPY . /sigma_app
4 RUN git clone https://github.com/SigmaHQ/sigma.git
5 RUN mv /sigma/tools/sigma /usr/local/lib/python3.8/site-packages/
6 RUN rm -rf sigma
7 RUN pip install coverage==5.0 yamllint==1.21 elasticsearch==7.6
      elasticsearch-async==6.2 pytest==5.4 colorama setuptools stix2
      attackcti pipfile requests==2.25 urllib3==1.26 progressbar2==3
      .47 pymisp==2.4.123 PyYAML==5.1 ruamel.yaml flask waitress

9 EXPOSE 5954

11 CMD ["python", "/sigma_app/src/sigma_server.py"]
```

Snippet 4.5 Dockerfile

Capitolo 5

Conclusioni

In questa relazione è stato descritto il lavoro svolto durante lo sviluppo di un Tool interno all'azienda **CYS4** presentando la necessità di una piattaforma web che permetta un facile utilizzo di tool di detection delle minacce in ambito Cyber Security.

Prima dello sviluppo di questo progetto era possibile utilizzare lo strumento originale (sigma) portando però notevoli difficoltà per l'analista in termini di configurazione dell'ambiente di lavoro.

Tra gli aspetti che sono stati migliorati troviamo quindi la possibilità di avere un interfaccia visuale per lo sviluppo delle regole e la creazione delle query specifiche per la tecnologia SIEM necessarie, rimozione della configurazione necessaria per predisporre il corretto funzionamento dello strumento, portabilità su ogni tipo di sistema.

L'aver realizzato la piattaforma con un approccio modulare ed estendibile è infine da considerare come il principale vantaggio tecnico. È stata infatti ridotta al minimo la dipendenza tra i vari livelli dell'architettura, così da rendere possibile facilmente future modifiche e nuove implementazioni. Il fatto che i dati vengano messi a disposizione tramite chiamate API lascia massima libertà sulla scelta delle tecnologie per lo sviluppo dell'interfaccia utente, consentendo ad esempio la creazione di un'equivalente applicazione mobile con le stesse funzionalità della versione web.

Durante lo sviluppo del progetto ho avuto modo di applicare alcuni dei concetti appresi nei vari corsi di studio, dalla gestione delle varie fasi di progettazione all'applicazione di buone tecniche e paradigmi di programmazione. Mi ha dato inoltre l'opportunità di imparare e approfondire alcune tecnologie e approcci, migliorando la mia conoscenza ed esperienza di progettazione e sviluppo per il web.

5.1 Sviluppi futuri

Sviluppi futuri potrebbero comprendere l'integrazione di più tecnologie utilizzate in ambito Defense nella Cyber Security come ad esempio YARA.

Cos'è YARA? **YARA (Yet Another Ridiculous Acronym)** è una tecnologia di sicurezza open-source che aiuta gli analisti a eseguire il pattern matching sui file.

Quello che Snort fa per i pacchetti a livello di rete e Sigma per i log, YARA lo fa per i file.

Autocompletamento delle regole in fase di creazione La funzione di autocompletamento delle regole potrebbe espandere il già presente sistema di suggerimenti delle scelte effettuabili dall'utente durante la creazione della regola. Tale funzionalità dovrebbe lavorare a runtime analizzando la struttura della regola ad ogni cambiamento in modo tale da poter formulare la migliore versione possibile in termini di autocompletamento

Autocorrezione semantica delle regole create Grazie all'implementazione di un sistema di autocorrezione semantica delle regole, il sistema potrebbe prevenire la creazione e sottomissione di regole che creerebbero errori logici durante l'esecuzione di sigma, evitando così la generazione di eventuali query non funzionanti.

Valutare tecnologie alternative Essendo stato lo sviluppo e il testing della piattaforma svolto solamente in locale, in seguito al rilascio ufficiale si potrebbero effettuare delle nuove valutazioni sulle tecnologie utilizzate e sulle scelte effettuate. Ad esempio sulla tipologia di framework per lo sviluppo dell'Application tier. Ci possono essere molteplici motivi per cui un cambiamento di questo tipo potrebbe rendersi necessario, ad esempio per un discorso di performance o di consumi. Importante è quindi aver progettato il sistema per adattarsi facilmente anche a modifiche che comporterebbero la sostituzione di un intero livello architetturale (v. sezione 3.1).

5.2 Ringraziamenti

Ringrazio il professor **Angelo Monti** per avermi dedicato il suo tempo affiancandomi nella stesura di questa tesi.
Ringrazio i miei colleghi **Alessio Dalla Piazza** e **Antonio Blescia** per avermi dato l'opportunità di svillupare la tesi per un progetto interno all'azienda e per avermi affiancato durante i momenti di difficoltà.
Un grazie speciale va ai miei genitori, **Ettore** e **Silvia**, per avermi sempre supportato nella scelta di questo mio percorso, per aver sempre creduto in me e per avermi aiutato in ogni modo possibile, comprendendomi anche nei momenti di sconforto e festeggiando insieme a me invece nei momenti di trionfo; non potrò mai ringraziare abbastanza la mia famiglia per tutto ciò che ha fatto per me e spero che questo mio traguardo sia anche un po' il loro.
In ultimo ma non per importanza ringrazio **Manuel** e **Dayana**, i miei amici di una vita, sempre pronti a tirarmi su di morale quando le cose andavano male, e sempre presenti nell'ascoltare i miei deliri in materia di cui magari capivano ben poco. E infine un grazie anche a me stesso, perchè alla fine se ho raggiunto questo traguardo lo devo anche un po' alla mia testardagine e perseveranza.

Bibliografia

[1] Eich, B. *Javascript: a programming language for the World Wide Web.* Available from: https://www.javascript.com/.

[2] Evans, C. *YAML: YAML Ain't Markup Language™.* Available from: https://yaml.org/.

[3] Garret, J. J. *AJAX: Asynchronous JavaScript and XML.* Available from: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX.

[4] GNU Public License 3. About The Licenses – GPLv3. Available from: https://www.gnu.org/licenses/gpl-3.0.html.

[5] Hampton Catlin. Sass: Syntactically awesome style sheets. Available from: https://sass-lang.com/.

[6] Hykes, S. *Docker: Accelerated, Containerized Application Development.* Available from: https://www.docker.com/.

[7] James Martin. create, read, update, e delete (crud). Available from: https://it.wikipedia.org/wiki/CRUD.

[8] Mozilla Public License 2.0. About The Licenses – MPLv2.0. Available from: https://www.mozilla.org/en-US/MPL/2.0/.

[9] Patzke, T. *sigma: A generic rule format to express search queries on log data.* Available from: https://github.com/SigmaHQ/sigma.

[10] Patzke, T. *sigma: A generic rule format to express search queries on log data.* Available from: https://github.com/SigmaHQ/sigma/wiki.

[11] Python. *The Python Standard Library: os.* Available from: https://docs.python.org/3/library/io.html.

[12] Python. *The Python Standard Library: os.* Available from: https://docs.python.org/3/library/os.html.

[13] Python. *The Python Standard Library: subprocess.* Available from: https://docs.python.org/3/library/subprocess.html.

[14] Ronacher, A. *Flask: web development, one drop at a time.* Available from: https://flask.palletsprojects.com/en/2.2.x/.

[15] Ronacher, A. *Jinja: a fast, expressive, extensible templating engine.* Available from: `https://pypi.org/project/Jinja2/`.

[16] Team, W. *Wazuh: The Open Source Security Platform.* Available from: `https://www.docker.com/`.

[17] Thomas, J. *Bulma: the modern CSS framework that just works. ™.* Available from: `https://bulma.io/`.

[18] van Rossum, G. *Python: A powerfull, fast and open scripting language.* Available from: `https://docs.python.org/3/`.