ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scuola di Ingegneria ed Architettura
Corso di Laurea Magistrale in Ingegneria Informatica

Tecniche e metodi per favorire nuove strategie di deception per la Cybersecurity

Relatore:
Chiar.mo Prof.
Michele Colajanni

Presentata da:
Antonio Marino

Co-Relatori:
Silvio Russo

Anno Accademico 2023/2024

Indice

Elenco delle figure

Abstract

Nel contesto attuale della cybersecurity, le minacce informatiche sono in costante evoluzione, richiedendo approcci innovativi e proattivi per proteggere le infrastrutture critiche e i dati sensibili. Questa tesi esplora l'uso della deception come strategia avanzata di cyber defense, con un focus particolare sul progetto realizzato, che implementa la creazione di fake database attraverso la conversione e l'inserimento di dati da file XML. Il progetto sfrutta la capacità di generare database simulati che replicano fedelmente le caratteristiche dei database reali, ma che contengono fake data progettati per attirare, e intrappolare gli attaccanti. L'impiego di questo progetto e quindi dei fake database, si rivela estremamente utile in ambito deception, come ad esempio per la creazione di honeypot, il monitoraggio delle attività malevole, ma anche per la simulazione di dati sensibili e per testare le vulnerabilità del sistema senza esporre informazioni reali. Questa tesi analizza in dettaglio le applicazioni e i benefici dei fake database, dimostrando come possano migliorare la resilienza dei sistemi informatici e supportare lo sviluppo di tecnologie di sicurezza avanzate. Vengono discussi vari casi d'uso, tra cui il miglioramento delle procedure di risposta agli incidenti, e il benchmarking della sicurezza. Inoltre, la ricerca evidenzia come l'adozione strategica dei fake database non solo rafforzi la difesa contro le minacce attuali, ma favorisca anche l'innovazione e la collaborazione nel campo della cybersecurity.

Capitolo 1

Introduzione

La cybersecurity è una disciplina essenziale nel panorama digitale contemporaneo, che mira a proteggere sistemi, reti e dati da attacchi, danni o accessi non autorizzati. La triade CIA (Confidentiality, Integrity, Availability) rappresenta i tre principi fondamentali su cui si basa ogni strategia di cybersecurity. La confidenzialità implica proteggere i dati tramite crittografia, rendendoli illeggibili a chi non dispone della chiave di decrittazione. L'integrità dei dati assicura che le informazioni non vengano alterate o distrutte in modo non autorizzato, utilizzando firme digitali e hash crittografici. La disponibilità garantisce che le informazioni e i sistemi siano accessibili agli utenti autorizzati, proteggendoli da attacchi Denial of Service (DoS) e assicurandone la resilienza.

Per raggiungere questi obiettivi, la cybersecurity impiega varie tecniche e pratiche, come la gestione delle identità e degli accessi (IAM), i sistemi di rilevazione delle intrusioni (IDS) e la risposta agli incidenti. Un approccio innovativo nella cybersecurity è la deception. Questa strategia implica la creazione di falsi bersagli, trappole e segnali ingannevoli che mirano a confondere e rallentare gli aggressori, distogliendoli dagli obiettivi reali. Attraverso l'uso di honeypots, honeynets e altre tecniche, le organizzazioni possono raccogliere informazioni sugli attacchi, studiare i comportamenti degli aggressori

e migliorare le loro difese. Per implementare efficacemente le tecniche di deception, è necessario disporre di tecnologie che consentano una gestione flessibile e dinamica delle infrastrutture di sicurezza.

Lo scopo principale di questa tesi è dimostrare come l'integrazione di tecnologie avanzate quali Docker, XML e database (sia relazionali che NoSQL) possa migliorare le strategie di deception nella cybersecurity. In particolare, si intende sviluppare e documentare un approccio che utilizza file XML per l'estrazione e la definizione di architetture di database. Questi database sono progettati non solo per scopi operativi legittimi ma anche per fungere da strumenti di deception, per attirare e tracciare potenziali aggressori. Il processo sviluppato all'interno di questa tesi inizia con il parsing e la manipolazione dei file XML utilizzando la libreria xml.etree.ElementTree per analizzare e navigare la struttura del file. I dati rilevanti vengono estratti dai nodi specifici, come "diagram" e "mxCell", ed eventuali tag HTML vengono rimossi tramite la libreria BeautifulSoup per ottenere testi puliti. Successivamente, viene stabilita una connessione a un database MongoDB tramite pymongo, dove i dati estratti dall'XML vengono inseriti in una collezione MongoDB. Per la gestione dei servizi, viene utilizzato Docker Compose, che orchestra i servizi necessari, come MongoDB e l'applicazione Python. Un file Dockerfile specifica l'ambiente di runtime Python, installando le dipendenze e copiando i file necessari per l'esecuzione dello script. Lo script Python è configurato per essere eseguito automaticamente all'interno del container Docker, garantendo un'implementazione ripetibile e isolata dell'intero processo. Questo metodo mira a creare un ambiente dinamico e flessibile per l'implementazione di tecniche di deception, come honeypots e honeynets, al fine di rilevare, analizzare e contrastare attacchi informatici. La tesi analizza i vantaggi offerti da questo approccio, evidenziando la scalabilità, poiché utilizzando Docker è possibile scalare rapidamente le risorse necessarie per l'implementazione di honeypots, adattandosi a differenti esigenze operative. La portabilità è garantita dai container Docker, che assicura che l'intera configurazione possa

essere trasferita e riprodotta su diversi ambienti senza modifiche significative. Inoltre, l'efficienza nella gestione delle infrastrutture di sicurezza è migliorata grazie alla gestione centralizzata tramite Docker Compose, che facilita il monitoraggio e la manutenzione delle infrastrutture stesse.

L'elaborato di tesi è formato da altri sei capitoli oltre quello introduttivo (Capitolo 1). Inizialmente, viene presentato lo stato dell'arte, analizzando tecnologie e metodologie esistenti nella cybersecurity e nella deception (Capitolo 2). Successivamente, viene descritta l'architettura del sistema proposto, illustrando la struttura e i componenti principali presenti (Capitolo 3), seguita dal capitolo sull'implementazione, che approfondisce i dettagli tecnici del processo di sviluppo (Capitolo 4). Dopo l'implementazione, un ulteriore capitolo è dedicato alla sperimentazione, dove vengono illustrati i test effettuati per valutare il corretto funzionamento del progetto e la sua efficacia (Capitolo 5). Si presentano poi dei possibili casi d'uso per dimostrare l'applicabilità del sistema evidenziando, inoltre, come il sistema possa essere perfezionato e adattato a nuove esigenze (Capitolo 6). Infine, l'ultimo capitolo riassume le principali conclusioni e risultati ottenuti dal progetto (Capitolo 7).

Capitolo 2

Stato dell'arte

Il capitolo in oggetto si configura come un'immersione dettagliata nelle infrastrutture tecnologiche e nelle attuali dinamiche del comparto della deception nella cybersecurity. La focalizzazione sulle tecniche di deception permetterà di esplorare le fondamenta su cui si basano le moderne strategie di deception. Verranno analizzati i principali strumenti e metodologie impiegati per creare ambienti artificiali destinati a rilevare e analizzare attività malevole, nonché i recenti sviluppi tecnologici che stanno rivoluzionando questo settore. Questo capitolo offrirà una panoramica approfondita delle attuali dinamiche e delle best practice nel campo della deception, delineando il loro ruolo nella protezione delle infrastrutture digitali contro le minacce sempre più sofisticate.

2.1 La deception

La deception nella cybersecurity rappresenta un campo in rapida evoluzione, mirato a ingannare, rilevare e analizzare le attività malevole all'interno dei sistemi informatici. Questo approccio si distingue per l'uso di tecniche di deception che creano ambienti artificiali, come honeypots e honeynets, destinati ad attirare e distrarre gli attaccanti, permettendo agli esperti di sicurezza di monitorare le loro azioni senza rischi per le risorse critiche [15]. L'obiettivo

principale di queste tecniche è quello di guadagnare tempo prezioso per la difesa, raccogliere informazioni sugli attacchi e prevenire danni reali[21]. Nel contesto attuale, caratterizzato da minacce sempre più sofisticate e persistenti, la deception si posiziona come una componente di alto rilievo, riguardo le strategie di cyber defense avanzate. La comprensione approfondita delle tecniche di deception e della loro efficacia è essenziale per sviluppare sistemi di difesa robusti e proattivi [2]. L'uso della deception nel contesto della cybersecurity ha radici storiche che risalgono agli anni '90, con l'introduzione dei primi honeypots. Inizialmente, questi sistemi erano semplici trappole che simulavano vulnerabilità basilari per attirare gli attaccanti. Con il tempo, la complessità e la sofisticazione delle tecniche di deception sono cresciute, dando origine a una varietà di strumenti e approcci più avanzati [28]. Ad oggi, esistono vari tipi di deception che possono essere implementati, ciascuno con specifiche funzionalità e scopi. Ad esempio, gli honeypots di alta interazione simulano sistemi completi e possono ingannare gli attaccanti facendogli credere di aver compromesso una vera risorsa, mentre i honeypots di bassa interazione imitano solo i servizi o le parti di un sistema, riducendo i rischi di compromissione completa [27]. La moderna applicazione delle tecniche di deception comprende anche l'uso di honeynets, reti di honeypots configurati per simulare un'intera rete aziendale, e altre tecnologie avanzate come i sistemi di decoy e le trappole per malware. Questi strumenti permettono ai professionisti della sicurezza di ottenere una visione più dettagliata delle tattiche, tecniche e procedure (TTP) utilizzate dagli attaccanti, fornendo dati preziosi per migliorare le difese [31][9]. Inoltre, l'integrazione della deception con altre strategie di difesa, come il threat hunting e il machine learning, rappresenta un'area di ricerca e sviluppo particolarmente promettente [2].

2.1.1 Honeypots

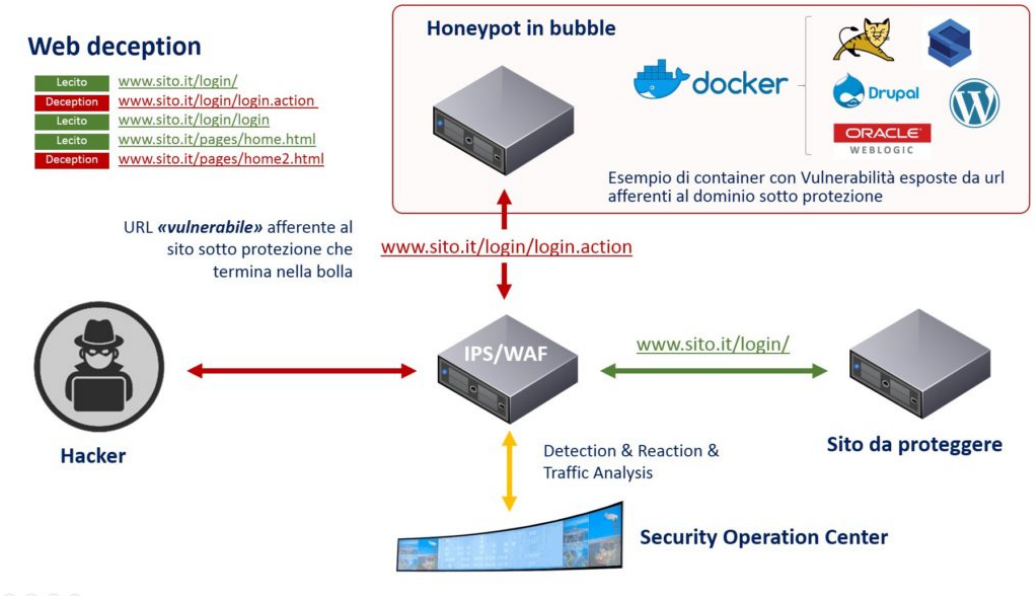

Figura 1: Honeypot per web-application.

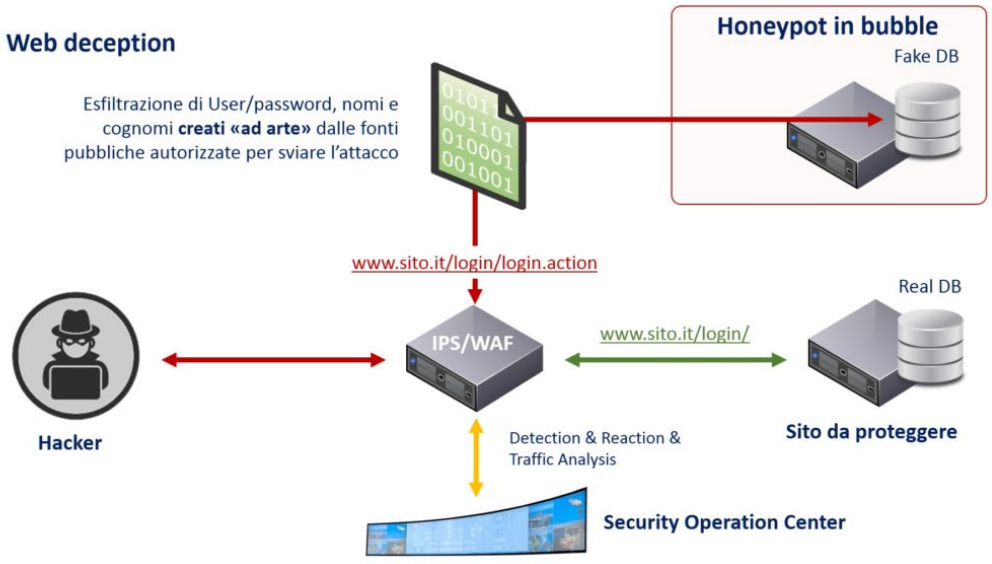

Figura 2: Honeypot multilayer per DB.

Gli honeypots rappresentano una componente essenziale nella difesa proattiva delle reti informatiche contro le minacce sempre più sofisticate dei giorni nostri. Definiti come sistemi o risorse di rete progettati deliberatamente per essere attaccati, gli honeypots fungono da esca per attirare, monitorare e studiare le attività malevole [16], [12]. Secondo (Spitzner L.), essi sono "strumenti di inseguimento dei criminali informatici" [29]. Esistono due tipi principali di honeypots:

- `Low-Interaction Honeypots:` I low-interaction honeypots emulano solo alcune funzionalità di un sistema operativo o di un'applicazione, senza fornire un ambiente operativo completo agli attaccanti. Di solito, questi honeypots rispondono solo a comandi e azioni predefinite, limitando l'interazione con gli aggressori [19].

  Quando un attaccante interagisce con un honeypot a bassa interazione, il sistema registra le attività dell'attaccante, inclusi tentativi di accesso, scansione della rete e tentativi di exploit. Le informazioni raccolte aiutano a identificare le fonti degli attacchi e a comprendere le tattiche utilizzate.

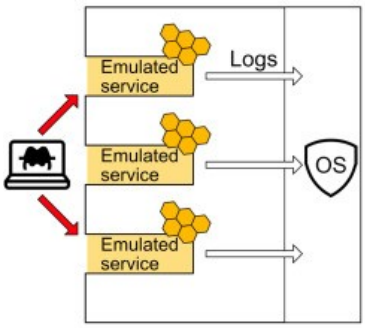

Figura 3: Low-interaction honeypot.

- `High-Interaction Honeypots:` Gli high-interaction honeypots, come descritto da (Provos N.), offrono un ambiente completo e realistico che simula un sistema operativo completo[25]. Questi honeypots permettono agli attaccanti di interagire pienamente con il sistema, raccogliendo dati dettagliati sulle loro azioni, intenti e strumenti utilizzati. Tuttavia, richiedono una gestione più avanzata e possono rappresentare un rischio se non isolati adeguatamente dalla rete reale [1].

  Gli high-interaction honeypots registrano tutte le azioni degli aggressori, inclusi tentativi di accesso, installazione di malware, esecuzione di comandi e movimenti laterali nella rete simulata. Questo fornisce informazioni preziose sulle intenzioni e sulle capacità degli attaccanti.

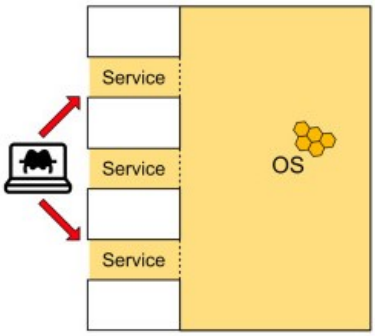

Figura 4: High-interaction honeypot.

Il funzionamento di un honeypot si basa su principi chiave di invisibilità, monitoraggio costante e isolamento sicuro. L'honeypot deve essere configurato in modo da sembrare una parte autentica della rete, al fine di attirare gli attaccanti senza destare sospetti. Durante le interazioni degli attaccanti con l'honeypot, tutte le attività vengono rigorosamente monitorate e registrate per analizzare le tecniche utilizzate e identificare le minacce emergenti [29]. L'implementazione di un honeypot può avvenire in diverse posizioni di rete, ideali sono le DMZ (Demilitarized Zone) o segmenti di rete periferici, in modo da non compromettere la sicurezza delle risorse critiche dell'organizzazione.

È importante che l'honeypot sia configurato e monitorato attentamente per evitare che diventi una risorsa utilizzata contro l'organizzazione stessa. È inoltre necessario garantire un adeguato isolamento degli honeypots dal resto della rete per evitare che un eventuale attacco possa compromettere altri sistemi. Questo isolamento protegge l'infrastruttura principale, mantenendo le attività potenzialmente dannose contenute all'interno del honeypot stesso.

Lo scopo degli honeypots è triplice e strategico. In primis, essi sono fondamentali per il rilevamento tempestivo delle minacce, spesso non identificabili dai sistemi di sicurezza convenzionali come firewall e antivirus. In secondo luogo, fungono da ambiente controllato per la ricerca e l'analisi approfondita delle tecniche di attacco e del comportamento del malware, contribuendo così alla creazione di contromisure più efficaci [25]. Infine, gli honeypots hanno anche una funzione di distrazione, attirando e impegnando gli attaccanti in attività inutili che rallentano i loro progressi verso obiettivi più critici dell'infrastruttura aziendale.

L'evoluzione degli honeypots include nuove tecnologie come le honeynets, che sono reti di honeypots interconnesse per raccogliere dati su una scala più ampia e per studiare le interazioni complesse tra attaccanti e sistemi. Inoltre, l'integrazione di tecniche di machine learning e analisi comportamentale sta migliorando la capacità degli honeypots di rilevare e rispondere automaticamente alle minacce emergenti.

2.1.2 Honeytoken

Gli honeytoken sono uno strumento ingegnoso e sottile nel campo della cybersecurity. Essi rappresentano dati o risorse appositamente creati per attirare e rilevare attività malevole all'interno di un sistema informatico. A differenza degli honeypot, che sono sistemi interi progettati per essere attaccati, gli honeytoken sono piccoli pezzi di informazione disseminati all'interno di un ambiente operativo. Questi possono includere falsi file, credenziali,

chiavi di accesso, o record di database. L'obiettivo principale degli honeytoken è quello di rilevare tentativi di accesso non autorizzato o violazioni della sicurezza [23], [20]. Quando un aggressore tenta di utilizzare un honeytoken per accedere a un sistema o a dati sensibili, l'utente viene allertato dell'attività sospetta. Poiché gli honeytoken non hanno alcuna funzione legittima all'interno del sistema, qualsiasi tentativo di utilizzarli indica un'azione malevola.

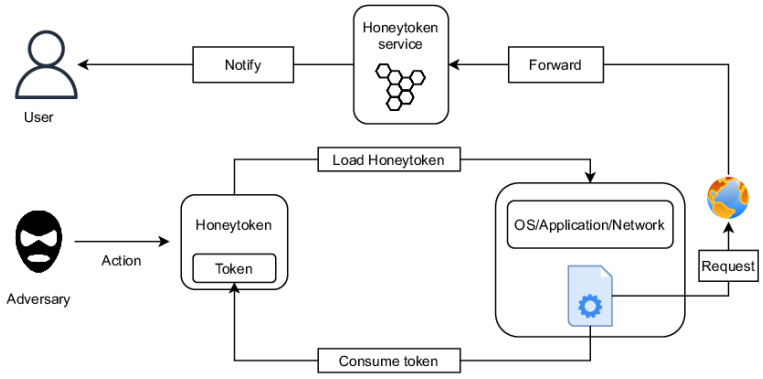

Figura 5: Funzionamento honeytoken.

La potenza degli honeytokens risiede nella loro capacità di attirare e rivelare attività sospette. Quando un honeytoken viene compromesso o utilizzato, genera un alert che avvisa i difensori dell'evento. Questo avviso può essere integrato con sistemi di sicurezza esistenti per avviare risposte immediate, come l'isolamento del segmento di rete interessato o l'avvio di procedure di analisi forense. Secondo (Spitzner L.), esperto di cybersecurity, "gli honeytokens rappresentano un'innovazione cruciale nel campo della cyber defense, permettendo alle organizzazioni di identificare e rispondere rapidamente alle minacce" [29]. Questa capacità di rilevazione precoce non solo aiuta a ridurre l'impatto delle violazioni, ma può anche dissuadere potenziali aggressori dall'approfondire ulteriormente l'attacco. Gli honeytokens servono a due scopi principali: deterrenza e rilevamento precoce. La loro presenza all'interno di una rete o di un sistema aumenta il rischio per gli aggressori, rendendo

meno attraente l'ambiente per l'esplorazione o l'attacco. Allo stesso tempo, gli honeytokens forniscono ai difensori informazioni vitali sulle tattiche degli aggressori e sulle vulnerabilità potenziali del sistema. Secondo uno studio, "l'uso degli honeytokens non solo migliora la capacità di rilevare attività sospette, ma può anche fungere da meccanismo educativo per migliorare la consapevolezza della sicurezza tra i membri dell'organizzazione" [7]. Questo aspetto educativo è essenziale per promuovere una cultura della cybersecurity all'interno delle aziende e delle istituzioni. Ulteriori approfondimenti sull'implementazione degli honeytokens e le loro implicazioni strategiche possono essere trovati nella ricerca condotta da (Miorandi D.) [18]. La ricerca ha approfondito le varie modalità con cui gli honeytokens possono essere distribuiti all'interno di un sistema o di una rete. Essi possono essere inseriti strategicamente in file, database, URL (Localizzatore Uniforme di Risorse) o altri punti di accesso critici, simulando dati sensibili ma senza valore operativo reale. Questa implementazione mirata consente di identificare con precisione quando e come un attaccante tenta di interagire con tali informazioni. Le implicazioni strategiche degli honeytokens sono state esaminate nel contesto della cyber defense, e hanno evidenziato che non solo migliorano la capacità di rilevare e rispondere a intrusioni, ma possono anche influenzare la pianificazione delle risorse di cybersecurity.

2.1.3 Deception Grids

Una deception grid è una rete distribuita di risorse ingannevoli che imitano i veri sistemi, servizi e dati di un'organizzazione. Le risorse all'interno di una deception grid sono progettate per apparire genuine agli occhi degli attaccanti, inducendoli a interagire con queste piuttosto che con le risorse reali. Questa interazione fornisce ai difensori una visibilità preziosa sulle intenzioni e le tecniche degli attaccanti, permettendo loro di raffinare continuamente le proprie strategie di difesa. Le deception grids rappresentano una delle tecniche più avanzate e sofisticate nell'ambito della cybersecurity, progettate per ingannare gli attaccanti e raccogliere informazioni preziose sulle loro tatti-

che, tecniche e procedure (TTP). Queste reti sono costituite da un insieme organizzato di risorse ingannevoli, distribuite in modo strategico attraverso l'infrastruttura di rete di un'organizzazione. L'obiettivo principale è quello di creare un ambiente che attiri e interagisca con gli attaccanti, permettendo ai difensori di studiare le loro mosse senza rischiare la compromissione dei veri sistemi critici.

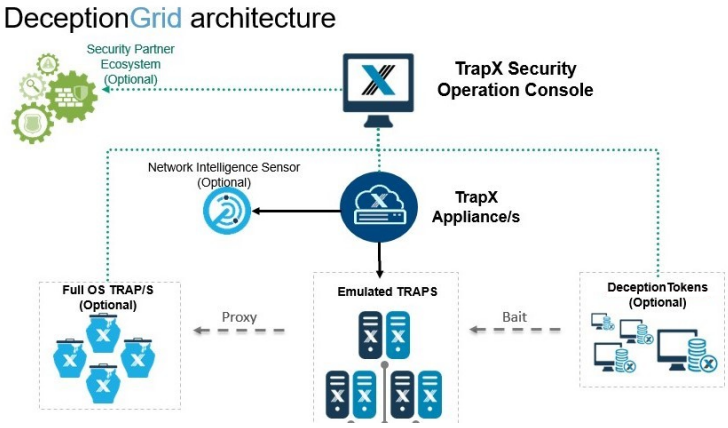

Figura 6: Struttura Deception Grids.

Le deception grids sono distribuite in modo strategico in tutta la rete aziendale. Possono essere posizionate in segmenti di rete ad alto valore, come zone DMZ, segmenti di rete interna, e nei pressi di database sensibili. La distribuzione è pensata per massimizzare le probabilità di interazione con gli attaccanti, creando un ambiente ricco di potenziali trappole. Le risorse ingannevoli all'interno di una deception grid possono includere una varietà di trappole, come honeypots (sistemi falsi che simulano servizi e applicazioni reali), honeytokens (dati falsi progettati per attirare gli attaccanti) e servizi falsi (come server web e database che imitano quelli veri). Questa diversificazione aumenta la probabilità di ingannare gli attaccanti e di raccogliere una gamma più ampia di informazioni sulle loro attività. Ogni interazione con le risorse ingannevoli viene attentamente monitorata e analizzata. Questo monitoraggio consente di rilevare attività sospette in tempo reale e di raccogliere

informazioni dettagliate sugli attaccanti. Le informazioni raccolte possono includere i metodi di attacco utilizzati, gli obiettivi specifici e le vulnerabilità sfruttate. Questi dati sono essenziali per comprendere meglio le minacce e per migliorare continuamente le difese. Le deception grids spesso includono componenti automatizzati che possono rispondere dinamicamente agli attacchi. Ad esempio, possono generare allarmi, modificare le risorse ingannevoli o attivare contromisure in base al comportamento dell'attaccante. Questa automazione aumenta l'efficacia della deception grid e consente ai difensori di reagire rapidamente alle minacce emergenti.

Le deception grids consentono di rilevare attività malevole in una fase molto precoce. Poiché gli attaccanti sono indotti a interagire con queste risorse ingannevoli, è possibile identificarli prima che possano accedere a sistemi critici. Questo rilevamento precoce fornisce un vantaggio significativo nella prevenzione degli attacchi. Le interazioni con le risorse ingannevoli forniscono dati preziosi sulle TTP degli attaccanti. Questa informazione può essere utilizzata per migliorare le difese di sicurezza e per comprendere meglio le minacce emergenti. Ad esempio, analizzando le tecniche di attacco utilizzate, i difensori possono identificare nuove vulnerabilità e sviluppare contromisure specifiche. Poiché le deception grids sono progettate per attirare gli attaccanti, riducono il rumore e i falsi positivi nei sistemi di rilevamento delle intrusioni. Questo miglioramento nell'accuratezza del rilevamento permette ai team di sicurezza di concentrarsi sulle vere minacce, migliorando l'efficienza delle operazioni di sicurezza. Esse inoltre sono altamente flessibili e possono essere adattate rapidamente per rispondere a nuove minacce. La loro natura distribuita rende difficile per gli attaccanti identificare ed evitare tutte le risorse ingannevoli. Inoltre, la possibilità di aggiornare e modificare continuamente le trappole consente di mantenere un ambiente sempre imprevedibile.

2.1.4 Fake Vulnerabilities

Le Fake Vulnerabilities sono difetti di sicurezza simulati che vengono deliberatamente introdotti nei sistemi informatici. A differenza delle vere vulnerabilità, che possono rappresentare un rischio reale per la sicurezza, le fake vulnerability sono progettate per apparire autentiche agli occhi degli attaccanti. Il loro scopo principale è quello di agire come trappole, attirando gli aggressori e rivelando i loro metodi di attacco. Questo approccio permette ai difensori di comprendere meglio le minacce e di migliorare le proprie strategie di sicurezza.

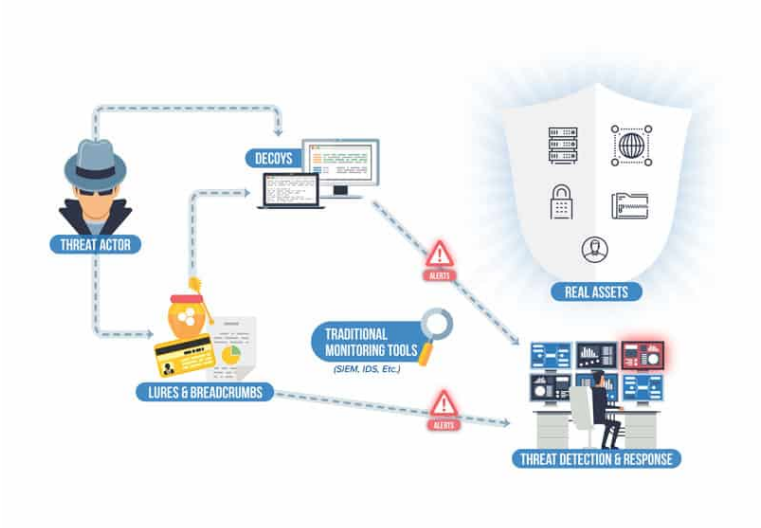

Figura 7: Fake Vulnerabilities.

Il funzionamento delle Fake Vulnerabilities può essere suddiviso in diverse fasi chiave:

- `Creazione delle Fake Vulnerability:` Le fake vulnerability sono integrate nei sistemi attraverso una progettazione attenta e deliberata. Queste possono includere configurazioni errate, credenziali deboli, porte aperte non necessarie, o errori di codice apparentemente inno-

cui. L'obiettivo è far sembrare queste vulnerabilità reali e sfruttabili, attirando l'attenzione degli attaccanti.

- `Distribuzione nei Sistemi:` Una volta create, le fake vulnerability vengono distribuite nei sistemi target. Questo può essere fatto in vari modi, come l'inserimento di codice vulnerabile in applicazioni web, la configurazione di servizi di rete con impostazioni deboli o l'implementazione di credenziali facilmente crackabili.

- `Monitoraggio e Raccolta di Dati:` Le fake vulnerability sono costantemente monitorate per rilevare qualsiasi tentativo di sfruttamento. I sistemi di monitoraggio possono includere logging dettagliato, rilevamento di intrusioni e altre forme di sorveglianza che registrano le attività degli attaccanti. Questo permette ai difensori di raccogliere dati su chi sta tentando di sfruttare la vulnerabilità, come lo stanno facendo e da dove proviene l'attacco.

- `Analisi e Risposta:` Le informazioni raccolte dai tentativi di sfruttamento delle fake vulnerability vengono analizzate per comprendere meglio le tecniche utilizzate dagli attaccanti. Questa analisi può rivelare nuove metodologie di attacco, strumenti utilizzati dagli aggressori e punti deboli nelle difese attuali. In base a questi dati, i difensori possono adattare le loro strategie di sicurezza, implementare nuove misure di protezione e rafforzare le difese esistenti.

Le Fake Vulnerabilities possono essere implementate in vari modi, tra cui ad esempio: configurazioni di sistema, codice vulnerabile e servizi falsi.

2.1.5 Sandboxing

Il sandboxing è una tecnica importante nel contesto della cybersecurity. Questo approccio consente di eseguire programmi all'interno di un ambiente isolato, o "sandbox", con l'obiettivo di limitare l'accesso del software a risorse critiche del sistema operativo, riducendo così i potenziali danni. In altre

parole, il sandboxing permette di testare applicazioni sospette o non verificate senza mettere a rischio l'integrità dell'intero sistema.

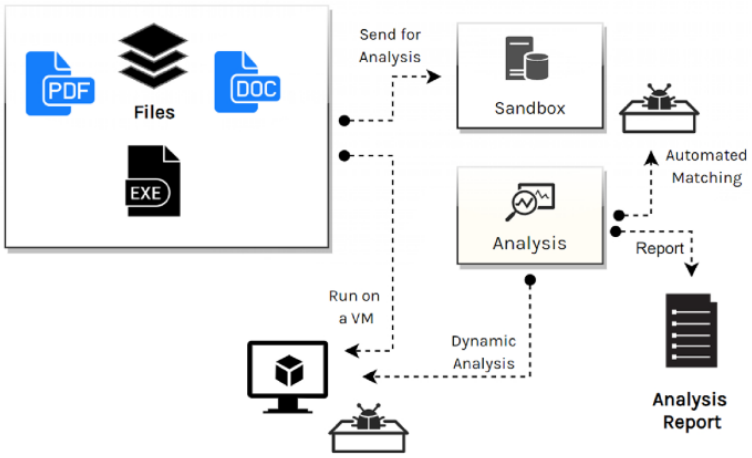

Figura 8: Struttura e funzionamento sandboxing.

Esistono diverse tipologie di sandboxing, ciascuna con caratteristiche specifiche in base al livello di isolamento e al contesto di applicazione.

Una delle principali categorie è il sandboxing a livello di sistema operativo. In questo caso, è il sistema operativo stesso a fornire l'isolamento tra le applicazioni. Un esempio di questa tipologia è rappresentato da Windows Sandbox, che crea un ambiente virtuale isolato in cui è possibile eseguire applicazioni senza che queste possano accedere a risorse sensibili del sistema operativo. Analogamente, macOS implementa una tecnologia simile denominata App Sandbox. È stato condotto uno studio al fine di evidenziare come queste tecnologie possano proteggere il sistema operativo principale da comportamenti potenzialmente dannosi delle applicazioni eseguite all'interno della sandbox [10].

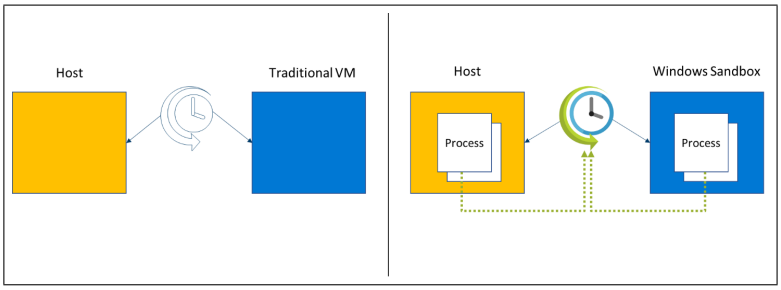

Figura 9: Struttura Windows Sandbox.

Un'altra importante tipologia è il sandboxing a livello di browser, che viene utilizzato principalmente per isolare i contenuti web e prevenire attacchi come il cross-site scripting (XSS). Browser moderni, come Google Chrome, adottano tecniche di sandboxing per isolare ogni scheda in un processo separato. Questo approccio aumenta significativamente la sicurezza, poiché un eventuale attacco a una scheda non può compromettere l'intero browser. Uno studio nello specifico ha dimostrato come l'architettura di Chromium riesca a migliorare significativamente la sicurezza del browser attraverso l'isolamento dei processi e l'applicazione di politiche di sicurezza rigorose, e come queste misure contribuiscono a mitigare i rischi associati all'esecuzione di codice potenzialmente malevolo da parte delle pagine web [3].

Infine, vi è il sandboxing a livello di applicazione, che coinvolge strumenti come le macchine virtuali (ad esempio, VirtualBox o VMware) e i container (ad esempio, Docker). Questi strumenti forniscono un ambiente sandbox isolato per interi sistemi operativi guest o applicazioni specifiche, garantendo che le attività all'interno della sandbox non possano interferire con il sistema operativo. (Merkel, D.) ha discusso ampiamente i benefici dell'uso dei container per l'isolamento delle applicazioni, sottolineando come questa tecnologia permetta una gestione più efficiente delle risorse e una maggiore sicurezza[17].

Il funzionamento del sandboxing si basa su alcuni principi fondamentali, tra cui l'isolamento, il controllo delle risorse e la limitazione dei privilegi. Ogni

sandbox crea un ambiente virtuale isolato che impedisce alle applicazioni eseguite al suo interno di accedere direttamente alle risorse del sistema operativo "Host". Questo isolamento è cruciale per prevenire la propagazione di comportamenti dannosi e garantire che un eventuale attacco non possa compromettere l'intero sistema [24].

2.2 La containerizzazione

La containerizzazione rappresenta una delle tecnologie più innovative e sicure per il deployment e la gestione delle applicazioni moderne. Uno dei principali vantaggi che essa offre riguarda la sicurezza, un aspetto cruciale nell'odierno panorama tecnologico sempre più soggetto a minacce e attacchi informatici.

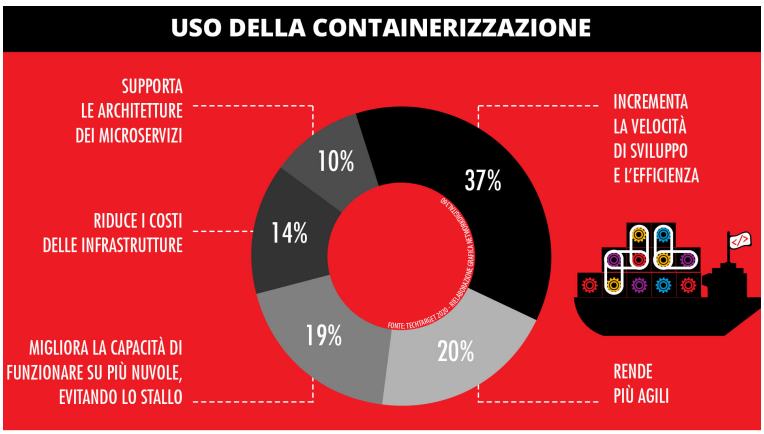

Figura 10: Vantaggi containerizzazione.

Un aspetto fondamentale della containerizzazione è l'isolamento delle applicazioni e delle loro dipendenze dal sistema operativo host e dalle altre applicazioni. Questo isolamento garantisce che un potenziale attacco su un container rimanga confinato al suo interno, riducendo significativamente il rischio che tale attacco si propaghi ad altre parti del sistema. Grazie a questa separazione, è possibile limitare l'impatto di eventuali vulnerabilità, proteg-

gendo l'integrità complessiva dell'infrastruttura [22]. Ogni container contiene solo i componenti strettamente necessari per eseguire una specifica applicazione. Questa riduzione dei componenti in esecuzione comporta una diminuzione della superficie di attacco potenziale, poiché meno software significa meno possibilità che esistano vulnerabilità sfruttabili da attaccanti. Questo approccio minimalista contribuisce a rendere l'intero sistema più sicuro. È possibile configurare i container per utilizzare solo una quantità specifica di risorse, impedendo che un container compromesso possa consumare tutte le risorse del sistema host e causare un Denial of Service (DoS). L'uso di namespace e cgroups fornisce ulteriore separazione e limitazione delle risorse tra i container. I namespace isolano i componenti del sistema operativo come rete, processi e mount points, mentre i cgroups controllano e limitano l'uso delle risorse [30]. Un aspetto vantaggioso riguarda la facilità con cui è possibile aggiornare e patchare le applicazioni. Ogni container è indipendente, quindi le applicazioni possono essere aggiornate senza influenzare altre applicazioni o il sistema operativo host. Un ulteriore vantaggio, consiste nella facilita l'integrazione con strumenti di monitoraggio e logging che centralizzano la raccolta di dati di performance e di sicurezza. Questo facilita il rilevamento di attività sospette e l'analisi delle anomalie, permettendo una risposta più rapida agli incidenti di sicurezza. I registri dei container possono essere configurati per essere immutabili, garantendo che le informazioni di logging non possano essere alterate da attori malintenzionati. Questo è fondamentale per mantenere l'integrità dei dati di audit e investigazione.

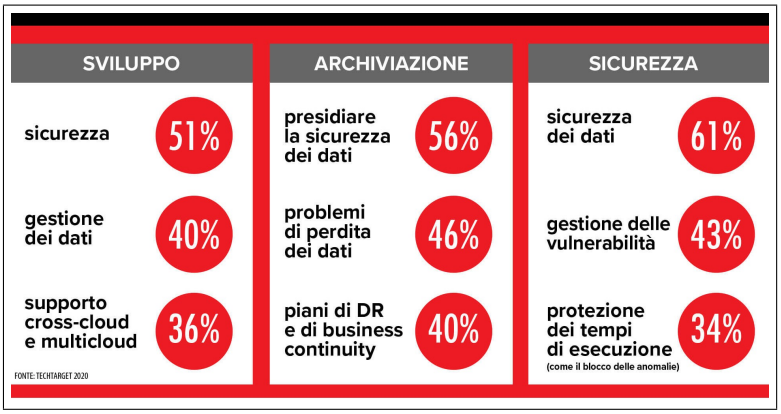

Figura 11: Miglioramento della sicurezza con i container.

Docker è una piattaforma di containerizzazione che ha rivoluzionato il modo in cui sviluppatori e amministratori di sistema gestiscono le applicazioni. L'adozione di Docker ha facilitato lo sviluppo, il deployment e la scalabilità delle applicazioni, introducendo un modello efficiente e flessibile per l'esecuzione di software.

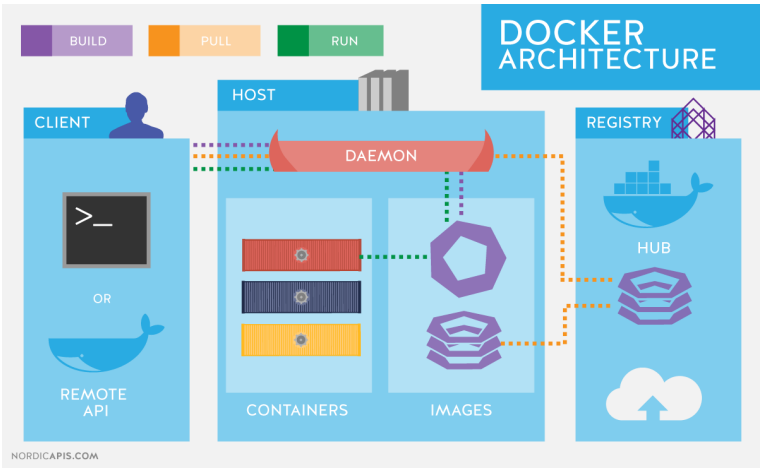

Figura 12: Struttura Docker.

Il funzionamento di Docker si basa sul concetto di container, che può essere visto come una leggera alternativa alle macchine virtuali (VM). A dif-

ferenza delle VM, che richiedono un hypervisor per emulare un'intera macchina fisica, i container condividono lo stesso kernel del sistema operativo host ma isolano le applicazioni e le loro dipendenze. Questo approccio rende i container molto più leggeri e veloci da avviare rispetto alle VM.

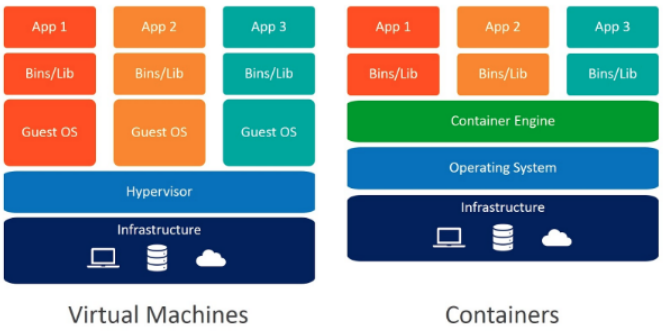

Figura 13: Struttura Virtual Machine vs Docker Containers.

Il cuore di Docker è il Docker Engine, che è responsabile della gestione delle immagini e dei container. Il Docker Engine può essere suddiviso ulteriormente in tre componenti principali: il daemon, che esegue e gestisce i container; il client, che permette agli utenti di interagire con il daemon attraverso la linea di comando, e l'API RESTful, che consente l'interazione con Docker tramite applicazioni.

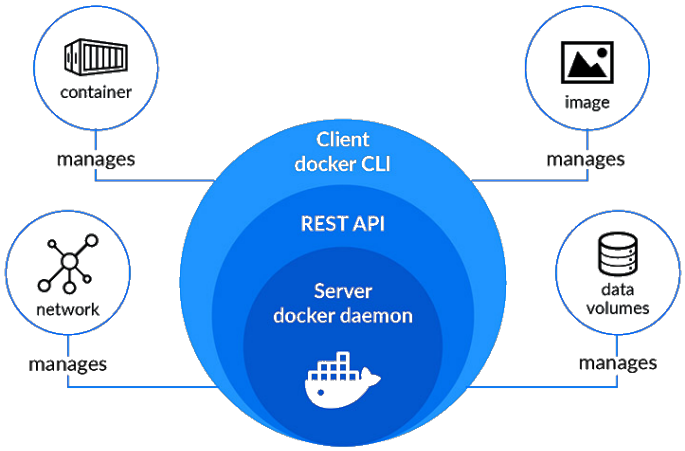

Figura 14: Struttura interna Docker.

Le immagini Docker sono una rappresentazione statica di un'applicazione e delle sue dipendenze. Le immagini sono create utilizzando un Dockerfile, un file di testo che contiene una serie di istruzioni per costruire l'immagine. Una volta costruite, le immagini possono essere memorizzate in un registro, come Docker Hub, e possono essere distribuite e utilizzate per avviare container su qualsiasi sistema compatibile con Docker. I container sono istanze runtime delle immagini. Ogni container è isolato dagli altri e dal sistema operativo host, garantendo che l'applicazione all'interno del container funzioni in modo coerente, indipendentemente dall'ambiente in cui viene eseguita.

La sicurezza è un aspetto importante della containerizzazione con Docker. Docker implementa diverse funzionalità di sicurezza per garantire che i container siano sicuri e isolati. L'isolamento dei container è ottenuto utilizzando namespace e cgroups (control groups) del kernel di Linux. I namespace forniscono isolamento per le risorse di sistema, come processi, rete e mount points, mentre i cgroups gestiscono e limitano l'uso delle risorse di sistema come CPU, memoria e I/O [4].

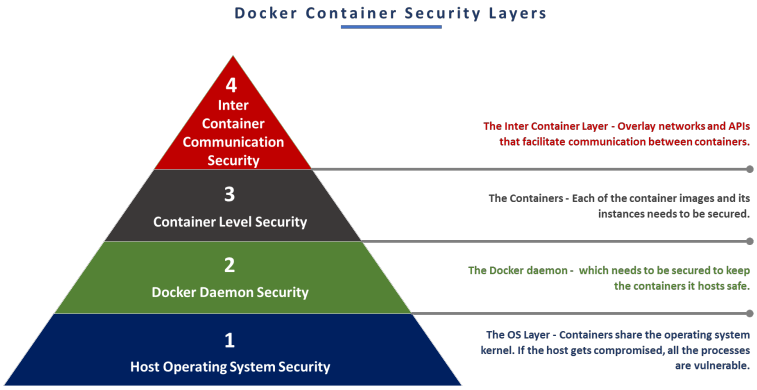

Figura 15: Strati sicurezza container Docker.

La containerizzazione con Docker ha trasformato il modo in cui le applicazioni sono sviluppate, distribuite e gestite. Questo approccio presenta diversi vantaggi rispetto ai metodi tradizionali. I container Docker possono essere eseguiti su qualsiasi sistema compatibile con Docker, garantendo che le applicazioni funzionino in modo coerente in ambienti diversi, come sviluppo, testing e produzione. Questo elimina i problemi di compatibilità e facilita la distribuzione delle applicazioni [5]. I container sono leggeri e si avviano rapidamente, poiché non richiedono un intero sistema operativo separato. Questo consente di eseguire un numero maggiore di container su un singolo host rispetto alle VM, ottimizzando l'uso delle risorse di sistema. Docker facilita la scalabilità delle applicazioni. È possibile avviare rapidamente nuovi container per gestire picchi di traffico o distribuire il carico su più container per migliorare le performance e la resilienza dell'applicazione [13].

Ogni container contiene tutte le dipendenze necessarie per eseguire l'applicazione, garantendo che le applicazioni siano isolate l'una dall'altra e dalle librerie di sistema. Questo semplifica la gestione delle dipendenze e riduce i conflitti tra diverse versioni di librerie o software. Docker è un componente chiave delle pratiche DevOps e CI/CD (Integrazione e Distribuzione Continue). La possibilità di creare, testare e distribuire container in modo automatizzato facilita l'integrazione continua e il deployment continuo, migliorando l'efficienza e la velocità del ciclo di vita dello sviluppo software.

Capitolo 3

Architettura del Sistema

L'architettura del sistema sviluppato per integrare dati tra file XML e un database MongoDB utilizzando Docker è un modello sofisticato e ben orchestrato. Questo sistema sfrutta le potenzialità della containerizzazione per garantire portabilità, isolamento e gestione efficiente delle dipendenze software. L'intero sistema è organizzato utilizzando Docker Compose, uno strumento che semplifica la configurazione e l'esecuzione di applicazioni multi-container [14]. L'architettura comprende quattro componenti principali: Draw.io, Python, MongoDB e Docker Compose, ognuno dei quali svolge un ruolo fondamentale nel processo complessivo.

3.1 Integrazione di XML e MongoDB tramite Docker

L'integrazione di dati tra file XML e un database MongoDB tramite Docker rappresenta una strategia moderna e flessibile per la gestione di dati eterogenei. Questo approccio sfrutta i container Docker per garantire portabilità e isolamento degli ambienti di esecuzione, semplificando la configurazione e la gestione delle dipendenze software. La containerizzazione con Docker consente di creare ambienti di esecuzione replicabili e coerenti, minimizzando

problemi di compatibilità tra diverse configurazioni di sistema e facilitando il deployment in ambienti di sviluppo e produzione.

3.2 Tecnologie e Linguaggi di Programmazione Utilizzati

3.2.1 Draw.io

Draw.io è uno strumento di progettazione diagrammi ampiamente utilizzato per creare rappresentazioni visive delle strutture dati. La sua interfaccia intuitiva permette di modellare facilmente diagrammi di flusso, diagrammi di entità-relazione (ER), diagrammi UML e altre rappresentazioni visive complesse.

L'utilizzo di draw.io è molto conveniente per la progettazione del diagramma, in quanto consente di generare ed estrarre file XML in modo semplice ed efficiente. Questo strumento offre un'interfaccia intuitiva e user-friendly che facilita la creazione di diagrammi ER per modellare le strutture del database. Una volta completato il diagramma, draw.io permette di esportarlo come file XML, fornendo una rappresentazione completa e dettagliata del diagramma stesso. Questo file XML può essere facilmente trasformato in dati JSON-like, pronti per l'inserimento in MongoDB, semplificando notevolmente il processo di generazione del sistema. La possibilità di estrarre facilmente l'XML rende draw.io una scelta ideale per progetti che richiedono l'integrazione e la manipolazione dei dati attraverso un flusso di lavoro automatizzato e ben strutturato.

3.2.2 Phyton

Phyton è il linguaggio di programmazione scelto per eseguire il processo di estrazione, trasformazione e caricamento (ETL) dei dati XML in MongoDB. Le operazioni principali eseguite dallo script python implementato sono:

- `Parsing del File XML:` Utilizzando il modulo xml.etree.ElementTree [26], lo script Python analizza il file XML generato da draw.io, estraendo le informazioni sulle entità e relazioni definite nel diagramma.

- `Pulizia dei Dati:` I dati estratti vengono puliti rimuovendo eventuali tag HTML con l'ausilio di librerie come BeautifulSoup, garantendo che i dati siano in un formato puro e utilizzabile.

- `Inserimento in MongoDB:` Una volta puliti e trasformati in formato JSON-like, i dati vengono inseriti in MongoDB utilizzando il driver pymongo, creando documenti che riflettono accuratamente le strutture e le relazioni definite nel diagramma XML.

3.2.3 MongoDB

MongoDB è un database NoSQL scelto per la sua capacità di gestire dati non strutturati e semistrutturati. Le caratteristiche principali che hanno portato alla scelta dell'utilizzo di MongDB sono:

- `Flessibilità del Modello di Dati:` Utilizza un modello di dati basato su documenti, permettendo di gestire strutture di dati dinamiche e flessibili.

- `Scalabilità Orizzontale:` Progettato per scalare orizzontalmente, distribuendo i dati su più server e supportando il bilanciamento del carico.

- `Alta Disponibilità e Tolleranza ai Guasti:` Grazie alla replica dei dati e alla gestione delle partizioni, MongoDB garantisce alta disponibilità e tolleranza ai guasti.

3.2.4 Docker Compose

Docker Compose è uno strumento che consente di definire e gestire applicazioni multi-container in un unico file YAML. Docker compose permette:

- `Definizione dei Servizi:` Nel file docker-compose.yml si definiscono i servizi necessari per l'applicazione, includendo i container per MongoDB e per lo script Python, descrivendo le relative immagini Docker, porte esposte e dipendenze.

- `Configurazione dei Container:` Docker Compose configura automaticamente i container, impostando variabili d'ambiente, volumi e reti per garantire che i servizi possano comunicare tra loro in modo sicuro ed efficiente.

- `Avvio e Gestione dei Servizi:` Con un singolo comando, Docker Compose avvia tutti i container definiti, assicurando che i servizi siano operativi e pronti per l'esecuzione del processo ETL. Gestisce anche le risorse e monitora lo stato dei container, facilitando la manutenzione e l'aggiornamento del sistema.

3.2.5 Dockerfile

Il Dockerfile è un file di configurazione che definisce l'ambiente di esecuzione per il container Python, specificando la base image, le dipendenze e i file necessari.

- `Base Image Python 3.9:` Definisce una base image standard di Python 3.9, garantendo compatibilità con lo script Python e le librerie utilizzate.

- `Installazione delle Dipendenze:` Utilizzando il comando RUN, il Dockerfile installa tutte le dipendenze specificate in un file requirements.txt, come pymongo e BeautifulSoup.

- `Copia dei File Necessari:` Il Dockerfile copia i file XML e Python nel container, preparando l'ambiente per l'esecuzione dello script ETL.

Questa architettura ben definita e dettagliata permette di gestire in modo efficiente e scalabile il flusso di dati dai file XML a MongoDB, sfruttando

la potenza e la flessibilità delle tecnologie Docker, MongoDB e Python. La containerizzazione con Docker Compose facilita la gestione delle dipendenze e delle configurazioni, mentre MongoDB fornisce un sistema di archiviazione robusto e scalabile. Python, con le sue potenti librerie, assicura un processo ETL fluido e affidabile. Questo approccio modulare e ben orchestrato garantisce un'elevata portabilità, facilità di manutenzione e scalabilità del sistema complessivo.

Capitolo 4

Implementazione del sistema

4.1 Progettazione del diagramma con Draw.io

Draw.io risulta utile nel nostro progetto per motivi pratici. La sua presenza aiuta la progettazione e la visualizzazione chiara delle strutture dati complesse, consentendo di modellare le relazioni tra entità e di rappresentare in modo intuitivo i flussi di dati e le interazioni all'interno del sistema. La scelta del suo utilizzo, è motivata dalla sua interfaccia user-friendly, che rende accessibile anche agli utenti meno esperti la creazione di diagrammi dettagliati. Questa facilità d'uso è efficace per garantire che tutto il team di sviluppo, compresi gli stakeholder non tecnici, possa partecipare attivamente alla progettazione e comprendere il layout del database.

Un altro vantaggio di draw.io risiede nella sua flessibilità e versatilità. Il software supporta una vasta gamma di modelli predefiniti, tra cui diagrammi ER, diagrammi di flusso, diagrammi UML e molti altri. Questa varietà consente di adattare il tipo di diagramma alla specifica necessità del progetto, assicurando che ogni aspetto delle relazioni e delle entità nel database sia rappresentato in modo accurato e completo.

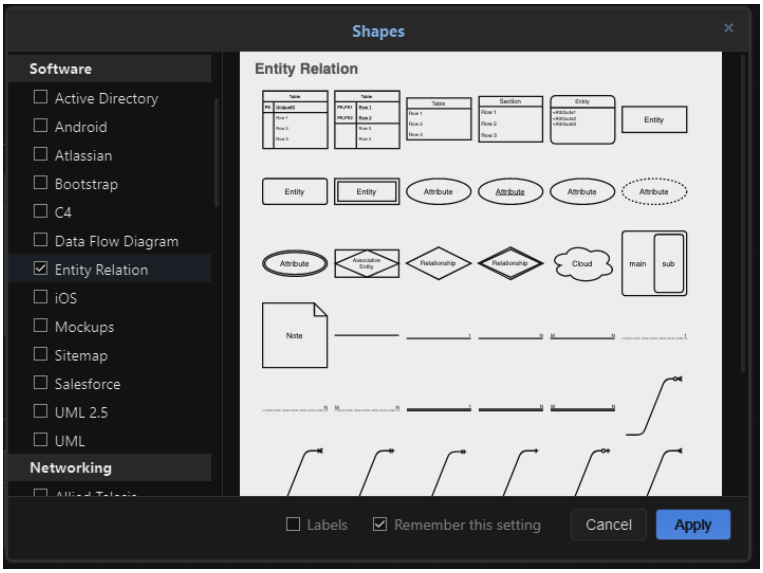

Figura 16: Tipi di diagrammi Draw.io.

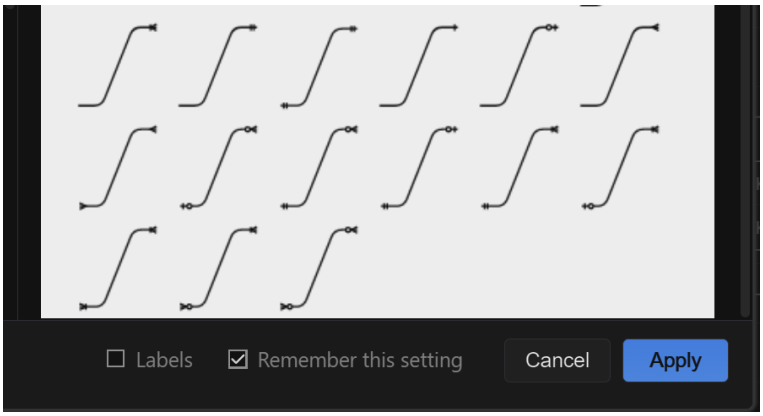

Figura 17: Connettori Entita-Relazioni.

Oltre alla creazione di entità e relazioni, il software consente di aggiungere dettagli con etichette di testo e formattazioni HTML. Tale dettaglio migliora la chiarezza visiva del diagramma, e fornisce anche un riferimento dettagliato durante l'implementazione effettiva del database.

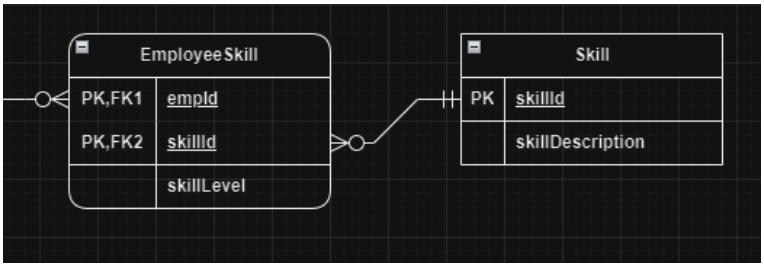

Figura 18: Personalizzazione degli elementi.

Il software inoltre, consente di minimizzare errori e ambiguità nel design del database, fornendo un fondamento solido e ben strutturato per il successivo sviluppo e gestione del sistema.

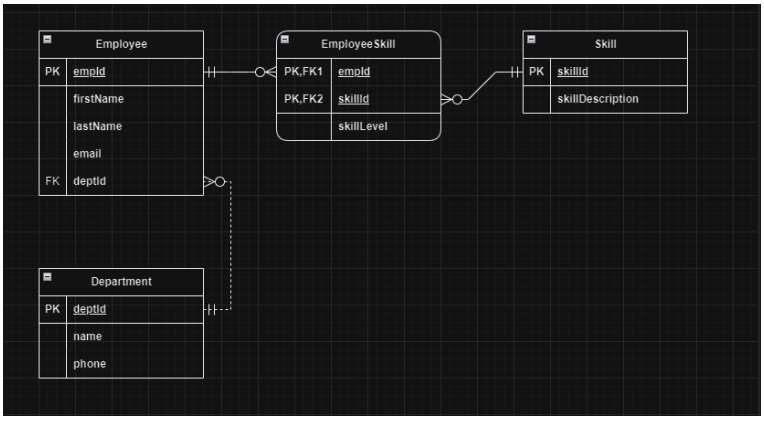

Figura 19: Diagramma finale.

Il suo utilizzo, non si limita alla fase di progettazione iniziale, ma continua a essere prezioso durante l'intero ciclo di vita del progetto. Mediante l'utilizzo di questo software, risulta molto semplice aggiornare e modificare i diagrammi. Questo permette di rispondere prontamente ai cambiamenti nei requisiti del progetto o alle nuove necessità emerse durante lo sviluppo. Tutto ciò rende draw.io un utile strumento di progettazione, ed anche un alleato strategico per la gestione dinamica ed efficace dei progetti IT.

4.2 Esportazione del diagramma in formato XML

Una volta completata la progettazione del diagramma su draw.io, è possibile convertirlo in diversi formati, tra cui XML. Questo formato è particolarmente vantaggioso perché consente di trasformare il diagramma visivo in una struttura dati facilmente interpretabile dalle macchine, utile per i processi di integrazione dei dati in ambienti complessi.

Durante l'esportazione in formato XML, Draw.io genera un file che utilizza tag specifici per rappresentare ogni elemento del diagramma in modo dettagliato. Ad esempio, un'entità come "Utenti" è descritta con un tag "entity" che include attributi come id, nome e email, mentre le connessioni tra entità sono definite con tag "relationship" che specificano i dettagli delle relazioni.

Al termine del processo di generazione, è possibile modellare e visualizzare le strutture dati in modo dettagliato. I dati sono pronti per essere elaborati e integrati in un database NoSQL come ad esempio, nel nostro caso, MongoDB. Questo approccio garantisce che le informazioni siano accuratamente trasferite e mantenute coerenti attraverso l'intero processo.

4.3 Processo Operativo

Il processo per l'integrazione dei dati dal file XML a MongoDB, orchestrato tramite Docker, segue una sequenza strutturata e ben definita di fasi. Questo approccio garantisce che i dati siano correttamente trasformati e inseriti nel database, preservando l'integrità e l'usabilità delle informazioni. Di seguito sono descritte in dettaglio le diverse fasi, dalla configurazione iniziale dei servizi fino alla chiusura e pulizia dell'ambiente.

4.3.1 Sviluppo e funzionamento del codice python

L'implementazione del codice Python per l'integrazione dei dati XML in MongoDB rappresenta una fase cardine del flusso di lavoro. Il codice è progettato per eseguire un processo ETL (Extract, Transform, Load) robusto ed efficiente, garantendo che i dati siano estratti, trasformati e caricati correttamente nel database.

Il processo inizia con un'attesa di alcuni istanti per garantire che il server MongoDB sia completamente operativo prima di stabilire una connessione. Questo passaggio è essenziale per evitare errori di connessione che potrebbero verificarsi se MongoDB non fosse ancora pronto. Il codice prosegue con il parsing del file XML utilizzando il modulo xml.etree.ElementTree. Viene analizzato il file XML situato nel percorso specificato, estraendo la struttura del documento e preparandolo per l'elaborazione successiva.

```
# Parse XML file
tree = ET.parse('/app/Db_example.xml')
root = tree.getroot()
```

Figura 20: Parsing file XML.

Successivamente, il codice stabilisce una connessione a MongoDB utilizzando il driver pymongo. Viene specificato l'host, la porta e il nome del database. In questo caso, il database si chiama 'mydatabase' e la collezione in cui verranno inseriti i dati è denominata "database".

```
# MongoDB connection
client = MongoClient('mongo', 27017)
db = client['mydatabase']

# Set the collection name to "database"
collection_name = "database"
collection = db[collection_name]
```

Figura 21: Connessione MongoDB.

Il codice procede iterando su tutti i diagrammi presenti nel file XML, estraendo le informazioni rilevanti. Per ogni mxCell nel diagramma, viene estratto il valore dell'attributo 'value'. Utilizzando la libreria BeautifulSoup, i tag HTML vengono rimossi dal testo per ottenere dati puliti.

```
# Create collections and insert sample data
for diagram in root.findall('.//diagram'):
    # Populate collection with sample data
    sample_data = {}
    for cell in diagram.findall('.//mxCell'):
        if 'value' in cell.attrib:
            value = cell.get('value')
            # Remove HTML tags
            text = BeautifulSoup(value, 'html.parser').get_text().strip()
```

Figura 22: Estrazione e pulizia informazioni.

Dopo la trasformazione, i dati vengono caricati nella collezione specificata in MongoDB. Il codice verifica i dati da inserire prima di procedere con l'inserimento. Questa precauzione evita di creare documenti vuoti nel database. Infine, una volta completato il processo di inserimento, la connessione a MongoDB viene chiusa in modo sicuro per liberare le risorse.

```
    # Insert sample data into MongoDB collection
    if sample_data:  # Check if sample_data is not empty before inserting
        collection.insert_one(sample_data)

# Close MongoDB connection
client.close()
```

Figura 23: Caricamento dati e chiusura conessione.

4.3.2 Avvio dei servizi

La prima fase del flusso di lavoro prevede l'avvio del servizio MongoDB. Questo avviene tramite Docker Compose. Docker Compose è uno strumento essenziale per definire e gestire ambienti multi-container. Nel nostro caso, utilizziamo Docker Compose per orchestrare due container principali: uno per MongoDB e uno per l'applicazione Python. Il file docker-compose.yml descrive la configurazione necessaria per eseguire questi container in modo coordinato. MongoDB è configurato per esporre la porta "27017", consentendo al container Python di connettersi ad esso. La dipendenza tra i servizi è specificata con la direttiva "depends-on", che garantisce che il servizio Python venga eseguito solo dopo che MongoDB è attivo e funzionante.

docker-compose.yml

```yaml
version: '3.9'
services:
  mongo:
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - mongo-data:/data/db

  python-app:
    build: .
    depends_on:
      - mongo
    volumes:
      - .:/app
    command: python /app/db_conv.py

volumes:
  mongo-data:
```

Figura 24: File Docker Compose.

4.3.3 Creazione dell'immagine docker per l'applicazione python

Per l'applicazione Python, utilizziamo un Dockerfile per definire l'ambiente di esecuzione. Partiamo da un'immagine base di Python, quindi impostiamo la directory di lavoro, copiamo i file necessari e installiamo le dipendenze elencate nel file requirements.txt. Infine, specifichiamo il comando per eseguire lo script Python.

```dockerfile
# Usa un'immagine base di Python
FROM python:3.9-slim

# Imposta la directory di lavoro
WORKDIR /app

# Copia i file necessari
COPY requirements.txt requirements.txt
COPY db_conv.py db_conv.py
COPY Db_example.xml Db_example.xml

# Installa le dipendenze Python
RUN pip install --no-cache-dir -r requirements.txt

# Esegui lo script Python
CMD ["python", "db_conv.py"]

```

Figura 25: Creazione dell'immagine phyton.

4.4 Vantaggi dell'integrazione delle tecnologie utilizzate nell'architettura applicativa

4.4.1 Interoperabilità tra Python, MongoDB e Docker

Integrazione Fluide tra Tecnologie: L'interoperabilità tra Python, MongoDB e Docker consente un'integrazione fluida e senza soluzione di continuità tra i diversi componenti del sistema. Python fornisce un'interfaccia flessibile per l'elaborazione dei dati XML e l'integrazione con MongoDB, mentre Docker offre un ambiente di esecuzione coeso e isolato per l'esecuzione dei servizi.

Trasformazione dei Dati Senza Soluzione di Continuità: L'uso di Python per l'elaborazione dei dati XML consente una trasformazione dei dati senza soluzione di continuità, passando direttamente dal parsing dei file XML alla loro inserzione in MongoDB. Questo processo continuo e integrato

riduce il rischio di errori e semplifica il flusso di lavoro complessivo.

4.4.2 Efficienza e manutenibilità del sistema

Automazione del Flusso di Lavoro: Docker Compose permette l'automazione del flusso di lavoro, semplificando l'avvio e la gestione dei servizi necessari per l'elaborazione dei dati XML e il loro inserimento in MongoDB. Questo riduce la necessità di intervento manuale e aumenta l'efficienza complessiva del sistema.

Facilità di Deploy e Manutenzione: L'uso di Docker semplifica il deployment e la gestione del sistema, consentendo di distribuire facilmente l'applicazione su diversi ambienti senza dover gestire manualmente le dipendenze e la configurazione dei componenti. Questo riduce il tempo e gli sforzi necessari per la distribuzione e la manutenzione del sistema.

4.4.3 Supporto per l'evoluzione del progetto

Flessibilità nell'Adattamento alle Esigenze Cambianti: L'architettura modulare del sistema basato su Docker, MongoDB e Python consente un facile adattamento alle esigenze cambianti del progetto. Modifiche o aggiornamenti possono essere facilmente implementati modificando i singoli componenti senza dover riprogettare l'intero sistema, garantendo una rapida iterazione e un'evoluzione continua.

Scalabilità Orizzontale e Verticale: L'uso di Docker e MongoDB permette di scalare il sistema sia orizzontalmente, distribuendo i servizi su più nodi o macchine, sia verticalmente, aumentando le risorse disponibili per ciascun componente del sistema. Questa flessibilità consente al sistema di adattarsi dinamicamente ai cambiamenti nei requisiti di prestazioni e di carico di lavoro.

L'integrazione sinergica delle tecnologie porta a un miglioramento globale dell'efficienza del sistema, riducendo i tempi di sviluppo, deployment e manutenzione e aumentando la scalabilità e la flessibilità complessive. Ciò si traduce in una maggiore produttività e in una migliore esperienza utente finale.

Capitolo 5

Sperimentazione

Questo capitolo è dedicato alla presentazione e all'analisi dei risultati ottenuti dall'esecuzione dei programmi sviluppati nel corso di questo lavoro di tesi. Verranno presentati i dati e le osservazioni raccolti durante l'esecuzione dei container, analizzando in dettaglio le prestazioni del sistema, l'integrazione tra i vari componenti e l'efficacia delle operazioni di parsing e inserimento dei dati in MongoDB. Attraverso l'analisi dei risultati, sarà possibile valutare l'efficienza e la robustezza dell'architettura proposta, nonché identificare eventuali aree di miglioramento per ottimizzare ulteriormente il sistema.

5.1 Costruzione ed esecuzione dei container Docker

Il cuore del processo è l'utilizzo di Docker per creare un ambiente isolato in cui eseguire un'applicazione Python dedicata all'elaborazione di dati XML e al loro inserimento in MongoDB. Docker permette di definire l'intero stack applicativo, inclusi i suoi requisiti e dipendenze, all'interno di container autonomi e portabili. Questo approccio elimina la necessità di configurare manualmente l'ambiente di sviluppo su diversi computer o server, garantendo che l'applicazione funzioni in modo identico ovunque venga eseguita. Nel Dockerfile, vengono specificate tutte le istruzioni necessarie per creare

un'immagine Docker personalizzata. Partendo da un'immagine di base come "python:3.9-slim", vengono installate le dipendenze Python richieste per eseguire lo script di conversione XML in MongoDB. Il Dockerfile assicura che l'ambiente di esecuzione sia completo e coerente, rendendo più semplice il deployment dell'applicazione. Docker Compose entra in gioco per gestire i vari servizi che compongono l'applicazione. Nel file docker-compose.yml, vengono definiti i due servizi principali: MongoDB, come il database dove verranno inseriti i dati elaborati, e l'applicazione Python responsabile dell'elaborazione e dell'inserimento dei dati. Questo file di configurazione permette di definire le dipendenze tra i servizi e di stabilire l'ordine di avvio, garantendo che MongoDB sia pronto e operativo prima che lo script Python inizi a eseguire le sue operazioni.

Durante il processo di deployment, l'esecuzione di "docker-compose build" compila l'immagine Docker utilizzando il Dockerfile specificato. Questo passaggio assicura che l'immagine contenga tutte le componenti necessarie, comprese le dipendenze Python e lo script di conversione, pronte per essere eseguite.

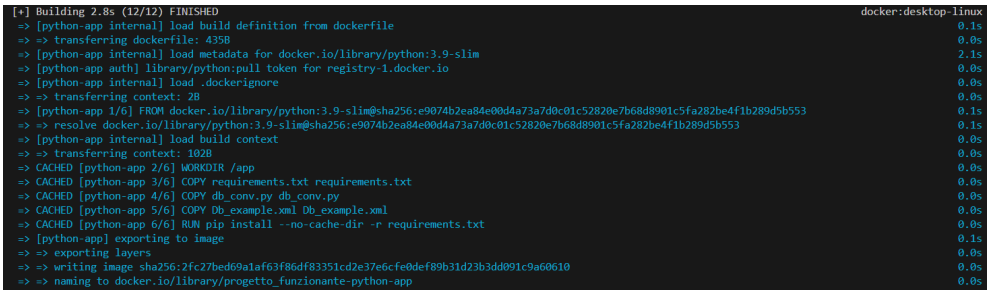

Figura 26: Output docker-compose build.

Una volta completata la costruzione, il comando "docker-compose up" avvia i container definiti nel file docker-compose.yml. Questa azione permette di orchestrare e gestire simultaneamente più servizi all'interno di un ambiente Dockerizzato, rendendo il deployment delle applicazioni più efficiente e con-

trollato. Il cuore di docker-compose up risiede nella capacità di interpretare e eseguire le istruzioni contenute nel file docker-compose.yml, un documento YAML che funge da blueprints per definire l'infrastruttura di deployment dell'applicazione. Questo file contiene tutte le informazioni necessarie su quali servizi devono essere avviati, quali immagini Docker devono essere utilizzate, come devono interagire tra loro e le configurazioni specifiche per ciascun servizio. Durante il processo di avvio dei container, Docker Compose si occupa anche di configurare le porte specificate nel docker-compose.yml. Questo include il mapping delle porte del container all'host locale, consentendo così l'accesso ai servizi tramite l'indirizzo IP e la porta appropriati. Inoltre, Docker Compose gestisce il montaggio dei volumi definiti nel file di configurazione. Questo permette di far persistere i dati generati dai container, come ad esempio i dati del database o i file di log dell'applicazione, garantendo che siano disponibili anche in caso di riavvio dei container stessi. Durante l'esecuzione del comando "docker-compose up", Docker Compose fornisce un output dettagliato delle operazioni in corso. Questo include i log di avvio dei container, eventuali avvisi o errori di configurazione e lo stato dei servizi in esecuzione. Questo feedback è essenziale per monitorare il corretto avvio e funzionamento dell'ambiente Dockerizzato, facilitando la risoluzione tempestiva di problemi eventualmente riscontrati.

Figura 27: Output docker-compose up.

5.2 Monitoraggio della creazione ed esecuzione dei container

La gestione e il monitoraggio della creazione ed esecuzione dei container Docker sono importanti per garantire un ambiente stabile ed efficiente per le applicazioni containerizzate. Docker Desktop emerge come uno strumento importante in questo contesto, offrendo un'interfaccia grafica che semplifica notevolmente il processo di gestione dei container Docker.Durante la fase di costruzione delle immagini Docker, è possibile seguire passo dopo passo l'avanzamento della compilazione, monitorando l'esecuzione di ciascuna istruzione definita nel Dockerfile. Questo è efficace per riuscire a identificare tempestivamente eventuali errori di configurazione e garantire che l'immagine venga creata correttamente. Una volta costruite le immagini, l'avvio dei container tramite il comando "docker-compose up" è possibile verificare facilmente lo stato di avvio dei container e controllare che tutti i servizi siano operativi senza dover ricorrere a complesse operazioni di controllo manuale. Durante l'esecuzione, l'accesso diretto ai log dei container consente di monitorare l'utilizzo delle risorse e di identificare eventuali problemi di performance o configurazione. Questo feedback dettagliato è prezioso per il debugging e l'ottimizzazione del sistema, migliorando la capacità di risolvere tempestivamente le problematiche che possono emergere durante l'operazione. È importante anche verificare che tutti i componenti del sistema siano avviati correttamente, compresi database come MongoDB, prima che lo script Python inizi il processo di parsing e inserimento dei dati. Questo passaggio preliminare assicura che tutte le dipendenze siano soddisfatte e che il sistema sia pronto per l'elaborazione dei dati senza interruzioni indesiderate.

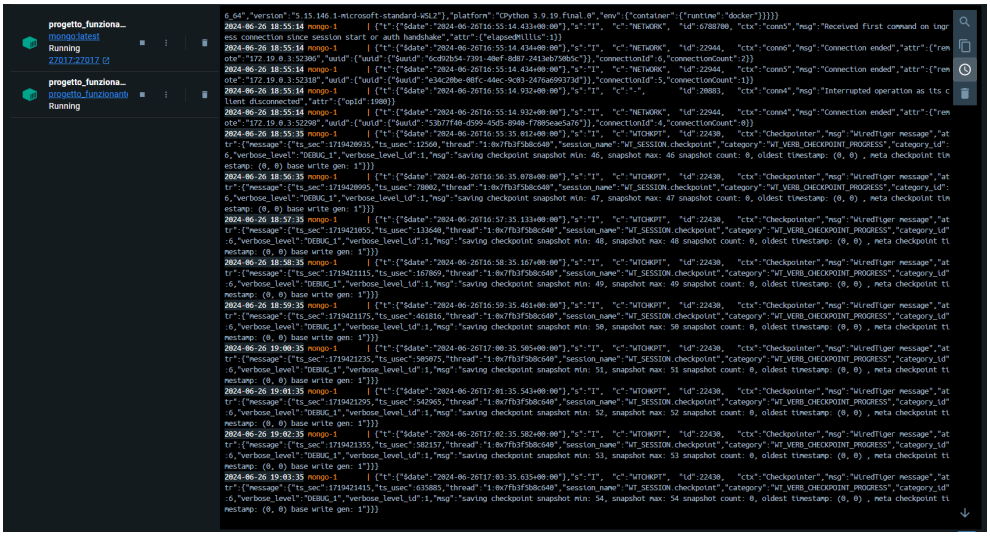

Figura 28: Visualizzazione container e log utilizzando Docker Desktop.

5.3 Visualizzazione dei risultati

Una volta che i container sono in esecuzione, lo script Python viene eseguito. Questo script è responsabile dell'analisi del file XML, dell'estrazione dei dati rilevanti e della loro pulizia per rimuovere eventuali informazioni non necessarie o formattazioni errate. Dopo aver preparato i dati, lo script procede con l'inserimento degli stessi in MongoDB, utilizzando una connessione al database stabilita attraverso il container MongoDB avviato in precedenza. Questo processo garantisce che i dati siano correttamente strutturati e memorizzati nel database, pronti per essere interrogati e analizzati. Con i dati correttamente inseriti nel database MongoDB, si passa alla fase di visualizzazione. MongoDB Compass offre una soluzione grafica ideale per chi preferisce un'interfaccia user-friendly. Per visualizzare i dati, è necessario avviare MongoDB Compass e connettersi al database specificando l'indirizzo del server (ad esempio, localhost:27017 se eseguito localmente). Una volta stabilita la connessione, si può navigare tra le diverse collezioni, visualizzare i documenti inseriti, esplorare la struttura dei dati e persino eseguire query

complesse e aggregazioni senza dover scrivere codice manualmente. Questa interfaccia grafica rende immediata e visiva l'analisi dei dati, facilitando l'individuazione di eventuali anomalie o la verifica della corretta importazione dei dati.

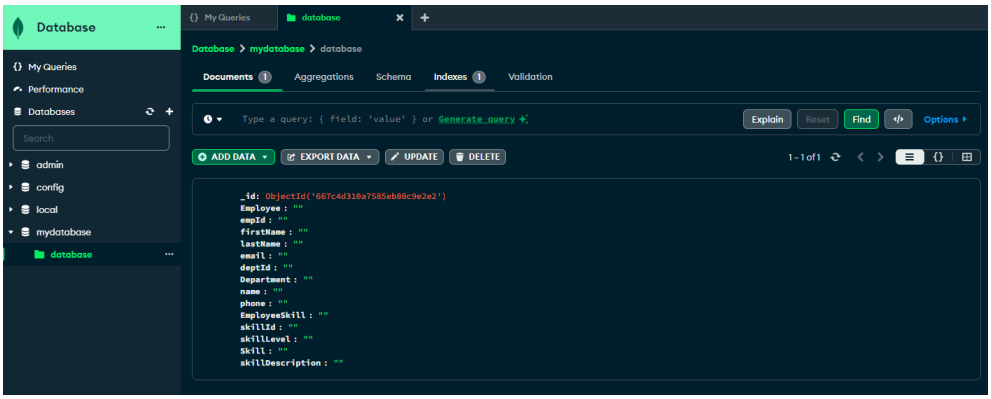

Figura 29: Visualizzazione dati su MongoDB Compass.

In alternativa, per chi preferisce un approccio più diretto e scriptabile, è possibile utilizzare la linea di comando del terminale. Con il comando mongo, ci si può connettere al database e utilizzare una vasta gamma di comandi per interagire con i dati. Ad esempio, eseguendo una query semplice come db.collection.find(), si possono visualizzare rapidamente i documenti presenti in una collezione specifica. La linea di comando offre un controllo dettagliato e potente, permettendo operazioni automatizzate e personalizzate per la gestione del database, e risulta particolarmente utile per gli sviluppatori che necessitano di un accesso rapido ai dati per eseguire debug o script di manutenzione.

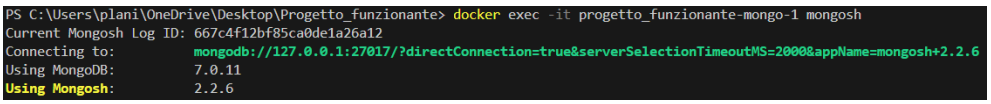

Figura 30: Esecuzione shell interattiva mongosh.

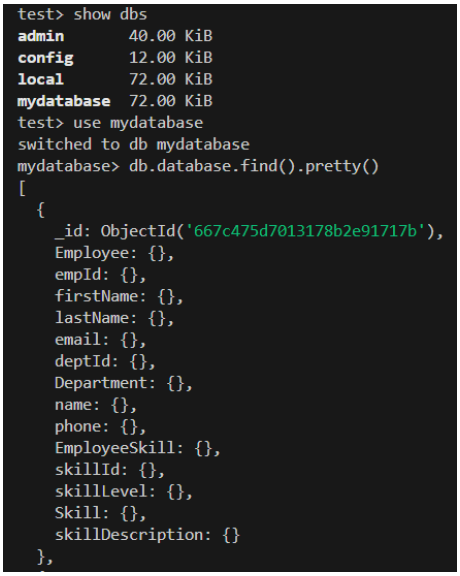

Figura 31: Visualizzazione dati da terminale.

Entrambi gli approcci, MongoDB Compass e il terminale, forniscono strumenti essenziali per garantire che i dati siano stati correttamente inseriti e per facilitare l'analisi e la manipolazione dei dati secondo le esigenze specifiche del progetto. Grazie a questi strumenti, è possibile assicurarsi che l'intero processo, dalla configurazione dell'ambiente Docker alla visualizzazione dei risultati, avvenga in modo fluido e senza intoppi, permettendo di sfruttare appieno le potenzialità di MongoDB per la gestione dei dati. Prima di concludere le operazioni, è necessario effettuare una verifica finale dei dati inseriti. Questa verifica deve essere accurata per assicurarsi che tutti i dati siano stati importati correttamente, senza errori o anomalie. È importante confermare che la struttura dei dati nel database rispecchi fedelmente le specifiche progettuali e che tutte le informazioni critiche siano presenti e corrette. Una volta verificati i dati e completata la documentazione, è importante procedere con la pulizia dell'ambiente Docker per liberare risorse e mantenere il sistema ordinato. I container in esecuzione possono essere arrestati utilizzando il comando docker-compose down, che interrompe tutti i servizi definiti nel file docker-compose.yml e rimuove i container associati. Questo passaggio aiuta

a prevenire sprechi di risorse e a mantenere l'ambiente di sviluppo pulito e organizzato. Inoltre, potrebbe essere utile rimuovere le immagini Docker non più necessarie tramite il comando docker image prune, per liberare ulteriore spazio su disco. Prima di concludere completamente le operazioni, è fondamentale garantire che i dati inseriti nel database MongoDB siano adeguatamente protetti. È consigliabile eseguire un backup completo del database utilizzando strumenti come mongodump, che consentono di creare copie di sicurezza dei dati. I backup dovrebbero essere conservati in location sicure e ridondanti per proteggere i dati da eventuali perdite accidentali o guasti del sistema.

5.4 Esempio implementativo

Una volta ottenuto il database mediante l'utilizzo del progetto realizzato ad hoc, il focus si e spostato sull'integrazione di alcune tecniche di deception utili a proteggere i dati reali e introdurre elementi fuorvianti e misure di sicurezza aggiuntive per ingannare eventuali attaccanti.

Per monitorare attentamente le operazioni eseguite durante l'esecuzione del programma, è stata configurata una robusta infrastruttura di logging. Il modulo di logging, è stato implementato per registrare in modo dettagliato ogni evento significativo, inclusi inserimenti di dati nel database e l'inserimento di honeytokens. Questo registro non solo facilita il monitoraggio continuo delle attività del sistema, ma risulta molto utile per la gestione degli audit e la risposta a eventuali incidenti di sicurezza.

```
import logging

# Configurazione del logging
logging.basicConfig(filename='/app/deception.log', level=logging.INFO, format='%(asctime)s %(message)s')
```

Figura 32: Aggiunta del modulo di "logging".

Il codice è stato progettato anche con l'obiettivo di mitigare i potenziali rischi di connessioni interrotte o non riuscite. Quando il programma tenta di stabilire una connessione con il server MongoDB, è predisposto a gestire diversi scenari di errore. Se la connessione iniziale non riesce, il sistema non si arrende immediatamente; al contrario, effetua ulteriori tentativi, consentendo al server il tempo necessario per rispondere e stabilizzarsi. Questo approccio non solo incrementa la resilienza del sistema, ma anche la continuità operativa e alla sicurezza dei dati. Ogni tentativo di connessione è strategicamente distanziato per evitare un sovraccarico del server e per massimizzare la probabilità di successo. Inoltre, il ritardo tra i tentativi assicura che il server MongoDB sia completamente pronto a ricevere e gestire le connessioni in arrivo, riducendo al minimo il rischio di congestione o di interruzioni improvvise durante l'elaborazione dei dati.

Per garantire la sicurezza e la privacy dei dati estratti dal file XML, è stato implementato un processo di offuscamento attraverso l'uso di una funzione dedicata chiamata "obfuscate-data". Questa procedura è stata progettata per proteggere informazioni sensibili come nomi, identificativi dipartimentali e altre dati rilevanti, che potrebbero essere vulnerabili in caso di accesso non autorizzato. Ogni valore estratto dal file XML, e quindi presente nel database di partenza, è stato sostituito da una versione pseudonimizzata, identificata dal prefisso "obfuscated". Questa pratica protegge la riservatezza delle informazioni sensibili, e impedisce anche la divulgazione involontaria di dati critici. Anche se un potenziale attaccante dovesse riuscire ad accedere al sistema, le informazioni pseudonimizzate non gli permetterebbero di identificare direttamente i dettagli specifici dei dati. Ovviamente l'offuscamento dei dati non influisce negativamente sull'utilità dei dati stessi. I dati rimangono accessibili e utilizzabili internamente, garantendo allo stesso tempo un livello elevato di protezione e sicurezza.

```python
# Funzione per offuscare i dati
def obfuscate_data(data):
    return {key: f"obfuscated_{key}" for key in data}
```

Figura 33: Funzione di offuscamento dei dati.

Oltre alla funzione di offuscamento, nel contesto della difesa attiva del nostro sistema di gestione dati basato su MongoDB, è integrata una strategia innovativa di sicurezza attraverso l'inserimento di honeytokens. Questi elementi sono stati progettati appositamente per identificare e contrastare possibili tentativi di accesso non autorizzato o manipolazione malevola dei dati, costituendo vere e proprie trappole per potenziali attaccanti. Gli honeytokens, sono dati falsi deliberatamente creati e distribuiti all'interno del nostro database MongoDB. La loro presenza mira a ingannare e attirare chi tenta di compromettere il sistema, offrendo così una strategia difensiva proattiva contro le minacce informatiche [6]. Ogni honeytoken è attentamente posizionato per mimetizzarsi tra i dati reali, creando un ambiente in cui ogni accesso o tentativo di manipolazione sospetto viene immediatamente rivelato. L'efficacia degli honeytokens si estende oltre la semplice rilevazione delle attività anomale. Ogni interazione con un honeytoken è accuratamente registrata nel nostro registro delle attività, noto come deception.log, consultabile mediante l'utilizzo del comando "cat deception.log". Questo registro fornisce una traccia dettagliata di ogni tentativo di accesso non autorizzato o di manipolazione dei dati, facilitando così l'analisi e la risposta tempestiva alle minacce.

```python
# Funzione per l'inserimento degli honeytokens
def insert_honey_tokens(collection):
    honey_tokens = [
        {"fake_data": "honeypot1", "description": "This is a honey token."},
        {"fake_data": "honeypot2", "description": "This is another honey token."}
    ]
    collection.insert_many(honey_tokens)
    logging.info("Honey tokens inserted.")
```

Figura 34: Funzione di inserimento Honeytoken.

```
PS C:\Users\plani\OneDrive\Desktop\Progetto_funzionante> cat deception.log
2024-07-13 09:47:09,309 Inserted data: {'Employee': 'obfuscated_Employee', 'empId': 'obfuscated_empId', 'firstName': 'obfuscated_firstName', 'lastName': 'o
bfuscated_lastName', 'email': 'obfuscated_email', 'deptId': 'obfuscated_deptId', 'Department': 'obfuscated_Department', 'name': 'obfuscated_name', 'phone':
 'obfuscated_phone', 'EmployeeSkill': 'obfuscated_EmployeeSkill', 'skillId': 'obfuscated_skillId', 'skillLevel': 'obfuscated_skillLevel', 'Skill': 'obfusca
ted_Skill', 'skillDescription': 'obfuscated_skillDescription', '_id': ObjectId('66924d1d1f3b13504b909bd9')}
2024-07-13 09:47:09,711 Honey tokens inserted.
```

Figura 35: Comando per la visualizzazione dei log.

Questa pratica non solo aumenta la sicurezza complessiva del nostro sistema, ma rappresenta anche un deterrente potente contro gli attacchi informatici.

L'integrazione delle tecniche di deception nel nostro sistema aumenta notevolmente la sicurezza dei dati. Attraverso l'implementazione dell'offuscamento dei dati e l'inserimento strategico di honeytokens, è stata notevolmente potenziata la nostra capacità di proteggere informazioni sensibili e di rispondere in modo proattivo alle minacce informatiche emergenti. Il processo di offuscamento dei dati, mediante l'utilizzo della funzione "obfuscate-data", ha rivestito ogni informazione critica estratta dal file XML con una pseudonimizzazione sicura. Questo approccio non solo ha ridotto la vulnerabilità dei dati sensibili a potenziali accessi non autorizzati, ma rappresenta anche una misura chiave per la conformità normativa e la protezione della privacy. Parallelamente, l'inserimento degli honeytokens ha introdotto un livello aggiuntivo di difesa attiva nel nostro sistema. Questi dati falsi, appositamente creati e distribuiti all'interno del database MongoDB, fungono da trappole per attirare e rivelare attività sospette degli attaccanti. Ogni interazione

con un honeytoken è accuratamente registrata nel "deception.log", fornendo una traccia chiara e dettagliata che facilita l'identificazione e l'analisi delle minacce informatiche.

Capitolo 6

Possibili applicazioni

Nel contesto sempre più complesso della cybersecurity, la creazione di strategie efficaci per proteggere le informazioni sensibili e mitigare le minacce informatiche è di grande importanza per le organizzazioni. Questo capitolo esplora il potenziale del progetto nell'ambito deception, focalizzandosi su come i fake database possano essere utilizzati per attirare, intrappolare e sconfiggere potenziali attaccanti. Esamineremo le diverse applicazioni pratiche di questa tecnologia, evidenziando i benefici per la cybersecurity, la formazione del personale e la ricerca tecnologica avanzata. Attraverso un'analisi approfondita dei casi d'uso proposti, sarà possibile comprendere come questa soluzione possa contribuire significativamente alla cyber defense delle organizzazioni.

6.1 Vantaggi della generazione di fake database

Uno dei tratti distintivi del progetto è la capacità di creare fake database, con l'obiettivo di confondere e intrappolare potenziali attaccanti:

- `Attrazione di attaccanti:` Creare fake database con dati che sembrano autentici rappresenta una strategia sofisticata per attirare attaccanti desiderosi di ottenere informazioni sensibili. Questo approccio

permette di monitorare attentamente l'interazione degli attaccanti con i dati simulati, raccogliendo informazioni dettagliate sulle loro tattiche e obiettivi. I fake database fungono da trappole digitali, progettate per apparire come obiettivi legittimi e di alto valore, inducendo gli attaccanti a rivelare le loro intenzioni e tecniche. I fake database servono come primi indicatori di compromissione. Quando un attaccante interagisce con un fake database, i sistemi di sicurezza possono rilevare queste attività anomale in tempo reale. Questo rilevamento precoce permette di identificare rapidamente le minacce e di attuare misure di contenimento immediate per proteggere i dati reali. Ad esempio un attaccante che tenta di accedere a un fake database potrebbe eseguire query di esplorazione per mappare la struttura del database. Queste query possono essere rilevate e analizzate, fornendo un avviso tempestivo della presenza di un attaccante nella rete. Analizzando le interazioni degli attaccanti con i fake database, è possibile raccogliere informazioni dettagliate sulle loro tecniche, tattiche e procedure (TTPs).

- `Simulazione di dati sensibili:` I fake database possono contenere informazioni che simulano dati sensibili e di alto valore, come informazioni finanziarie, proprietà intellettuali o dati personali. Questi dati sono progettati per fungere da esche efficaci, attirando e ingannando gli attaccanti. Monitorare le interazioni con questi dati simulati consente di identificare comportamenti anomali o tentativi di accesso non autorizzato. Per esempio, un fake database potrebbe contenere dati fittizi su conti bancari. Se un attaccante tenta di trasferire fondi da uno di questi conti, il tentativo può essere immediatamente rilevato e bloccato, mentre il comportamento dell'attaccante viene analizzato per migliorare le difese del sistema reale.

- `Test di sicurezza avanzati:` Durante i test di sicurezza, l'utilizzo di fake database offre un ambiente controllato per valutare le vulnerabilità del sistema senza rischiare l'esposizione di dati sensibili reali.

Questo approccio consente di identificare e correggere le vulnerabilità prima che possano essere sfruttate da attaccanti effettivi. Ad esempio, durante un penetration test, gli analisti di sicurezza possono utilizzare fake database per simulare scenari di attacco realistici. In questo modo, è possibile testare l'efficacia delle misure di sicurezza implementate, come firewall, sistemi di rilevamento delle intrusioni e meccanismi di autenticazione, assicurandosi che siano in grado di resistere a tentativi di compromissione.

6.2 Implementazione di Honeypot e Honeynet

L'integrazione di fake database all'interno di un honeypot permette di fornire dati che appaiono credibili ma sono, in realtà, falsi. Quando un intrusore interagisce con questi dati, rivela le sue intenzioni e le sue modalità operative. Questo processo di interazione non solo permette di studiare e comprendere meglio le tecniche utilizzate dagli attaccanti, ma consente anche di raccogliere prove che possono essere utilizzate per rafforzare le difese del sistema. Inoltre, i dati raccolti possono essere utilizzati per migliorare le capacità di rilevamento e risposta alle minacce.

Estendendo il concetto di honeypot, una honeynet è una rete di honeypot interconnessi che collaborano per creare un ambiente ancora più complesso e realistico [11], [8]. In una honeynet, i fake database possono essere distribuiti in diversi punti della rete, simulando vari servizi e applicazioni utilizzati all'interno dell'organizzazione. Questo approccio permette di creare un ambiente di rete che sembra autentico e operativo, inducendo gli attaccanti a rivelare comportamenti avanzati che potrebbero non essere visibili in un singolo honeypot. L'analisi delle operazioni su larga scala in una honeynet fornisce informazioni preziose che possono essere utilizzate per rafforzare le misure di sicurezza. Ad esempio, le informazioni raccolte possono aiutare a

identificare pattern di attacco comuni, strumenti utilizzati dagli attaccanti e strategie di evasione.

Per quanto riguarda le possibili migliorie applicabili al progetto, l'adozione di nuove tecnologie e l'implementazione di ulteriori funzionalità possono accrescere le capacità, l'efficienza e la sicurezza del sistema. L'obiettivo è delineare una roadmap per l'evoluzione del progetto, mantenendo un approccio flessibile e orientato all'innovazione, considerando anche i benefici derivanti dalla containerizzazione e dalle tecniche di deception.

6.3 Automazione avanzata e integrazione continua

L'automazione dei processi e l'integrazione continua (CI) rappresentano fondamentali strumenti per migliorare l'efficienza e la qualità del progetto. Ecco alcune possibili aree di miglioramento:

- `Automazione dei test:` L'integrazione di un framework di testing automatizzato consentirebbe di eseguire test unitari, di integrazione e di sistema su ogni componente del progetto. Questi test garantirebbero una maggiore affidabilità del codice, riducendo il tempo necessario per la verifica manuale delle funzionalità. Strumenti come pytest o unittest in Python potrebbero essere utilizzati per automatizzare i test delle funzioni critiche del progetto. L'automazione dei test permette di rilevare e correggere rapidamente eventuali bug, migliorando la stabilità del sistema e facilitando l'implementazione di nuove funzionalità senza compromettere quelle esistenti.

- `Integrazione Continua (CI):` L'implementazione di una pipeline CI automatizzata, utilizzando strumenti come Jenkins, Travis CI o GitHub Actions, permetterebbe di automatizzare il processo di build, test e de-

ployment. Questo migliorerebbe la coerenza e la velocità del rilascio delle nuove versioni del software, riducendo al minimo gli errori umani e garantendo un feedback immediato sulle modifiche al codice. L'integrazione continua permette di mantenere un codice sempre aggiornato e testato, riducendo i tempi di sviluppo e migliorando la qualità del prodotto finale.

6.4 Scalabilità e prestazioni

Per supportare la crescita del sistema e gestire carichi di lavoro crescenti, è essenziale considerare migliorie in termini di scalabilità e prestazioni:

- `Scalabilità Orizzontale:` L'adozione di tecnologie di orchestrazione dei container come Kubernetes permetterebbe di scalare orizzontalmente il sistema, distribuendo il carico di lavoro su più nodi e garantendo un'alta disponibilità e tolleranza ai guasti. Questo consentirebbe di gestire picchi di traffico e di assicurare la continuità del servizio anche in caso di guasti hardware. Kubernetes automatizza il deployment, la gestione e la scalabilità delle applicazioni containerizzate, rendendo più facile gestire grandi cluster di container e migliorare la resilienza del sistema.

- `Ottimizzazione delle Prestazioni del Database:` L'ottimizzazione delle query e l'implementazione di tecniche di caching avanzate, come Redis o Memcached, migliorerebbero le prestazioni complessive del sistema. Inoltre, la partizione dei dati e l'adozione di indici appropriati potrebbero ridurre significativamente i tempi di risposta del database. Ad esempio, il miglioramento dell'uso degli indici in MongoDB può ridurre il tempo necessario per le operazioni di lettura e scrittura. La partizione dei dati suddivide un grande database in segmenti più piccoli e gestibili, migliorando le prestazioni e facilitando la scalabilità.

6.5 Containerizzazione e DevOps

L'adozione completa delle pratiche DevOps e l'ulteriore utilizzo della containerizzazione possono portare a numerosi vantaggi:

- `Monitoraggio e logging avanzati:` L'integrazione di strumenti di monitoraggio e logging avanzati, come Prometheus, Grafana ed ELK Stack (Elasticsearch, Logstash, Kibana), permetterebbe di monitorare in tempo reale le prestazioni del sistema e di analizzare i log per identificare rapidamente eventuali problemi. Questo approccio garantirebbe una visibilità completa sul funzionamento del sistema, facilitando la diagnosi e la risoluzione dei problemi. Grafana, ad esempio, può essere utilizzato per creare dashboard personalizzate che visualizzano le metriche di sistema in tempo reale, mentre ELK Stack può raccogliere e analizzare i log da diverse fonti, offrendo una panoramica dettagliata delle operazioni del sistema.

- `Continuous Deployment (CD):` L'estensione della pipeline CI con funzionalità di continuous deployment (CD) consentirebbe di automatizzare il rilascio delle nuove versioni del software direttamente negli ambienti di produzione. Questo migliorerebbe la reattività del team di sviluppo, riducendo i tempi di deployment e minimizzando i rischi associati ai rilasci manuali. Il CD garantisce che ogni modifica al codice venga automaticamente testata e distribuita, riducendo il tempo di rilascio e migliorando la qualità complessiva del software.

6.6 Integrazione con tecnologie emergenti

L'adozione di tecnologie emergenti potrebbe aprire nuove opportunità per il progetto:

- `Intelligenza Artificiale e Machine Learning:` L'integrazione di algoritmi di intelligenza artificiale (AI) e machine learning (ML) potrebbe migliorare le capacità di analisi del sistema. Ad esempio, nel

rilevamento delle minacce e nell'ottimizzazione delle risorse, modelli di apprendimento automatico potrebbero essere utilizzati per prevedere comportamenti anomali e suggerire azioni correttive. L'AI potrebbe anche automatizzare la risposta agli incidenti di sicurezza, riducendo il tempo di reazione. Algoritmi di machine learning possono analizzare grandi quantità di dati per identificare pattern e anomalie, migliorando la capacità del sistema di rilevare e prevenire attacchi.

- `Blockchain:` L'utilizzo della tecnologia blockchain per garantire l'integrità e la trasparenza delle transazioni e dei dati potrebbe aggiungere un ulteriore livello di sicurezza al sistema. Le blockchain potrebbero essere utilizzate per tracciare e verificare le modifiche ai dati sensibili, prevenendo frodi e manomissioni. Ad esempio, l'uso di smart contracts potrebbe automatizzare e proteggere le transazioni finanziarie all'interno del sistema. Le blockchain offrono un registro immutabile delle transazioni, che può essere utilizzato per verificare l'autenticità dei dati e garantire la trasparenza delle operazioni.

Il progetto ha il potenziale per evolversi e migliorare continuamente attraverso l'adozione di nuove tecnologie e l'implementazione di ulteriori funzionalità. Automazione avanzata, sicurezza potenziata, scalabilità migliorata, integrazione DevOps e l'adozione di tecnologie emergenti rappresentano aree chiave per futuri sviluppi. Continuare a innovare e adattarsi alle nuove sfide e opportunità sarà fondamentale per garantire che il sistema rimanga robusto, efficiente tempo. Inoltre, l'adozione delle tecniche nuove di deception non solo migliorano la sicurezza ma anche la flessibilità e l'efficienza del sistema, rendendolo più resiliente e capace di affrontare un panorama di minacce in continua evoluzione. L'integrazione di tecnologie avanzate, come l'AI e la blockchain, potrebbe aprire nuove opportunità per migliorare ulteriormente le capacità del sistema, rendendolo ancora più competitivo e all'avanguardia nel settore.

Capitolo 7

Conclusioni

Questo lavoro di tesi si è concentrato sull'approfondimento e sull'implementazione di strategie avanzate di cybersecurity, con un focus particolare sulla deception, integrate con tecnologie moderne come la containerizzazione con Docker e la gestione dei dati con MongoDB. L'obiettivo principale è stato esplorare come queste tecnologie possano essere efficacemente utilizzate per migliorare la sicurezza informatica, ottimizzare le operazioni aziendali e rafforzare la gestione dei dati nelle infrastrutture IT moderne. Durante questo percorso, è emerso chiaramente il ruolo importante della containerizzazione con Docker. Questa tecnologia ha consentito la creazione di ambienti isolati e portabili, facilitando il deployment delle applicazioni in modo efficiente e sicuro. La capacità di Docker di separare le applicazioni in container ha migliorato notevolmente la gestione delle risorse, riducendo i conflitti di dipendenze e garantendo una maggiore continuità operativa. Le strategie di deception, integrate nel progetto mediante l'uso di offuscamento dei dati e l'inserimento di honeytokens, hanno dimostrato di essere efficaci nel rilevare e monitorare le attività dei potenziali attaccanti, fornendo un'importante linea di difesa contro le minacce informatiche. L'inserimento di honeytokens nel database MongoDB ha aggiunto un livello di sicurezza supplementare, creando trappole per identificare attività anomale e fornendo evidenze nel registro delle attività per l'analisi forense e la risposta alle minacce. Guardando al

futuro, ci sono numerose opportunità per ampliare e raffinare le tecnologie esaminate in questo studio. L'ottimizzazione delle performance dei container, l'integrazione di tecniche avanzate di deception e l'implementazione di strategie di monitoraggio avanzato rappresentano sfide e potenzialità per ulteriori sviluppi. In conclusione, questo studio ha fornito un contributo significativo alla comprensione e all'applicazione delle strategie di deception e delle tecnologie associate nella cyber defense. L'integrazione efficace di queste tecnologie rappresenta un passo avanti nella creazione di sistemi IT più sicuri, efficienti e adattabili. Continuare a perseguire l'innovazione e la ricerca è fondamentale per affrontare con successo le sfide emergenti nel panorama sempre più complesso della cybersecurity.

Bibliografia

[1] Eric Alata, Vincent Nicomette, Mohamed Kaâniche, Marc Dacier, and Matthieu Herrb. Lessons learned from the deployment of a high-interaction honeypot. In *2006 Sixth European Dependable Computing Conference*, pages 39–46. IEEE, 2006.

[2] Massimiliano Albanese, Ermanno Battista, and Sushil Jajodia. Deceiving attackers by creating a virtual attack surface. *Cyber Deception: Building the Scientific Foundation*, pages 167–199, 2016.

[3] Adam Barth, Collin Jackson, Charles Reis, TGC Team, et al. The security architecture of the chromium browser. In *Technical report*. Stanford University, 2008.

[4] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE cloud computing*, 1(3):81–84, 2014.

[5] Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[6] Jin-Hee Cho, Dilli P Sharma, Hooman Alavizadeh, Seunghyun Yoon, Noam Ben-Asher, Terrence J Moore, Dong Seong Kim, Hyuk Lim, and Frederica F Nelson. Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Communications Surveys & Tutorials*, 22(1):709–745, 2020.

[7] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Know your enemy: Honeytokens. In *Proceedings of the 19th USENIX Security Symposium*, Washington, DC, August 2010. USENIX Association.

[8] Wenjun Fan, Zhihui Du, and David Fernández. Taxonomy of honeynet solutions. In *2015 SAI Intelligent Systems Conference (IntelliSys)*, pages 1002–1009. IEEE, 2015.

[9] Kimberly J Ferguson-Walter, Maxine M Major, Chelsea K Johnson, and Daniel H Muhleman. Examining the efficacy of decoy-based and psychological cyber deception. In *30th USENIX security symposium (USENIX Security 21)*, pages 1127–1144, 2021.

[10] Tal Garfinkel, Mendel Rosenblum, et al. A virtual machine introspection based architecture for intrusion detection. In *Ndss*, volume 3, pages 191–206. San Diega, CA, 2003.

[11] Zhang Heng-ru and Gong Jie. Research and design of network attack and defense platform based on virtual honeynet. In *2010 International Conference on Computational and Information Sciences*, pages 507–510. IEEE, 2010.

[12] Thorsten Holz. Learning more about attack patterns with honeypots. In *Sicherheit*, pages 30–41, 2006.

[13] Michael Httermann. *DevOps for developers*. Apress, 2012.

[14] Md Hasan Ibrahim, Mohammed Sayagh, and Ahmed E Hassan. A study of how docker compose is used to compose multi-component systems. *Empirical Software Engineering*, 26:1–27, 2021.

[15] Sushil Jajodia, V Subrahmanian, Vipin Swarup, and Cliff Wang. Cyber deception. *Springer*, 1:2–1, 2016.

[16] Abhishek Mairh, Debabrat Barik, Kanchan Verma, and Debasish Jena. Honeypot in network security: a survey. In *Proceedings of the 2011 in-*

ternational conference on communication, computing & security, pages 600–605, 2011.

[17] Dirk Merkel et al. Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2, 2014.

[18] Daniele Miorandi et al. Honeytokens: The other side of deception. *IEEE Security & Privacy*, 10(4):62–69, 2012.

[19] S Mukkamala, K Yendrapalli, R Basnet, MK Shankarapani, and AH Sung. Detection of virtual environments and low interaction honeypots. In *2007 IEEE SMC Information Assurance and Security Workshop*, pages 92–98. IEEE, 2007.

[20] Keshnee Padayachee. Aspectising honeytokens to contain the insider threat. *IET Information Security*, 9(4):240–247, 2015.

[21] Giulio Pagnotta, Fabio De Gaspari, Dorjan Hitaj, Mauro Andreolini, Michele Colajanni, and Luigi V Mancini. Dolos: A novel architecture for moving target defense. *IEEE Transactions on Information Forensics and Security*, 2023.

[22] Claus Pahl. Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31, 2015.

[23] AB Robert Petrunić. Honeytokens as active defense. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1313–1317. IEEE, 2015.

[24] Niels Provos, Markus Friedl, and Peter Honeyman. Preventing privilege escalation. In *12th USENIX Security Symposium (USENIX Security 03)*, 2003.

[25] Niels Provos and Thorsten Holz. *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, 2007.

[26] Python Software Foundation. *xml.etree.ElementTree*, 2023. Python Documentation (version 3.10).

[27] Rowe, Duong, and Custy. Fake honeypots: A defensive tactic for cyberspace. In *2006 IEEE Information Assurance Workshop*, pages 223–230. IEEE, 2006.

[28] Lance Spitzner. The value of honeypots in network security. Technical report, SANS Institute, 2002.

[29] Lance Spitzner. *Honeypots: tracking hackers*, volume 1. Addison-Wesley Reading, 2003.

[30] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[31] Jim Yuill, Dorothy Denning, and Fred Feer. Using deception to hide things from hackers: Processes, principles, and techniques. *Journal of Information Warfare*, 5(3):26–40, 2006.