UNIVERSITÀ DEGLI STUDI DI UDINE

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

TESI DI LAUREA IN INFORMATICA

Sviluppo di un Cyber Range: configurazione scenari attraverso lo standard TOSCA

CANDIDATO
Filippo Merlo

RELATORE
Prof. Marino Miculan

Anno accademico 2021-2022

Contatti dell'Istituto
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

© 2023 Filippo Merlo
This work is shared under the Creative Commons 4.0 License Attribution-NonCommercial-ShareAlike.

Ringraziamenti

Alcune volte non ci sono modi per dire grazie, ma ci tengo comunque a ringraziare di cuore la mia famiglia e la mia fidanzata Arianna per avermi dato tutto l'amore di cui avevo bisogno.

Indice

1 Introduzione **1**
1.1 Requisiti . . . 2
1.2 Struttura del documento . . . 3

2 Cyber Range **5**
2.1 Definizione . . . 5
2.2 SIEM . . . 7
2.3 SDL . . . 8
2.4 Soluzioni open source di riferimento . . . 8
2.4.1 CRACK . . . 8
2.4.2 DigitalTwinSocCyberrange . . . 9

3 Analisi del problema e progetto della soluzione **11**
3.1 Casi d'uso . . . 11
3.2 Requisiti . . . 12
3.3 Architettura . . . 13
3.3.1 TOSCA . . . 14
3.3.2 Puccini . . . 14
3.3.3 Terraform . . . 14
3.3.4 Docker . . . 15
3.3.5 Scenario . . . 15
3.4 L'architettura MAPE-K . . . 16
3.4.1 Concretizzare MAPE-K . . . 17

4 Implementazione **19**
4.1 TOSCA . . . 19
4.1.1 Introduzione . . . 19
4.1.2 Struttura . . . 19
4.2 Puccini . . . 24
4.2.1 Puccini-Clout . . . 26

5 Test e validazione **29**
5.1 Test e validazione SDL . . . 29
5.1.1 Validazione . . . 32
5.2 Test e validazione SIEM . . . 32
5.2.1 Host Header Poisoning . . . 32
5.2.2 Remote Code Execution . . . 33
5.2.3 SQL injection . . . 34

6 Conclusioni **37**
6.1 Considerazioni finali . . . 37
6.2 Limitazioni e possibili miglioramenti . . . 37

Elenco delle figure

1.1 Attacchi per semestre dal rapporto ottobre 2022 Clusit [9] 2

2.1 Schema di un Cyber Range relativo al documento [22] 6

3.1 Casi d'uso più frequenti 12
3.2 Diagramma dei casi d'uso 13
3.3 L'architettura del progetto 14
3.4 Architettura di Terraform 15
3.5 Struttura dei container Docker 15
3.6 Diagramma MAPE-K [18] 16

4.1 Diagramma del frammento di codice 4.1.2 sulle relazioni di una rete 21

5.1 Scenario di esempio 30
5.2 Attacco rilevato in DSIEM 32
5.3 Rilevazione Remote Code Execution 34
5.4 L'allarme di Suricata in Kibana in seguito a SQL Injection 34

1

Introduzione

Gli attacchi e la sicurezza informatica sono stati tra gli argomenti più trattati da giornali, televisione e sono stati fonte di pubblici dibattiti e confronti sia nel mondo televisivo che nelle piattaforme social. Recentemente è stata posta molta attenzione sull'allerta di possibili attacchi rivolti ad enti governativi e strutture sanitarie nei confronti degli Stati Uniti e dell'Europa, in particolar modo contro Francia, Svezia e Italia. Secondo l'ultimo rapporto di Ottobre 2022 pubblicato dalla Clusit [9], solamente nella prima metà del 2022, sono stati ben 1.141 gli attacchi informatici gravi, ovvero con un impatto sistemico in diversi aspetti della società, della politica e dell'economia, con una crescita pari all'8,4% rispetto allo stesso periodo del 2021.

Come si può osservare nel grafico 1.1, il numero di attacchi informatici continua a crescere linearmente di anno in anno. In particolar modo, secondo una valutazione più dettagliata degli attacchi ricevuti, i ricercatori della Clusit sono riusciti a classificarli secondo il livello d'impatto ed hanno osservato come, sempre considerando il primo semestre 2022, quelli **gravi** sono stati il 45% del totale, mentre quelli **critici** sono arrivati ben al 33%, andando così ad evidenziare come il **78%** del totale sia rappresentato da attacchi gravi e critici, mentre solo il 22% siano lievi o moderati.

I cambiamenti su scala globale avvenuti negli ultimi anni, tra cui la pandemia da Covid-19 ed il conflitto russo-ucraino, hanno influenzato maggiormente il settore informatico, portando a profondi cambiamenti in molte categorie, tra cui quello della sicurezza informatica. Ciò è stato evidenziato anche nel rapporto Clusit, in cui viene mostrata un'impennata del 414% nella categoria dell'*hacktivismo* e del 119% in quella dell'*information warfare*, mentre gli attacchi che rientrano nella categoria del *cybercrime* sono diminuiti del 3,4%. Nonostante la decrescita, il *cybercrime* rimane comunque la motivazione principale degli attacchi informatici, in quanto rappresenta ancora il 78,4% del totale.

Il termine cybercrime fa riferimento ad ogni reato nel quale la condotta o l'oggetto materiale del crimine sono correlati ad un sistema informatico o telematico, ovvero perpetrato utilizzando un tale sistema o colpendolo [3]. La maggior parte dei cyber crimini vengono effettuati a scopo di lucro e nella restante parte di casi per motivi politici o personali.

La formazione di personale esperto nella difesa e protezione da possibili attacchi informatici diventa quindi una priorità e a tale scopo sono necessari strumenti adeguati che permettano di sperimentare sul campo le tecniche di attacco e difesa in modo da prevenire, o almeno limitare, i danni provocati dalle varie offensive. L'importanza di tali tematiche risulta evidente andando ad osservare le misure intraprese

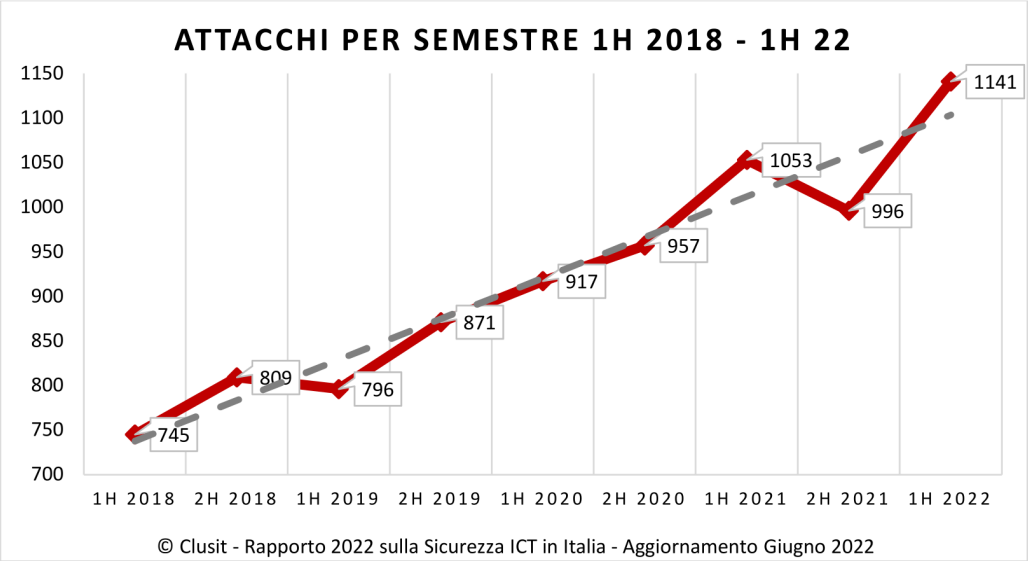


Figura 1.1: Attacchi per semestre dal rapporto ottobre 2022 Clusit [9]

dal nostro governo in merito, prima fra tutte l'investimento di 623 milioni di euro (tramite il PNRR[1]) per il potenziamento dell'ecosistema digitale nazionale entro il 2024. [11, 4]

Nasce quindi la necessità di una soluzione aperta e multipiattaforma per aumentare l'alfabetizzazione in tema cybersecurity, che faccia appoggio su tecnologie dalla comprovata solidità e liberamente disponibili, in modo da abbattere barriere economiche e tecnologiche, permettendone l'installazione nel maggior numero di contesti possibile. A questo stanno lavorando già altre Università italiane, fra le quali si cita l'Università degli Studi di Genova con la quale c'è stato un utile confronto nelle prime fasi del lavoro.

1.1 Requisiti

La proposta di progetto è lo sviluppo di una pipeline per la configurazione ed il deployment di un Cyber Range sfruttando soluzioni open-source già presenti sul mercato, estese con moduli software *ad-hoc*, che permettono la sinergia delle diverse componenti.

Nello sviluppo della nostra proposta di un Cyber Range ci siamo concentrati su 3 punti chiave:

- scalabilità
- repricabilità
- semplicità

Si è voluto realizzare un prodotto che fosse personalizzabile, estensibile e modificabile sotto ogni suo aspetto, in modo da fornire uno strumento utilizzabile indipendentemente dall'infrastuttura su cui esso viene applicato, data la disomogeneità nei servizi, strumenti e configurazioni usati nelle singole aziende. È stato prioritario puntare quindi ad una soluzione che avesse una struttura semplice e versatile, senza troppe dipendenze nelle componenti della sua architettura, per facilitarne la replicabilità e l'utilizzo,

[1] Piano Nazionale di Ripresa e Resilienza

in modo da gettare le basi per lo sviluppo di una piattaforma con la potenzialità di essere in futuro standardizzata.

Punto cardine del progetto è offrire un modo quanto più accessibile agli istruttori in cybersecurity per costruire gli ambienti in cui mettere in pratica gli addestramenti; gli ambienti vengono quindi progettati prima attraverso un linguaggio formale e poi realizzati nel concreto da una suite di software in grado di automatizzare gran parte del processo, riducendo così il gap fra la fase di progettazione e quella di deployment.

1.2 Struttura del documento

Nei prossimi capitoli verranno illustrate le architetture della SDL 2.3 e quella della piattaforma SIEM 2.2, con un particolare focus sullo standard TOSCA.

Nel capitolo 2 sono esposte le caratteristiche di un Cyber Range e alcune soluzioni già presenti sul mercato che sono state prese come riferimento. Il capitolo 3 contiene l'analisi dei casi d'uso e i requisiti derivati da essi, oltre che la descrizione delle architetture scelte sia per realizzare la pipeline atta a costruire gli ambienti di addestramento, che per la parte di monitoraggio ed analisi degli eventi. Nel capitolo 4 viene descritta la struttura e la sintassi di TOSCA, con un approfondimento ulteriore sul funzionamento dello strumento Puccini, per infine passare allo studio di alcune dichiarazioni specifiche realizzate per la creazione degli scenari necessari al testing del funzionamento e della solidità dello strumento di deployment. Gli scenari di test realizzati al termine del progetto sono stati analizzati al capitolo 5, con la conseguente verifica che gli obiettivi posti siano stati raggiunti. Infine nel capitolo 6 vengono sottolineate le problematiche sorte durante lo sviluppo ed i possibili miglioramenti futuri da integrare.

2

Cyber Range

2.1 Definizione

Un Cyber Range è un'infrastruttura dedicata al test e alla validazione di strumenti e scenari nell'ambito della sicurezza informatica.

I Cyber Range sono nati dalla necessità di colmare la differenza di conoscenze ed abilità tra uno studente ed un professionista informatico, in quanto vi è una crescita costante nelle conoscenze base che sono richieste nel campo di lavoro in cui si opera. Un Cyber Range, infatti, permette la simulazione e/o integrazione di reti e protocolli di comunicazione in modo analogo ai contesti reali, fornendo un ambiente guidato e controllato con cui interagire al fine di agevolare lo sviluppo in maniera graduale ed uniforme delle abilità dell'apprendista.

Questo strumento è stato realizzato principalmente per la formazione professionale ed è stato realizzato in primo luogo in ambito militare, di cui un esempio è il *National Cyber Range* sviluppato da DARPA [6], una agenzia governativa del Dipartimento della difesa degli Stati Uniti, incaricata dello sviluppo di nuove tecnologie per uso militare. Dato il potenziale fornito da un Cyber Range, quest'ultimo è stato impiegato anche a livello aziendale nella formazione di personale specializzato per la difesa e la protezione degli asset aziendali, il quale prende il nome di *blue-team*.

L'implementazione dei Cyber Range è nata dal concetto di *digital twin*, ovvero la rappresentazione digitale di un modello, prodotto o processo fisico reale. Questo termine è stato inizialmente coniato dalla NASA per rappresentare il modello virtuale di un velivolo che riproducesse fedelmente il comportamento della controparte fisica (*physical twin*). Il significato è poi mutato nel tempo, estendendosi a qualsiasi ambiente e non più unicamente a quello aerospaziale. La nascita e lo sviluppo dei digital twin è legato quindi alla necessità di valutare il funzionamento e l'interazione degli elementi del mondo reale. Di seguito una definizione proposta da [26]:

> These are digital representations based on semantic data models that allow running simulations in different disciplines, that support not only a prognostic assessment at design stage (static perspective), but also a continuous update of the virtual representation of the object by a real time synchronization with sensed data. This allows the representation to reflect the current status of the system and to perform real-time optimizations, decision making and predictive maintenance according to the sensed conditions.

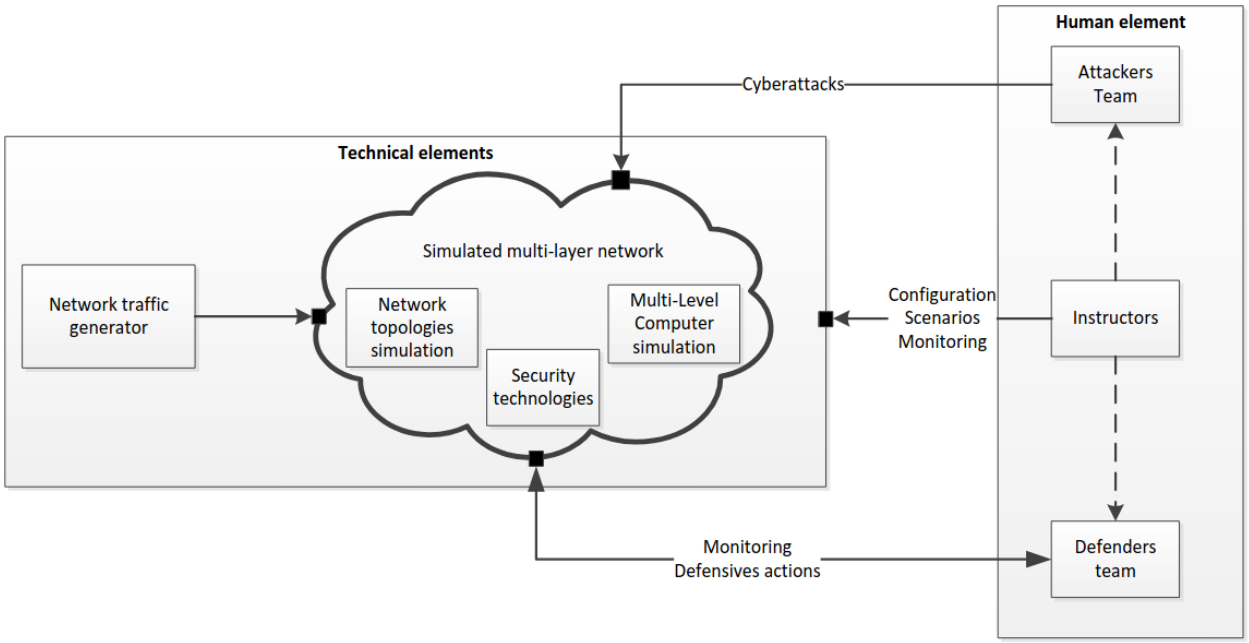


Figura 2.1: Schema di un Cyber Range relativo al documento [22]

Una piattaforma Cyber Range generalmente comprende [14]:

- Macchine e servizi bersaglio
- Strumenti di attacco e di difesa
- Strumenti per la generazione di traffico atto a rendere più realistico l'ambiente di addestramento
- Strumenti di monitoraggio

La scelta delle soluzioni migliori per ognuno di questi punti è chiaramente dipendente dal caso d'uso specifico: esistono infatti numerose proposte commerciali e non, ognuna con i propri vantaggi e svantaggi [22]. La figura 2.1 illustra la struttura generica di un Cyber Range.

In base a come sono implementate le funzioni di un Cyber Range, questo si colloca in una specifica categoria [22]:

1. **Simulazione**: in questo caso ogni componente della rete viene implementata con un modello virtuale che ne riproduce il comportamento in modo non dettagliato, ma solo nelle funzioni essenziali. È la soluzione con il minor costo ma anche con la minor fedeltà.

2. **Ad hoc/Overlay**: vengono installati moduli software sui dispositivi fisici effettivamente utilizzati per aggiungere funzionalità ed effettuare i test.

3. **Emulazione**: la topologia della rete viene fedelmente riprodotta in ogni suo dettaglio nell'infrastuttura del Cyber Range. Il costo sarà elevato ma i risultati dei test saranno quelli più vicini alla realtà.

Se un Cyber Range include oltre ai moduli di simulazione/emulazione anche dispositivi fisici, questo viene definito ibrido.

Come evidenziato nella figura 2.1 una parte fondamentale è data dagli utenti del sistema, i quali si suddividono in tre macro categorie:

- Attaccanti (*red team*): sono professionisti hacker etici, ovvero che operano nel rispetto delle leggi, impiegati dall'organizzazione che adotta il Cyber Range, il cui compito è cercare di sfruttare le vulnerabilità dei sistemi in gioco per mettere alla prova il team di difesa.
- Difensori (*blue team*): si occupano di proteggere gli asset dell'organizzazione dagli attacchi del red team.
- Istruttori (*white team*): sono gli addetti alla creazione degli scenari di addestramento e alla valutazione delle performance degli altri team.

L'impiego di un Cyber Range con obiettivi formativi può quindi essere orientato verso l'attacco (penetration testing, sfruttamento di vulnerabilità), la difesa (progettazione ed attuazione dei meccanismi di sicurezza più adeguati), l'analisi o anche il *disaster recovery*, che prevede la soluzione dei problemi causati da attacchi già portati a termine all'inizio dell'addestramento [13].

Degno di nota è l'impiego di questa categoria di piattaforme in competizioni del tipo *Capture The Flag*, in cui ai partecipanti vengono proposte sfide che prevedono in primis l'identificazione di una o più vulnerabilità informatiche ed in seguito lo sfruttamento di esse, fornendo in questo modo un primo approccio alla sicurezza informatica anche a studenti e principianti (nonostante esistano competizioni per tutti i livelli di esperienza) [2].

2.2 SIEM

Il SIEM, acronimo per ***S**ecurity **I**nformation and **E**vent **M**anagement* [10], è nato dall'unione tra SIM (Security Information) e SEM (Security Event) e permette sia di monitorare ed analizzare in tempo reale gli eventi all'interno di una rete, sia di tener traccia e *loggare* dati inerenti la sicurezza. Il SIEM quindi, aiuta l'organizzazione che lo implementa ad individuare possibili minacce e vulnerabilità prima che queste vengano sfruttate per intaccare dati, operazioni o funzioni vitali dell'azienda. Secondo la definizione ufficiale del NIST[1], il SIEM è:

> An application that provides the ability to gather security data from information system components and presents those data as actionable information via a single interface[19].

Nonostante nel mercato siano presenti proposte differenti per l'implementazione di un SIEM, è possibile identificare alcune funzionalità sempre ricorrenti, indipendentemente dalle componenti del pacchetto software utilizzato [21]:

- **Gestione dei *log***: il SIEM cattura una grande quantità di eventi da un altrettanto ampio numero di sorgenti diverse all'interno della rete aziendale ed, attraverso filtri e regole pre-imposte, permette al processo di collezionamento e catalogazione degli eventi di avvernire in maniera automatica.

[1] National Institute of Standards and Technology

- **Correlazione di eventi e statistiche**: data l'enorme collezione di log raccolta, gestita dalla funzione precedente, il SIEM riesce a utilizzare quest'ultima come base per compiere analisi approfondite sulla correlazione degli eventi. Ovviamente viene posta maggior attenzione su applicazioni e catene di avvenimenti che hanno portato al riscontro di comportamenti imprevisti. Inoltre, sempre rispetto alle regole con cui è stato configurato, il SIEM procede automaticamente in risposta con determinate azioni, come notificare l'analista o intervenire sull'applicazione.

- **Monitoraggio incidenti ed avvisi di sicurezza**: attraverso profonde e complesse analisi di ciò che sta avvenendo e degli eventi passati, il SIEM riesce ad identificare complessi pattern nei log ed efficacemente individuarne la/e fonte/i di un attentato attacco, riducendo esponenzialmente i tempi di individuazione e risposta alla minaccia.

2.3 SDL

Lo ***S**pecification and **D**efinition **L**anguage* è un linguaggio formale orientato agli oggetti sviluppato dall'ITU-T [2] a partire dal 1972. Il linguaggio è stato sviluppato con l'intento di descrivere applicazioni interattive, complesse, *real-time* e che svolgessero molteplici attività concorrentemente.

Lo sviluppo di un SDL è nato dalla crescita esponenziale nella produzione software, che ha portato alla necessità di avere uno strumento che permettesse di gestire qualsiasi tipo di sistema distribuito in maniera omogenea. Questa necessità è stata quindi tradotta nella scrittura di un linguaggio che implementasse metodi formali per far comunicare, e quindi collaborare, strumenti di diversa natura. Un SDL garantisce per l'appunto, la possibilità di descrivere struttura, dati e comportamento di un sistema, in un formato che sia comprensibile in tutte le sue specifiche ed utilizzabile da una qualsiasi applicazione.

Come vedremo nel dettaglio in un capitolo successivo 4, l'SDL da noi utilizzato per la descrizione di tutta la struttura dei Cyber Range con i relativi scenari è stato TOSCA.

2.4 Soluzioni open source di riferimento

2.4.1 CRACK

La prima proposta presa come riferimento è stata CRACK (***C**yber **R**ange **A**utomated **C**onstruction **K**it*), sviluppata dall'Università degli Studi di Genova e disponibile su GitHub [29]. Il progetto "supporta design, deployment, verifica e test automatici di scenari Cyber Range complessi" [30], proponendo una *Scenario Definition Language* basata sullo standard TOSCA, che prende appunto il nome di CRACK SDL, per supportare le varie fasi elencate. Sebbene CRACK sia sulla carta la risposta più adatta ai requisiti rilevati in precedenza, sono state trovate purtroppo non poche difficoltà nel suo utilizzo, prima fra tutte la mancanza di un plugin fondamentale per il suo funzionamento, in quanto basato su *AriaTosca* [12], un SDK utile ad implementare scenari definiti in TOSCA, il cui sviluppo è stato purtroppo interrotto oramai dal 2018, come si evince dalla pagina di presentazione. Da questo la scelta orientarsi verso l'architettura della sezione 3.3.

[2] International Telecommunications Union – Telecommunications Standardization Sector

2.4.2 DigitalTwinSocCyberrange

Questo progetto nasce da una ricerca della *University of Regensburg* e della *Ionian University*, le quali hanno realizzato un prototipo di Cyber Range [31] funzionante e utilizzabile a scopo di addestramento [32]. Attraverso questo sistema è possibile assemblare insieme tutte le componenti fondamentali di un Cyber Range, dagli strumenti di attacco e difesa a quelli di monitoraggio, ma, data la natura prototipale del progetto, esso non è particolarmente adatto all'estensione a nuovi scenari.

DigitalTwinSocCyberrange ha svolto un ruolo chiave nella scelta dei tool utilizzati per lo sviluppo del SIEM all'interno del nostro progetto, ma va altrettanto conferito il merito allo strumento CRACK, per l'ottimo lavoro di integrazione dello standard TOSCA all'interno del processo di deployment, da cui si è di fatto preso spunto.

3

Analisi del problema e progetto della soluzione

3.1 Casi d'uso

Come precendentemente descritto, la realizzazione dei Cyber Range è avvenuta per scopi formativi, con l'obiettivo di creare uno strumento di addestramento che fosse elastico e che permettesse all'utilizzatore di mettere mano ed operare in situazioni che replicano la realtà, facendo acquisire all'apprendista le conoscenze ed abilità rihieste nel lavoro. Per questi motivi, l'utilizzo dei Cyber Range è stato combinato a dei percorsi di formazione su misura, poichè specifi per l'ambiente di lavoro, che permettessero una specializzazione sempre più fine dell'apprendista nell'ambito richiesto. Questa soluzione si è rivelata alquanto efficace, riuscendo ad ottenere, al termine del periodo di insegnamento, personale allenato e pronto ad affrontare casi e situazioni che si presenteranno realmente nell'ambiente lavorativo. Per un corretto funzionamento dei Cyber Range, è stato sviluppato un progetto per la gestione dello strumento che richiede la presenza di due tipi di utenti:

- l'amministratore di sistema: adibito a costruire lo scenario di addestramento.
- l'analista: incaricato di valutare i problemi di sicurezza dell'infrastuttura.

Osservando la figura 3.3, la figura dell'amministratore viene posta all'inizio della pipeline, mentre l'analista opera nella parte terminale di quest'ultima. Normalmente ogni operatore assume unicamente uno tra i due ruoli, tuttavia, per necessità specifiche, ciascuno può andare a ricoprire entrambi; ciò è legato unicamente alla direttive dell'azienda e dalle loro disponibilità di risorse.

Qui di seguito, sono proposti i casi d'uso più frequenti, rappresentati sotto forma di diagramma (fig. 3.1), distinguendo quali riguardano l'SDL e quali il SIEM.

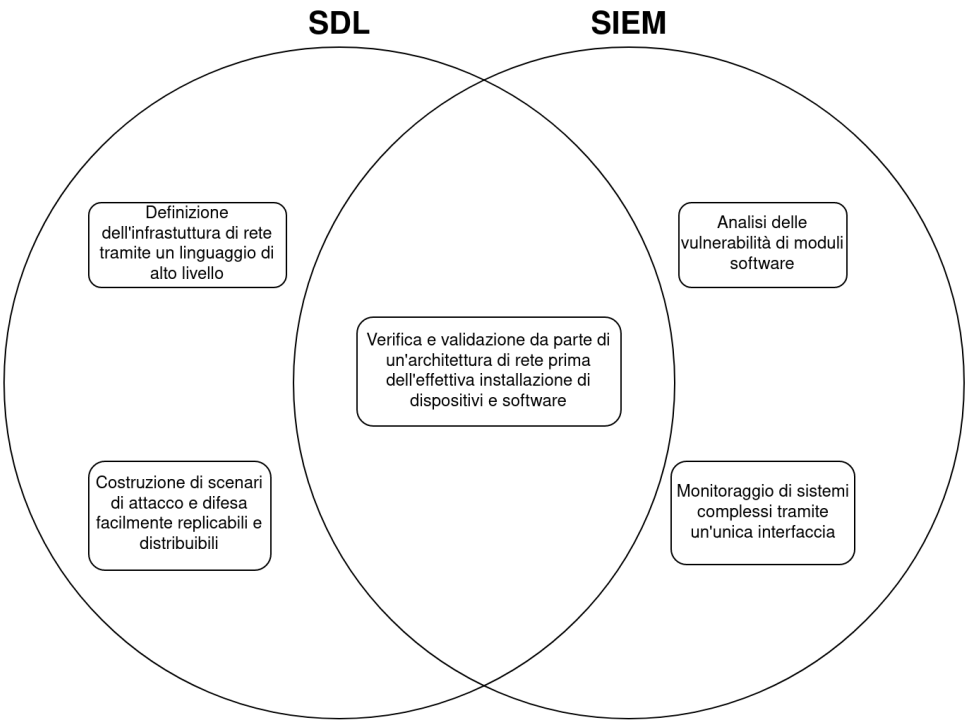


Figura 3.1: Casi d'uso più frequenti

3.2 Requisiti

A partire dai casi d'uso si derivano i requisiti funzionali e non funzionali per entrambe le parti del progetto:

SDL	
Requisito	**Funzionale/Non funzionale**
Creare reti virtuali in cui posizionare gli host	Funzionale
Poter assegnare manualmente gli indirizzi	Funzionale
Definire quali programmi devono essere eseguiti su ogni host	Funzionale
Specificare privilegi e risorse di ogni host	Funzionale
Interazione dinamica con il filesystem degli host	Funzionale
Essere compatibile con standard già presenti	Non funzionale
Estendibilità	Non funzionale
Portabilità	Non funzionale
Indipendenza da implementazioni specifiche	Non funzionale

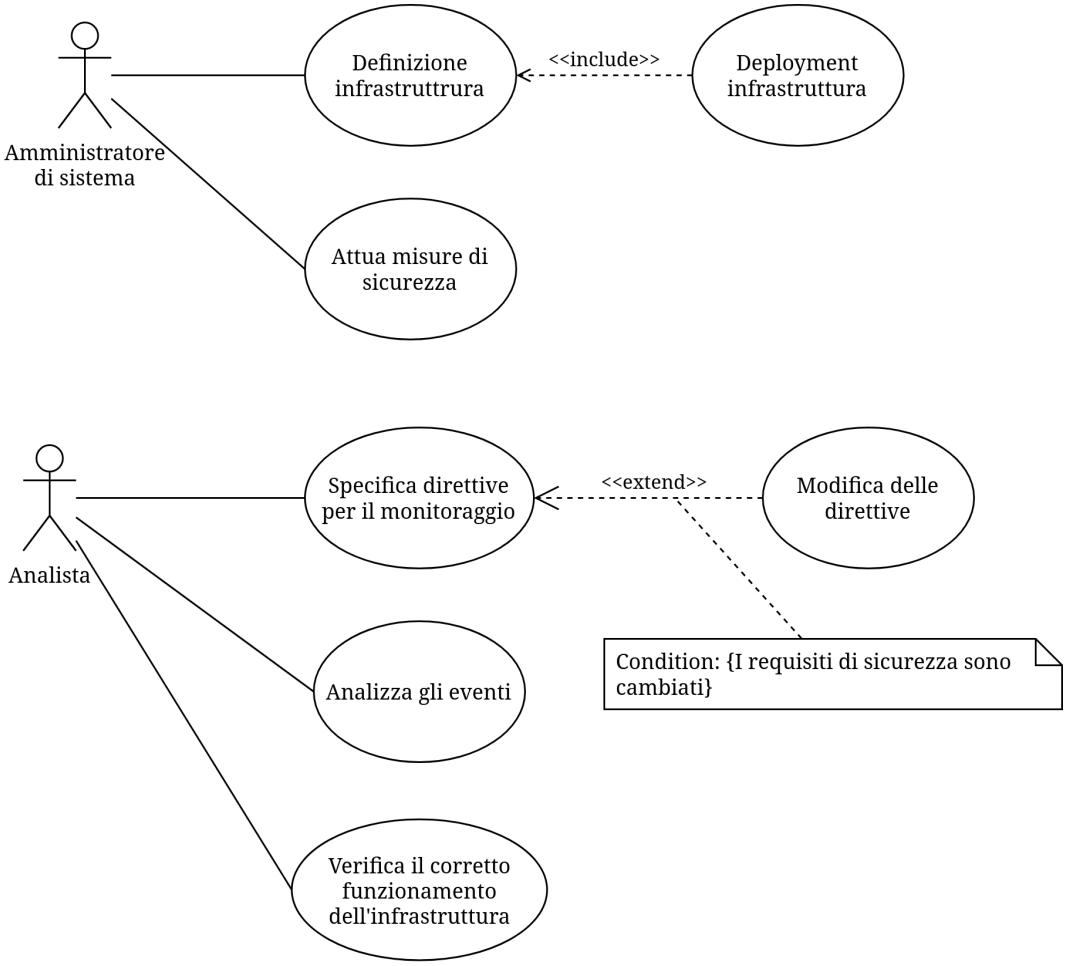


Figura 3.2: Diagramma dei casi d'uso

SIEM	
Requisito	**Funzionale/Non funzionale**
Analisi di rete e log di sistema	Funzionale
Correlazione di eventi	Funzionale
Configurazione dinamica delle direttive di sicurezza in base alle policy	Funzionale
Interfaccia grafica per esplorare i dati	Funzionale
Creazione di dashboard	Funzionale
Interfaccia unificata	Non funzionale
Compatibilità con altri software	Non funzionale
Possiblità di scegliere diversi IDS	Non funzionale

3.3 Architettura

L'architettura del progetto (fig. 3.3) è una pipeline che parte dalla definizione dell'infrastruttura desiderata tramite lo standard TOSCA proposto da OASIS, passando poi per il formato *clout* prodotto dal software Puccini e procedendo infine alla costruzione dello scenario tramite Terraform e Docker.

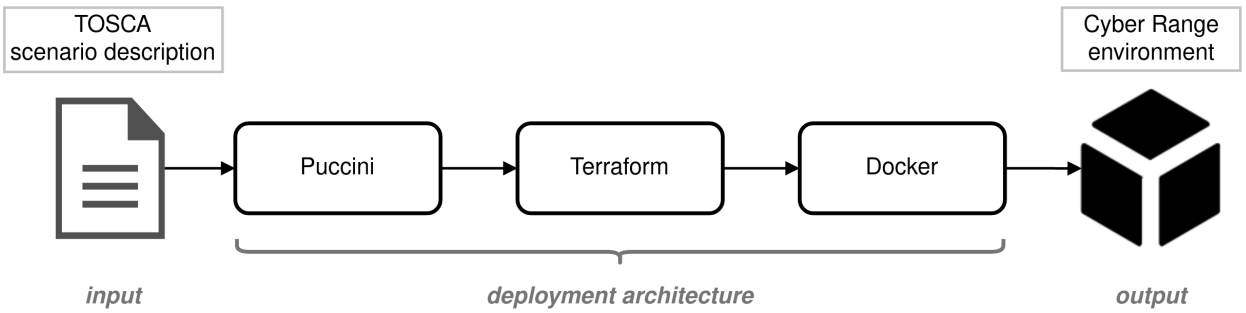


Figura 3.3: L'architettura del progetto

3.3.1 TOSCA

TOSCA è uno standard proposto da OASIS che rende possibile la descrizione di servizi ed applicazioni in vari ambienti, tra cui il cloud; tale descrizione include componenti (dispositivi fisici, macchine virtuali, container, ...), relazioni, dipendenze, requisiti e funzionalità, proponendo un modello portabile, la cui gestione sia indipendente dall'infrastruttura sottostante ed automatizzabile [7]. I modelli costruiti con TOSCA sono intrinsecamente disaccoppiati da implementazioni specifiche, per questo motivo ogni *vendor* propone la propria [27].

3.3.2 Puccini

Puccini è un parser per il linguaggio TOSCA che produce testo nel formato *clout*, ovvero una rappresentazione intermedia che si pone come ponte fra il codice TOSCA e le implementazioni specifiche. Per design, Puccini non è basato su un particolare *orchestrator* ed i file clout devono essere interpretati da opportuni programmi per essere utilizzati: questo aumenta il disaccoppiamento delle componenti della pipeline, facendo sì che la scelta dell'orchestrator sia lasciata liberamente agli sviluppatori.[23, 24]

3.3.3 Terraform

Terraform è un software che sposa la metodologia *Infrastructure as Code*: viene messo a disposizione un linguaggio descrittivo per definire un'infrastruttura, la quale viene poi effettivamente costruita tramite *provider* che sfuttano le API di pubblico utilizzo [17]. Fra i provider forniti si può trovare Kubernetes, AWS, Docker, Azure, ma anche molti altri, infatti attualmente ne esistono oltre 2000 [15]). L'approccio scelto da Terraform è quello della *Immutable Infrastructure*, che consiste nella completa ricostruzione del sistema quando una modifica è necessaria: il principale vantaggio è di poter effettuare il deployment finale delle nuove versioni dei servizi solo quando questi sono effettivamente funzionanti, senza incorrere in problemi derivanti da aggiornamenti parziali o in errori che portano a stati non previsti. Una soluzione di questo tipo richiede l'esternalizzazione dei dati in modo che, ad ogni nuova iterazione dell'infrastruttura, questi non vengano persi [16].

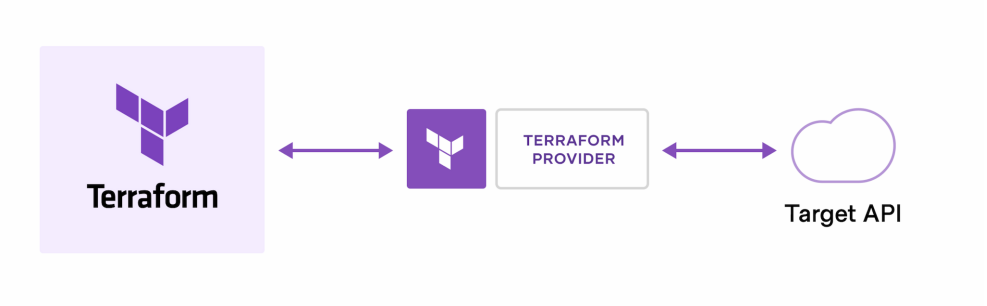


Figura 3.4: Architettura di Terraform

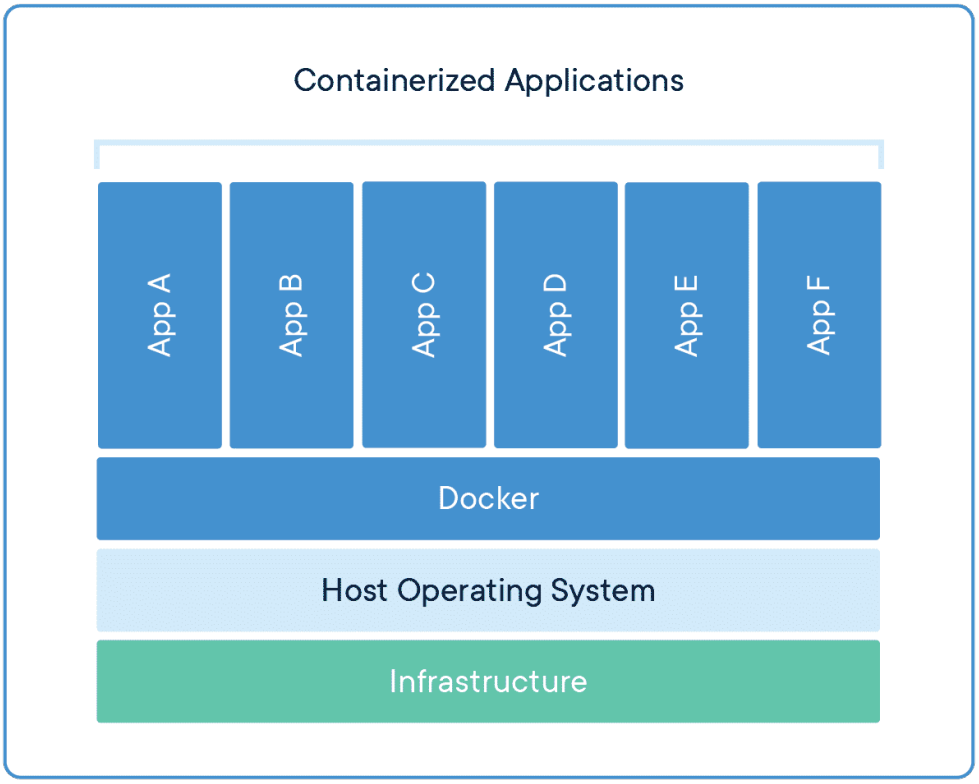


Figura 3.5: Struttura dei container Docker

3.3.4 Docker

Docker è lo standard *de facto* per la creazione di container: pacchetti software che sfruttano la virtualizzazione a livello del kernel (fig. 3.5 ne è una rappresentazione schematica). Ogni container viene visto come uno *user space* differente dal sistema operativo: ciò significa che le applicazioni interne ad un container siano isolate dalle altre, potendo avere differenti configurazioni ed impostazioni di sistema ad-hoc. A differenza di una macchina virtuale, un contaier di Docker ha un minor impatto sulle prestazioni e risulta più portabile.

3.3.5 Scenario

Con "scenario" si intende l'insieme di tutti i nodi che vanno a comporre il Cyber Range, dalle singole macchine al sistema di monitoraggio. Le macchine utilizzate possono essere fisiche o virtuali, ma appariranno allo stesso modo al progettista che scrive il codice TOSCA.

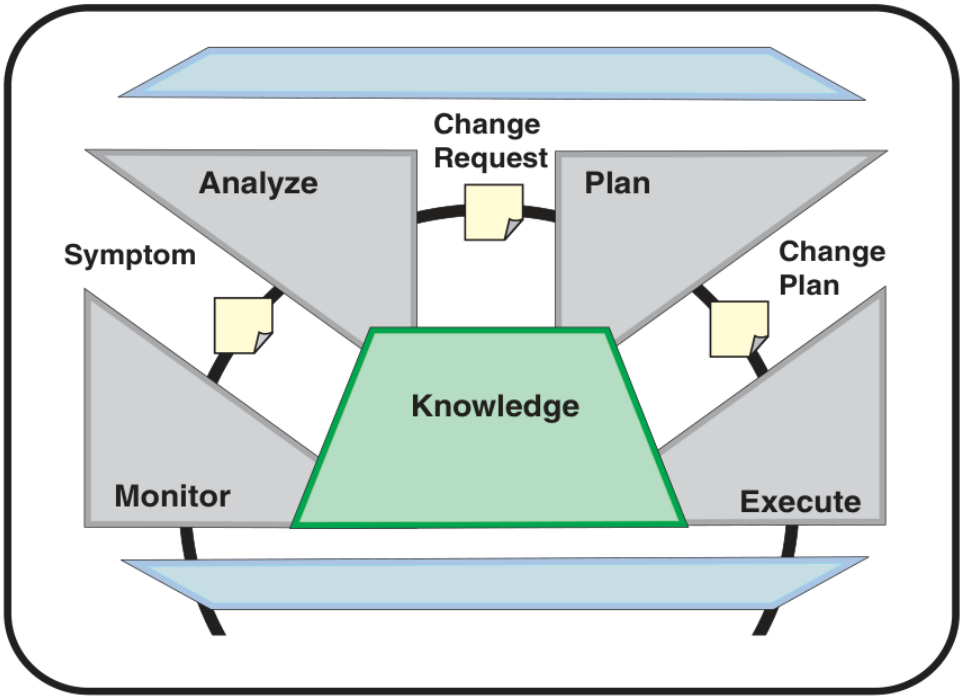


Figura 3.6: Diagramma MAPE-K [18]

3.4 L'architettura MAPE-K

La struttura del sistema SIEM trova una correlazione molto stretta con l'architettura MAPE-K proposta per la prima volta da IBM nel contesto dei sistemi autonomici [18]. MAPE-K è l'acronimo di *Monitor, Analyze, Plan, Execute, Knowledge*, che, secondo la proposta di IBM, sono le funzioni fondamentali che un *autonomic manager* deve possedere.

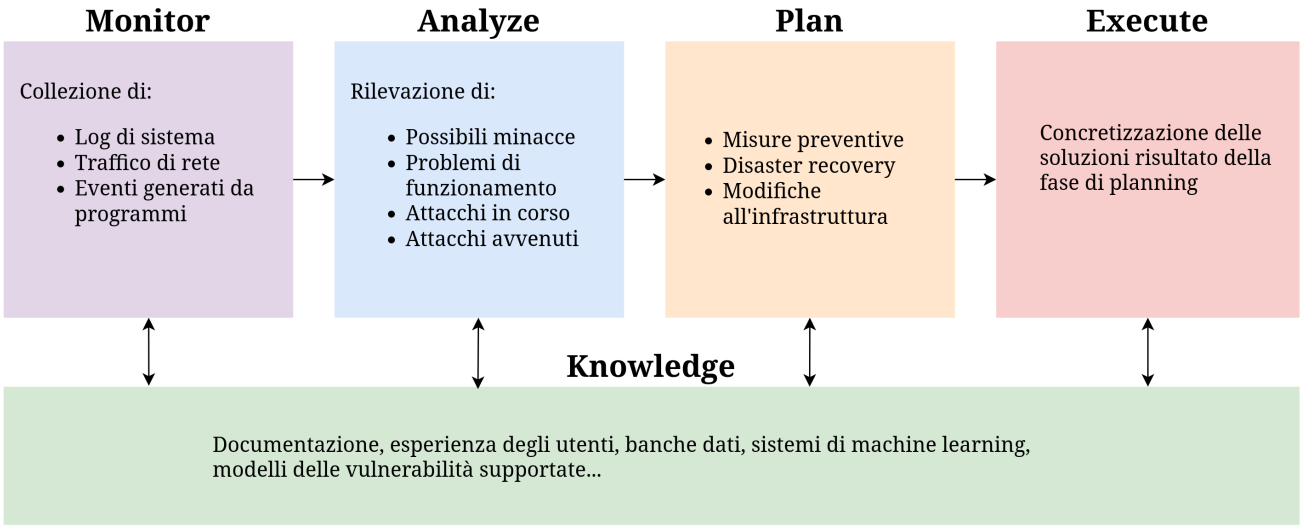
Il sistema è organizzato ciclicamente in quanto ogni parte di esso svolge la propria precisa funzione utilizzando come input, l'output della precedente. In particolar, ciascun blocco ha i determinati compiti ora descritti:

- **Monitor**: fase di collezione di dati e dettagli di funzionamento delle risorse come dispositivi fisici o traffico di rete, con conseguente normalizzazione e aggregazione di essi per permetterne l'analisi futura.

- **Analyze**: la cui finalità consiste nel determinare se risulta necessario apportare dei cambiamenti quando alcune policy non sono rispettate. Questa fase è fondamentale per comprendere se il sistema sia in grado di svolgere i propri compiti nel modo corretto, non solo in quel determinato momento, ma anche in quelli futuri.

- **Plan**: necessario a pianificare le azioni da eseguire: può comprendere sia semplici comandi che sequenze complesse di operazioni. È comunque fondamentale che alla fine di questa fase, i cambiamenti da effettuare alle risorse siano formalizzati in modo da poter essere eseguiti.

- **Execute**: blocco dove vengono attuate le direttive prodotte durante l'immediatamente precedente pianificazione, utilizzando le interfacce offerte delle risorse. Questa fase può anche portare nuo-



ve conoscenze (*Knowledge*) riguardo al funzionamento del modello e quindi influenzare le altre componenti.

3.4.1 Concretizzare MAPE-K

Sebbene ciò che viene proposto in questa trattazione non possa essere considerato un sistema autonomico "puro", in quanto è sempre necessario l'intervento umano, i tool scelti supportano gli operatori in tutte le fasi descritte. In modo analogo alla sezione precedente, si vede ora nella seguente tabella come ogni parte dell'architettura svolga una funzione MAPE:

Monitor	Filebeat, Logstash, Elasticsearch
Analyze	Suricata, Dsiem, Kibana
Plan	Utenti
Execute	Terraform, Docker

4

Implementazione

All'interno del capitolo vengono descritti lo standard TOSCA ed il parser Puccini, con una breve accenno alla loro struttura e funzionamento generico, per poi concentrarsi sulle scelte implementative prese per la descrizione degli scenari e di come e in che grado gli strumenti appena citati siano stati utilizzati durante il processo di sviluppo dei vari Cyber Range.

4.1 TOSCA

4.1.1 Introduzione

TOSCA è l'acronimo di ***T****opology and* ***O****rchestation* ***S****pecification for* ***C****loud* ***A****pplications* [8] ed è uno standard, proposto da OASIS e basato sul linguaggio YAML [1], pensato e realizzato per descrivere in maniera unica, qualsiasi tipo di applicazione a supporto di un'infrastruttura cloud. Infatti, attraverso TOSCA è possibile gestire tutti gli aspetti di un servizio cloud, ovvero operazioni, relazioni e dipendenze, durante tutto il ciclo di vita, indipendentemente dalle tecnologie sottostanti e dal fornitore del servizio. Una qualsiasi azienda che adotta lo standard TOSCA è in grado di modellare le proprie reti e configurazioni senza preoccuparsi di problemi legati alla portabilità o all'interoperabilità, migliorando così il processo di gestione ed automazione della rete. TOSCA inoltre è un linguaggio orientato agli oggetti fortemente tipato, ciò significa che sono presenti controlli di tipo molto rigidi; questi sono però la chiave del successo per riuscire a coniugare due proprietà altrimenti incompatibili:

- affidabilità
- estendibilità

4.1.2 Struttura

Overview

TOSCA, nonostante si basi su YAML e presenti quindi una struttura molto semplice ed una sintassi di facile lettura e comprensione, può risultare inizialmente alquanto complesso da apprendere. Ciò è dovuto alla libertà fornita da questo standard, il quale non pone limiti se non quelli sulla correttezza dei tipi,

[1] YAML Ain't Markup Language

nei quali nulla è lasciato al dubbio. Il livello di granularità che è possibile esprimere attraverso TOSCA dipende unicamente dalla volontà dell'utilizzatore o dalle necessità poste dal caso d'uso. Allo stesso tempo è infatti possibile passare da descrizioni molto generiche ma valide per molteplici applicazioni, ad altre molto dettagliate e ricche ma nettamente più complesse. Pertanto, per agevolare i primi approcci durante l'apprendimento dello standard TOSCA, è stata realizzata una documentazione ufficiale [28] da parte del team di OASIS, con l'obiettivo di abbattere nettamente i tempi di apprendimento di quest'ultimo.

Tipi

Come sopra spiegato, TOSCA è fortemente tipato, pertanto è necessario comprendere a fondo le differenze esistenti tra i tipi base che questo standard implementa, i quali sono 6 e sono così suddivisi:

- **dati**: descrivono i possibili tipi di dato esistenti ed utilizzabili dalle componenti.
- **nodi**: servono a descrivere le singole entità, o nodi appunto, di un'infrastruttura o topologia.
- **interfacce**: vanno a definire le operazioni per un nodo durante il suo ciclo di vita.
- **relazioni**: servono a definire le relazioni esistenti tra le componenti dichiarate all'interno della topologia. Le relazioni sono direzionate poichè uniscono i requisiti di un nodo alle capacità di un altro.
- **gruppi**: utili per l'aggregazione di più componenti, incrementano il livello di organizzazione, strutturazione e gestione delle dipendenze nella topologia.
- **polize**: sono dei vincoli che una componente deve necessariamente possedere per esistere ed essere utilizzata.

Tra le tipologie appena elencate, i nodi sono sicuramente quella più importante, in quanto sono l'unità base da cui partire per descrivere una qualsiasi topologia, comprese le più semplici. Ogni nodo viene definito tramite i seguenti campi:

- **proprietà**: funzionalità di tipo statico, ovvero il cui valore è assegnato durante il processo di descrizione e che sono quindi immutabili durante l'utilizzo della risorsa.
- **attributi**: sono simili alle proprietà, ma vengono utilizzare per salvare valori che sono instaziati al momento dell'inizializzazione del nodo. Possono essere visti idealmente come una proprietà, ma dinamica.
- **capacità**: funzionalità che il nodo offre, utilizzate dalle relazioni ed abbinate a requisiti di altre componenti.
- **requisiti**: al contrario delle capacità, questi sono le funzionalità di cui il nodo necessità e vengono soddisfatti attraverso l'utilizzo delle relazioni.

Come è possibile osservare nel seguente frammento di codice TOSCA e nel corrispettivo diagramma (fig. 4.1), viene creata una relazione all'interno del nodo `server_port` in cui vengono referenziati i nodi `server_1` e `network_1`.

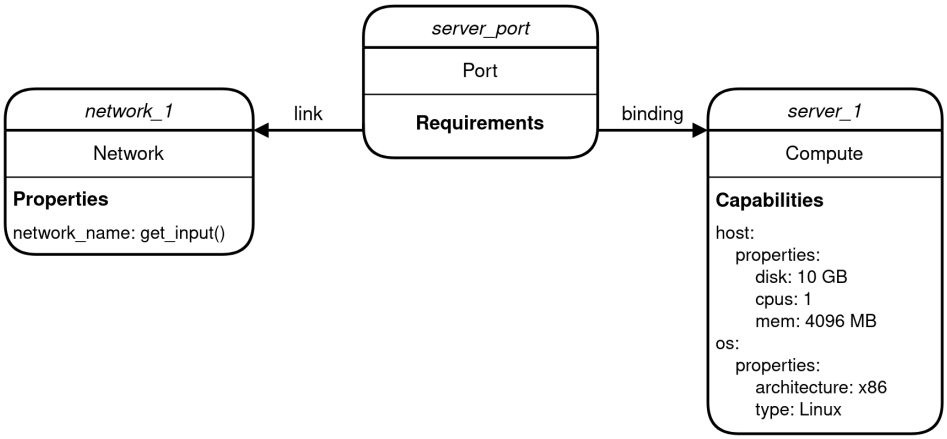


Figura 4.1: Diagramma del frammento di codice 4.1.2 sulle relazioni di una rete

```
server_1:
  type: tosca.nodes.Compute
  capabilities:
    host:
      properties:
        disk_size: 10 GB
        num_cpus: 1
        mem_size: 4096 MB
    os:
      properties:
        architecture: x86_64
        type: Linux

network_1:
  type: tosca.nodes.network.Network
  properties:
    network_name: { get_input: network_name }

server_port:
  type: tosca.nodes.network.Port
  requirements:
    - binding:
      node: server_1
    - link:
      node: network_1
```

Template

Un template è una descrizione di un'applicazione cloud ottenuta dalla composizione degli elementi precedentemente descritti all'interno del file di specifica, quali, nodi, relazioni o gruppi. È possibile istanziare ciascun elemento con tipi primitivi già inclusi e definiti in TOSCA, di cui il più base è *Root*, oppure si può procedere alla "creazione" dei propri tipi personali; questi non sono effettivamente creati, ma bensì sono l'estensione di tipi primitivi a cui sono stati aggiunti campi di propria definizione. Attraverso i template è possibile definire delle ereditarietà che permetteranno in futuro allo sviluppatore il suo riutilizzo in altri progetti, ovviamente con le opportune correzioni sui tipi utilizzati, ove necessario. Come si può notare, questo processo diventa fondamentale nella produzione di scenari di Cyber Range, poichè rende il deployment molto più veloce ed aggiunge la possibilità che questo venga automatizzato.

Nell'esempio di codice seguente sono stati descritti vari template di nodi, tra cui `siem_network` e `web_server`, i quali possono essere riutilizzati a proprio piacimento per la descrizione di nuovi scenari, apportando solo leggere modifiche, tra cui i tipi utilizzati. Il file contenente i tipi personali creati è "`my_types.yaml`" e nel caso in cui si volesse cambiare i tipi utilizzati nel template sarà necessario solamente sostituire il PATH con un'altro che punti al file con le dichiarazioni che si vogliono utilizzare. Al contrario, se lo scenario è molto differente, ma i tipi precedentemente utilizzati sono comunque validi per la topologia, basta cambiare il template usato.

```
imports:
- file: ./my_types.yaml
  namespace_prefix: mt

topology_template:

  node_templates:

    siem_network:
      type: mt:my_net
      properties:
        name : siemnet
        driver: bridge
      attributes:
        subnet: 192.168.80.0/20
        gateway: 192.168.80.1

    shared_network:
      type: mt:my_net
      properties:
        name : sharednet
        driver: bridge
      attributes:
        subnet: 172.30.0.0/16
        gateway: 172.30.0.1
```

```
    external_network:
      type: mt:my_net
      properties:
        name : extnet
        driver: bridge

    web_server:
      type: mt:my_host
      properties:
        image: simple-web-server
        name: webserver
        networks_advanced:
          - name: siemnet
            ipv4_address: 192.168.80.10
        hostname: webserver
        ports:
          internal: 80
          external: 80
        capabilities:
            add: [NET_ADMIN]
```

Parte del contenuto del file `my_types.yaml` è invece raccolto nel seguente frammento di codice:

```
node_types:
  my_net:
    description: network node type
    derived_from: tosca.nodes.Root
    properties:
      name:
        type: string
        required: true
      driver:
        type: string
        default: bridge
        required: false
      ipv6:
        type: boolean
        default: false
        required: false
    attributes:
      subnet:
        type: string
      gateway:
        type: string
```

```
my_host:
  description: host node type
  derived_from: tosca.nodes.Root
  properties:
    name:
      type: string
      required: true
    hostname:
      type: string
      required: false
    image:
      description: name of the image used by the host
      type: string
      required: false
    user:
      description: user that can access the host
      type: string
      required: false
    networks_advanced:
      description: network configuration of the host
      type: list
      entry_schema:
        type: container.datatypes.network
      required: false
    ports:
      description: port configuration of the host
      type: container.datatypes.port
      required: false
    capabilities:
      type: container.datatypes.capabilities
      required: false
```

I frammenti di codice riportati in questa sezione, fanno tutti riferimento ad alcuni file utilizzati nel progetto per lo sviluppo degli scenari che verranno poi analizzati nel capitolo successivo di test e validazione 5.

4.2 Puccini

Come descritto brevemente nella sezione 3.3.2, Puccini è un parser per lo standard TOSCA, o più nello specifico è un *Cloud Topology management and deployment tool based on TOSCA* [24]. Puccini è nato principalmente come processatore di TOSCA e per questo motivo non svolge anche il ruolo di orchestrator. Vengono citate alcune parole dell'autore, scritte per la descrizione di Puccini:

> Note that Puccini is intentionally not an orchestrator. This is a "BYOO" kind of establishment ("Bring Your Own Orchestrator").

L'autore Tal Liron ha voluto realizzare un prodotto realmente utile e che si differenziasse dalle offerte presenti sul mercato, senza reinventare o far concorrenza a strumenti già diffusi, supportati e stabili, dei quali si menziona *Turandot*. La seconda motivazione dell'autore per lo sviluppo di Puccini, è stata la volontà di mantenersi fedele alla linea di pensiero degli sviluppatori dello standard TOSCA: fornire un prodotto funzionale e compatibile indipendentemente dall'infrastruttura sottostante.

Puccini è stato sviluppato attraverso il linguaggio Go [5], ottendendo così un parser in grado di gestire template di servizi molto grandi e complessi, ottimizzato per la velocità attraverso la concorrenza ed il *caching*. Nel tentativo di mantenere questo tool conforme alle specifiche di TOSCA, il processo di *parsing* è stato suddiviso in 5 fasi [25]:

1. **Lettura**: serve a validare la sintassi del file TOSCA preso in input ed è composto a sua volta dai seguenti sottoprocessi:

   - controllo dei dati presenti in file e/o url importati.
   - conversione del file input da YAML a ARD [2], in modo che i dati all'interno diventino leggibili da ogni linguaggio di programmazione.
   - ri-convertimento dei dati da ARD in strutture dati TOSCA, per la normalizzazione di tutte le notazioni nella forma completa ed un controllo sui campi per la rilevazione di possibili *field* obbligatori lasciati vuoti.
   - importazione ricorsiva e concorrente di tutti i file controllati al primo punto.

2. **Namespace**: questo passaggio serve a gestire i nomi presenti nel file TOSCA, i quali vengono controllati per assicurarsi che non siano ambigui o vi possano essere possibili errori nelle associazioni. Inoltre vengono sostituiti i prefissi dei file importati con i *namespace* completi.

3. **Gerarchie**: ogni entità all'interno di un file TOSCA possiede il proprio tipo (primitivo o creato), pertanto, essendo che TOSCA è fortemente tipato, viene effettuato un controllo sulla gerarchia per evitare possibili loop nelle ereditarietà o la presenza di qualche genitore con la dichiarazione incompleta.

4. **Ereditarietà**: è il compito più complesso e di difficile gestione, in quanto essendo tutto il file basato sui tipi, non è immediato risolvere le dipendenze derivanti dall'ereditarietà, poichè molto spesso sono molteplici e legate tra di loro. La soluzione implementata dallo sviluppatore è stata quella di utilizzare una struttura dati a grafo, in cui vengono risolte prima le entità prive di dipendenze, per poi proseguire con quelle le cui dipendenze sono già state risolte precedentemente, fino al risolvimento di tutte le entità.

5. **Rappresentazione**: in quest'ultima fase vengono infine istanziati i valori di default dei campi vuoti e viene verificato che ogni field abbia un numero corretto di occorrenze nel range specificato.

Al termine di tutti e 5 gli step di verifica e validazione, il file risulterà completo e si potrà esser certi dell'assenza di errori di qualsiasi natura.

[2] Agnostic Raw Data [1]

4.2.1 Puccini-Clout

L'output del processo di parsing eseguito da Puccini sarà un file in formato *clout*, ovvero un linguaggio rappresentativo di una ***clou**d **t**opology*. Un file clout non è altro che un database contenente dati grezzi (ARD), rappresentato attraverso una struttura dati a grafo. L'estensione di un file clout è di default YAML, tuttavia può essere convertito in altri formati quali JSON o XML. Il clout serve da intermediario tra un file descrittore iniziale, scritto in TOSCA, ed il corrispettivo file finale, pronto per il deployment attraverso lo strumento che si desidera (nel caso di questo progetto Terraform). Il vantaggio del clout è il distacco totale da qualsiasi tipo di tecnologia utilizzata, in quanto contiene unicamente dei nodi nei quali sono salvate le specifiche per il deployment e i vertici sono collegati tra loro da archi rappresentanti le relazioni tra le varie entità. Una gestione così grezza dei dati ha però un costo, ovvero l'impossibilità di essere leggibile, o perlomeno di essere compresa, da una persona umana e per questo motivo il compito di interpretare correttamente il contenuto di un file clout è lasciato al calcolatore.

A titolo rappresentativo viene fornito qui di seguito un piccolo frammento di un file clout generato da Puccini per uno scenario realizzato per il progetto:

```
vertexes:
  _2DwIeEe3e9MP5nTqI6mAHMboZaJ:
    metadata:
      puccini:
        kind: NodeTemplate
        version: "1.0"
    capabilities:
        feature:
          attributes: {}
          description: ""
          location:
            column: 5
            path:
            ↪  topology_template.node_templates["filebeat-es-data"].capabilities["feature"]
            row: 152
          maxRelationshipCount: -1
          minRelationshipCount: 0
          properties: {}
          types:
            tosca::Node:
              description: The Node capability indicates the base capabilities of a TOSCA
              ↪  Node Type.
              metadata:
                specification.citation: '[TOSCA-Simple-Profile-YAML-v1.3]'
                specification.location: 5.5.2
                tosca.canonical-name: tosca::Node
                tosca.normative: "true"
              parent: tosca::Root
            tosca::Root:
```

```
properties:
  artifacts: {}
  attributes:
    state:
      $information:
        definition:
          description: The state of the node instance.
        type:
          name: string
      $value: initial
. . .
```

La nostra scelta implementativa per la gestione dei file clout generati per il progetto è stata la realizzazione di *scriplet* ad-hoc in JavaScript che, attraverso un'apposita conversione, permettessero l'utilizzo immediato del file da parte di Terraform. Attraverso una funzione integrata di Puccini, che prende il nome di Puccini-Clout, è stato possibile accodare automaticamente l'esecuzione dello scriplet direttamente nel processo di parsing, riuscendo così, con l'utilizzo di un unico strumento, a passare da un file SDL in TOSCA ad un file tf pronto per l'utilizzo in Terraform.

5

Test e validazione

Nella figura 5.1 è illustrato lo scenario utilizzato per verificare il corretto funzionamento di tutti i tool, la cui architettura è simile a quella presentata in [30], rappresentante una rete aziendale fittizia.

Il diagramma comprende innanzitutto quattro reti:

- `Servernet`: l'intranet aziendale, i cui host non devono essere accessibili dall'esterno
- `DMZ`: in cui sono contenuti i servizi esposti pubblicamente
- `Extnet`: la rete in cui risiedono i client e quindi chi conduce gli attacchi
- `IoT`: necessaria alla connessione di sensori e plc, il cui comportamento è simulato via software

Il traffico fra le varie reti viene filtrato ed analizzato dall'host su cui è stato installato Suricata, che blocca l'accesso a `Servernet` agli host di `Extnet`. I servizi offerti in questo caso sono:

- Una applicazione web che comunica con il database in `Servernet`, vulnerabile ad *SQL injection* e *HTTP Host header poisoning*
- Una python jail vulnerabile a *RCE* [20]
- Un DNS usato per raggiungere gli altri host tramite nome simbolico

Il client nella rete esterna utilizza il DNS per contattare i servizi della DMZ, mentre i dati raccolti dai sensori nella rete `IoT` sono collezionati da un'istanza di Filebeat e convogliati verso il database Elasticsearch nella rete interna: l'interazione fra Filebeat ed i sensori avviene via volume di Docker, grazie al quale i due container riescono a condividere alcuni file (in questo caso i log dei sensori).

5.1 Test e validazione SDL

Lo scenario è stato completamente definito in TOSCA, in cui sono stati aggiunti i seguenti tipi specifici per la rappresentazione delle risorse necessarie: `terraform_network`, `terraform_volume`, `terraform_image`, `terraform_host` e `terraform_volume`.

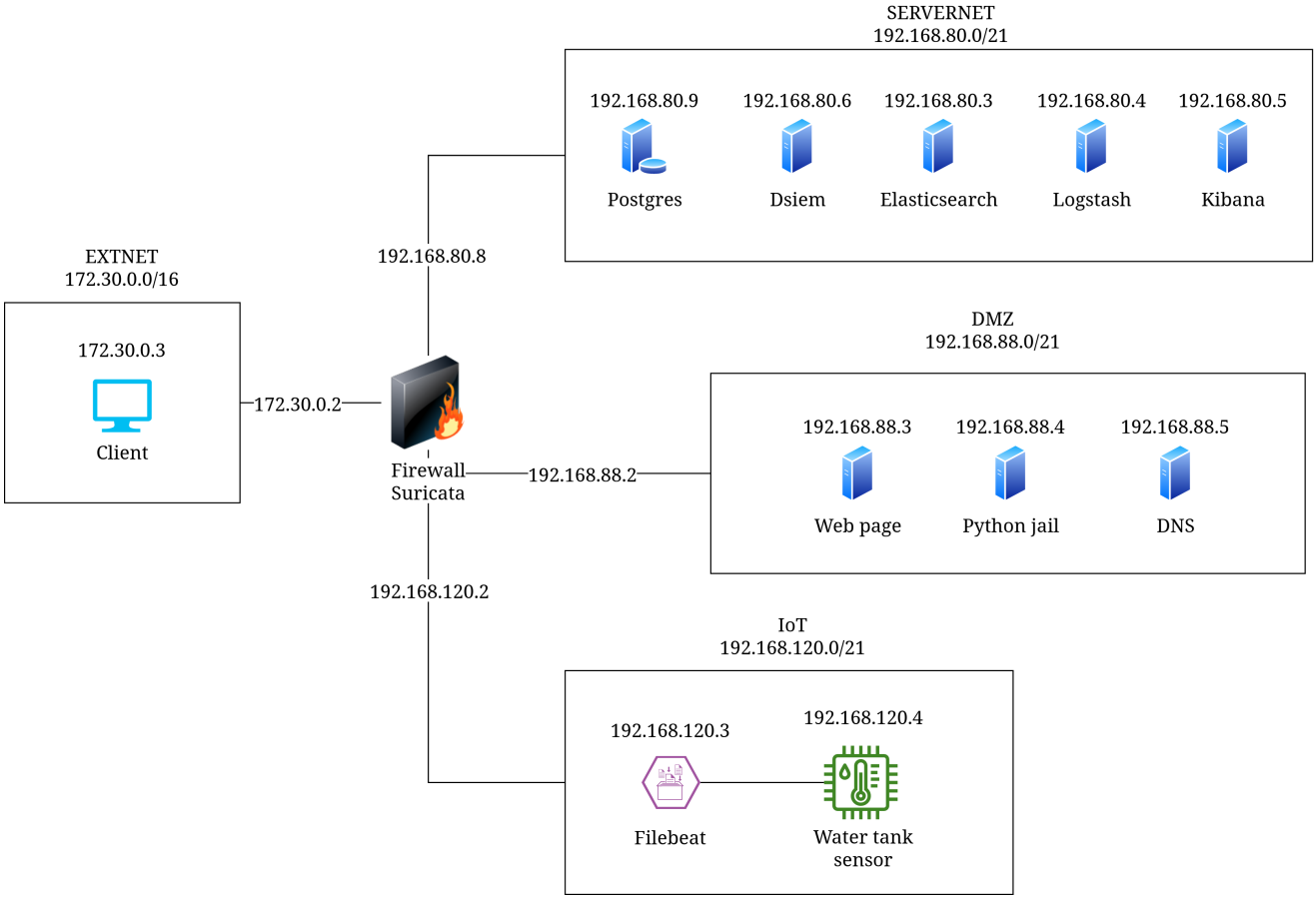


Figura 5.1: Scenario di esempio

```
# IMAGES
suricata_image:
    type: tf:terraform_image
    properties:
        name: gianzav/suricata-filebeat:7.4.0
        keep_locally: true
...

# NETWORKS
server_network:
  type: tf:terraform_network
  properties:
    name : servernet
    driver: bridge
  attributes:
    subnet: 192.168.80.0/21
    gateway: 192.168.80.1
...
```

```
# HOSTS
suricata:
  type: tf:terraform_host
  properties:
    image: suricata_image
    name: suricata
    networks_advanced:
      - name: servernet
        ipv4_address: 192.168.80.8
      - name: dmznet
        ipv4_address: 192.168.88.2
      - name: extnet
        ipv4_address: 172.30.0.2
    hostname: suricata
    capabilities:
      add:  ["NET_ADMIN", "SYS_ADMIN"]
    privileged: true
    volumes:
      - volume_name: "suricata-log"
        container_path: "/var/log/suricata"
      - container_path: "/var/lib/suricata/rules/"
        host_path: "/home/gz/repos/Tesi/deployments/crack-1/terraform-main/suricata_rules/"
    command: ["/bin/bash", "-c", "service filebeat start && service suricata start && sleep
    ↪  1h"]

...

# VOLUMES
suricata-log:
  type: tf:terraform_volume
  properties:
    name: suricata-log
...
```

Il file TOSCA viene poi interpretato da Puccini, il quale produrrà un file clout utilizzato come input da uno *scriplet* in JavaScript, che a sua volta fornirà in output il file descrittore finale. Questo file sarà utilizzabile direttamente da Terraform per creare i container Docker che saranno gli host. In questo modo i requisiti funzionali dell'SDL sono soddisfatti.

```
# Comandi per la costruzione dello scenario
> puccini-tosca compile --exec terraform.generate --output=terraform.tf input.yaml
> terraform init
> terraform validate
> terraform fmt
> terraform apply -auto-approve
```

Action	Corr. Stage	Started	Ended	Status	Name	From	To	Protocol	Port From	Port To	Events
👁	1	a minute ago	a minute ago	finished	Http Host header poisoning	ANY	ANY	ANY	ANY	ANY	1/1
👁	2	a minute ago	a minute ago	finished	Http Host header poisoning	172.30.0.3	192.168.88.3	ANY	ANY	ANY	5/5
👁	3	a minute ago	-	active	Http Host header poisoning	172.30.0.3	192.168.88.3	ANY	ANY	ANY	3/10

Event Detail

Event ID	Timestamp	Title	Source	Destination	Protocol	Port From	Port To	Sensor	Plugin	Plugin SID
10216627-c054-41ac-b863-ec355503caf6	a minute ago	HTTP Host header poisoning to vulnerable web server	172.30.0.3	192.168.88.3	TCP	41140	80	suricata	1001	1000005

Figura 5.2: Attacco rilevato in DSIEM

5.1.1 Validazione

Avendo utilizzato TOSCA, è sempre possibile definire nuovi tipi ed estendere il linguaggio in base alle necessità e fare in modo che l'interpretazione dello stesso non debba produrre per forza un file terraform che sfrutti Docker: mantenendo il disaccoppiamento fra specifica dello scenario e implementazione, non è difficile riadattare il linguaggio in modo da supportare tecnologie diverse. Sono soddisfatti quindi i requisiti non funzionali di compatiblità ed estendibilità. Infine la portabilità è assicurata dall'utilizzo di Docker, perchè il comportamento dei container sarà sempre lo stesso a prescindere dalla macchina su cui viene eseguito.

5.2 Test e validazione SIEM

Il test del SIEM ha previsto di effettuare degli attacchi con l'host client verso le risorse della DMZ e verificare che questi vengano effettivamente rilevati.

5.2.1 Host Header Poisoning

L'applicazione web nella DMZ è basata su PHP e, al caricamento della pagina da parte del browser, viene eseguito un file JavaScript caricato in maniera dinamica:

```
<script type="text/javascript" src="http://<?php echo \$_SERVER['HTTP_HOST'];
↪  ?>/script.js"></script>
```

Il valore della variabile di PHP `$_SERVER['HTTP_HOST']` dipende dall'header "Host" presente nella richiesta HTTP fatta al server: un attaccante può cambiare tale header e far eseguire uno script diverso da quello previsto. L'attacco viene effettuato con *Burp suite*, un programma che fra le varie funzioni permette di intercettare e modificare le richieste e risposte HTTP.

5.2.2 Remote Code Execution

Nello scenario di test è presente una *challenge* proposta per la prima volta nella *uiuctf* del 2022, una competizione in stile *Capture The Flag*. L'obiettivo dell'attaccante è riuscire ad eseguire del codice arbitrario per estrapolare informazioni (nello specifico la *flag*) superando le restrizioni imposte dall'ambiente in cui il codice viene eseguito. Una python jail è infatti un ambiente di esecuzione le cui funzioni vengono limitate in modo da impedirne il misuso.

```
#!/usr/bin/python3
import re
import random
horse = input("Begin your journey: ")
if re.match(r"[a-zA-Z]{4}", horse):
    print("It has begun raining, so you return home.")
elif len(set(re.findall(r"[\W]", horse))) > 4:
    print(set(re.findall(r"[\W]", horse)))
    print("A single horse cannot bear the weight of all those special characters. You
    ↪  return home.")
else:
    discovery = list(eval(compile(horse, "<horse>", "eval").replace(co_names=())))
    random.shuffle(discovery)
    print("You make it through the journey, but are severely dehydrated. This is all you
    ↪  can remember:", discovery)
```

In questo caso il codice viene privato dell'ambiente globale a tempo di esecuzione, impedendo di fatto l'utilizzo di tutte le funzioni built-in di python. Inoltre sono ammessi solo 4 caratteri speciali (lo stesso carattere speciale ripetuto conta come se fosse stato usato una volta sola): la vulnerabilità risiede nel fatto che parole riservate del linguaggio rimangono disponibili (p. es. `lambda`) fornendo qualche grado di libertà all'attaccante. Un esempio di codice per l'attacco è:

```
from pwn import *

# context.log_level = 'debug'
r = remote('pyjail', 1337)

p1 = b'(lambda:eval(input()))()'
p2 = b'print(open("/flag.txt").read())'

r.sendlineafter(b'Begin your journey: ', p1)
r.sendline(p2)
print(r.recvline().decode())
```

Tramite una funzione lambda il codice che verrà eseguito dall'ambiente remoto sarà quello ricevuto con la funzione `input()` e a questo punto l'attaccante riuscirà ad estrapolare l'informazione desiderata.

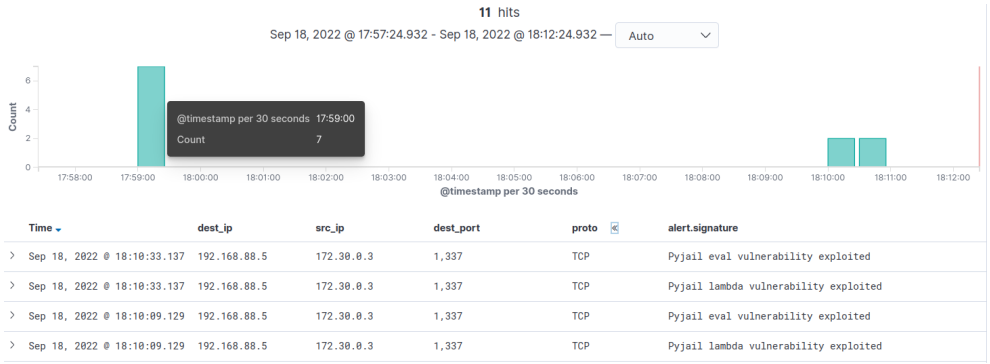


Figura 5.3: Rilevazione Remote Code Execution

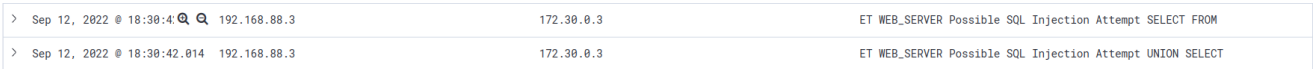


Figura 5.4: L'allarme di Suricata in Kibana in seguito a SQL Injection

5.2.3 SQL injection

Visto che la web app di cui si è già parlato permette anche di fare ricerche in un catalogo prodotti fittizio, un attaccante potrebbe fornire nel campo di ricerca una stringa malevola che, se mal gestita nel backend, può portare ad un'SQL injection con conseguente estrapolazione di dati riservati. Nel database è presente una tabella *product*, normalmente acceduta dalla web app, e una tabella *account* in cui sono contenuti i dati di accesso degli amministratori della pagina. L'input del campo di ricerca non viene filtrato, dunque si possono ottenere username e password degli utenti inserendo in input un'apposita stringa.

```
# payload per ottenere gli username
' AND 1=0 UNION SELECT NULL,username,NULL from account; --

# payload per ottenere le password
' AND 1=0 UNION SELECT NULL,password,NULL from account; --
```

Nella figura 5.4 si vede come Suricata riesca a rilevare l'utilizzo improprio dell'applicazione e come venga segnalato l'evento all'interno dell'interfaccia di Kibana (la richiesta HTTP viene fatta dal container in `Extnet`).

Riprendendo i requisiti esposti nel capitolo 4 vediamo come l'analisi di rete e di sistema può essere fatta grazie a Filebeat e Suricata attraverso Elasticsearch, esplorando i dati in un'interfaccia grafica unificata offerta da Kibana. Si è quindi verificato il funzionamento ed il rispetto dei requisiti del Cyber Range attraverso la simulazione di attacchi reali, sia dal punto di vista network, costruito seguendo le specifiche TOSCA, che nella rilevazione degli attacchi, svolta dal SIEM.

6

Conclusioni

6.1 Considerazioni finali

L'obiettivo del progetto descritto nel capitolo 1, nello specifico la realizzazione di una pipeline per lo sviluppo ed il deployment di un Cyber Range, si può considerare pienamente raggiunto. Come approfondito nel capitolo 4 e verificato nel successivo 5, l'implementazione della topologia del Cyber Range attraverso le descrizioni in TOSCA e l'analisi ed il controllo dell'intero sistema per mezzo del SIEM, si sono mostrate efficaci e stabili, indipendentemente dallo scenario presentato. Inoltre, la pipeline si è dimostrata all'altezza di un impiego in ambito educativo, ovvero per l'addestramento di personale o la formazione di studenti relativamente al mondo della sicurezza informatica.

6.2 Limitazioni e possibili miglioramenti

Alla condizione attuale, il progetto risulta comunque agli inizi e non effettivamente pronto per un impiego a livello aziendale, in cui la richiesta di affibilità della pipeline deve essere assoluta, per evitare anche la remota possibilità di danni gravi a plc, componenti fisiche o addirittura al personale. Senza alcun dubbio sono necessari ulteriori test, tra cui alcuni comprendenti l'utilizzo di dispositivi fisici esterni nel Cyber Range.

In secondo luogo l'obiettivo di automatizzazione del processo richiede ulteriori raffinamenti e l'estensione della compatibilità di alcuni componenti della pipeline, in quanto sono stato riscontrati i seguenti difetti:

- lo scriplet finora permette unicamente l'utilizzo di Docker come provider.
- le immagini Docker utilizzate dai container e chiamate all'interno del file TOSCA richiedono di essere già costruite prima dell'utilizzo.
- il processo di *error detection* risulta molto lento, in quanto anche il minimo errore nella scrittura dei file TOSCA, nelle immagini Docker o in un'altra componente è difficilmente individuabile e si deve procedere con un'analisi della pipeline ad ogni step per riuscire ad identificarlo.

Qui di seguito vengono pertanto proposti alcuni tra i possibili miglioramenti necessari a risolvere i problemi appena evidenziati:

- l'estensione della compatibilità dello scriplet ad altri provider.
- l'estensione dell'utilizzo di TOSCA, per permettere la descrizione di entità e di topologie maggiormente complesse.
- l'aggiunta di processi intermedi di validazione e controllo per ogni fase della pipeline, per un'individuazione degli errori precisa e veloce.
- lo sviluppo di un'interfaccia grafica *user-friendly* per la gestione della pipeline ed in particolar modo per la semplificazione del processo di scrittura dei file TOSCA, per avere un immediato responso visivo della topologia, fondamentale nella descrizione di scenari complessi.

Bibliografia

[1] Agnostic raw data (ard) library github repository. https://github.com/tliron/kutil/tree/main/ard.

[2] Ctfd : What is capture the flag? https://ctfd.io/whats-a-ctf/.

[3] Definizione cybercrime treccani. https://www.treccani.it/enciclopedia/cybercrime_%28Lessico-del-XXI-Secolo%29/.

[4] Difese informatiche più efficaci per una pubblica amministrazione più sicura. https://italiadomani.gov.it/it/Interventi/investimenti/cybersecurity-sicurezza-informatica.html.

[5] Golang language official site. https://tip.golang.org/.

[6] The national cyber range - whitehouse.gov.

[7] OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC | FAQ. https://www.oasis-open.org/committees/tosca/faq.php.

[8] OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC | OASIS. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca.

[9] Rapporto Clusit ottobre 2022. https://clusit.it/rapporto-clusit/.

[10] What is a siem? https://www.ibm.com/topics/siem.

[11] ACN. Il programma cybersecurity. https://www.acn.gov.it/pnrr.

[12] Apache. Apache incubator - ariatosca. https://incubator.apache.org/projects/ariatosca.html.

[13] Razvan Beuran, Ken ichi Chinen, Yasuo Tan, e Yoichi Shinoda. *Title Towards Effective Cybersecurity Education and Training.* 2018.

[14] Giovanni Cerrato. Cyber range cos'è. https://www.giovannicerrato.it/cyber-range/, Aug 2021.

[15] HashiCorp. Terraform providers. https://registry.terraform.io/browse/providers.

[16] HashiCorp. What is mutable vs. immutable infrastructure? https://www.hashicorp.com/resources/what-is-mutable-vs-immutable-infrastructure.

[17] HashiCorp. What is terraform? https://www.terraform.io/intro.

[18] IBM. An architectural blueprint for autonomic computing. https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf, Jun 2005.

[19] Arnold Johnson, Kelley L. Dempsey, Ronald S. Ross, Sarbari Gupta, e Dennis Bailey. Guide for security-focused configuration management of information systems. p. 73, 2011.

[20] kmh. Uiuctf. https://2022.uiuc.tf/challenges#A%20Horse%20with%20No%20Names-214.

[21] Igor Kotenko, Olga Polubelova, e Igor Saenko. The ontological approach for siem data repository implementation. In *2012 IEEE International Conference on Green Computing and Communications*, pp. 761–766, 2012.

[22] Marc Lamotte. *D2.1: State of the art cyber range technologies analysis*, p. 45. giugno 2020.

[23] Tal Liron. Clout - github. https://github.com/tliron/puccini/tree/main/clout.

[24] Tal Liron. Puccini - github. https://github.com/tliron/puccini.

[25] Tal Liron. Puccini tosca parser. https://puccini.cloud/tosca/parser/.

[26] Elisa Negri, Luca Fumagalli, e Marco Macchi. A review of the roles of digital twin in cps-based production systems. *Procedia Manufacturing*, 11:939–948, 2017. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy.

[27] OASIS. Known tosca implementations. https://github.com/oasis-open/tosca-community-contributions/wiki/Known-TOSCA-Implementations.

[28] OASIS. *TOSCA Simple Profile in YAML Version 1.3*.

[29] Enrico Russo. Cyber range automated construction kit (crack) - github. https://github.com/enricorusso/crack.

[30] Enrico Russo, Gabriele Costa, e Alessandro Armando. Building next generation cyber ranges with crack. *Computers & Security*, Volume 95, 04 2020.

[31] Ionian University University of Regensburg. Digitaltwinsoccyberrange github repository. https://web.archive.org/web/20220309014930/https://github.com/DigitalTwinSocCyberrange/DigitalTwinCyberrange.

[32] Manfred Vielberth, Magdalena Glas, Marietheres Dietz, Stylianos Karagiannis, Emmanouil Magkos, e Günther Pernul. A digital twin-based cyber range for soc analysts In *Data and Applications Security and Privacy XXXV*. A cura di Ken Barker e Kambiz Ghazinour, pp. 293–311, Cham, 2021. Springer International Publishing.