UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in
Ingegneria Informatica e dell'Automazione

Data Analysis di Security Logs ed implementazione di notarizzazione log su una permissioned blockchain per il dispositivo LECS

Security Logs Data Analysis and implementation of permissioned blockchain log notarization for LECS device

Relatore:
Prof. Spalazzi Luca

Laureando:
Massimo Ciaffoni

Anno Accademico 2021-2022

Indice

Capitolo 1

Introduzione

Negli ultimi anni la sicurezza informatica è divenuta una priorità per le aziende e per le pubbliche amministrazioni. La capacità di difendere le proprie infrastrutture critiche ed informatiche da attacchi, insieme alla protezione dei dati sensibili, sono divenuti un investimento necessario per le aziende. Secondo uno studio [1] in Italia il numero degli attacchi informatici nel 2020 è aumentato così come la loro complessità ed efficacia. I dati più recenti inoltre mostrano una crescita degli attacchi informatici gravi. Dunque non solo la complessità degli attacchi ma anche la loro "Severity", ovvero la valutazione degli impatti generati dagli stessi è aumentata significativamente. Nelle analisi dei rischi e degli attacchi è possibile definire 4 livelli di impatto: Basso, Medio, Alto e Critico.
Gabriele Faggioli [2],responsabile scientifico dell'osservatorio Cybersecurity & Data Protection del Politecnico di Milano afferma che:

"*Nel 2021 gli attacchi di livello Critico sono stati ben il 32%, mentre quelli di livello Alto si sono verificati quasi nella metà dei casi (47%). Quelli con impatto Medio rappresentano invece il 19% del totale. Dunque il numero di attacchi di livello Critico e Alto abbiano sfiorato insieme l'80% del totale degli attacchi hacker.*"

Per far fronte alle diverse minacce esterne esistono diversi strumenti tecnologici che le aziende utilizzano per gestire la sicurezza delle proprie infrastrutture:

- **Identity and Access Management:** Strumenti per la gestione dell'autenticazione, dell'autorizzazione ed il monitoraggio delle identità digitali.
- **Vulnerability/Risk Management:** Valutazioni dei rischio e delle vulnerabilità intrinseche delle proprie infrastrutture.
- **Intrusion Prevetion ed Intrusion Detection:** Strumenti e tecniche di monitoraggio e machine learning in grado di identificare e classificare minacce esterne.
- **Incident Response:** Tecniche di pianificazione per il contenimento delle minacce esterne e successivo ripristino delle infrastrutture attaccate.

In particolare negli ultimi anni tra i molteplici strumenti tecnologici utilizzati dalle aziende i *Security Information and Event Management (SIEM)* sono diventati una delle tecnologie più utilizzate [3]. Tali strumenti uniscono la gestione delle informazioni (SIM) e degli eventi di sicurezza (SEM) in un unico sistema. Il task principale è la raccolta, l'aggregazione e la successiva analisi delle informazioni provenienti da molteplici fonti. Le informazioni raccolte discendono da diverse sorgenti a partire dalle informazioni dei segmenti di rete, delle applicazioni e dei vari dispositivi presenti all'interno dell'infrastruttura. I vari dati raccolti sono principalmente *log* i quali poi sono successivamente aggregati, trasformati ed integrati in un unico formato in modo da poter essere successivamente analizzati. Infatti una volta trasformati i log, il SIEM utilizza delle regole predeterminate basate su correlazione degli eventi o basate su tecniche di Machine Learning per rilevare eventuali minacce all'interno dell'infrastruttura, prendere le corrette contromisure e notificare l'utente. Oltre al monitoraggio e alla risposta agli incidenti, i SIEM permettono di ottenere in tempo reale visualizzazioni centralizzate utili ai manager per la gestione della sicurezza e dell'intelligence delle infrastrutture dell'organizzazione.

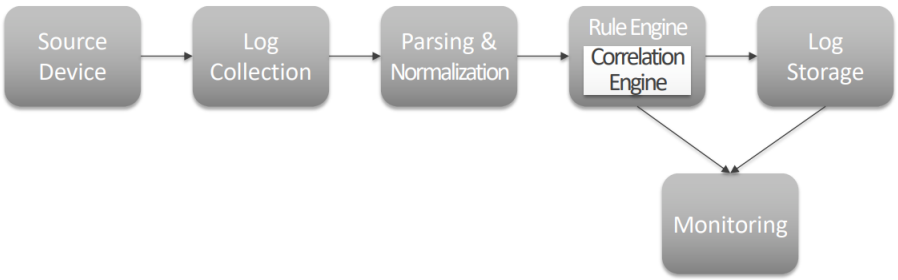

Figura 1.1: Architettura di un SIEM

1.1 LECS

Last Eletrical Cyber Shield (LECS) è un dispositivo innovativo di sicurezza informatica, il quale permette di proteggere la propria rete locale, le infrastrutture Cloud e gli impianti industriali dagli attacchi e dalle minacce informatiche più pericolose tramite l'uso di un sistema brevettato di contromisura [4]. Il dispositivo presenta diverse caratteristiche:

- **Plug & Play**: LECS non necessita di alcuna configurazione per entrare in funzione, è sufficiente collegare il dispositivo in un punto strategico della rete.
- **Compilance**: Grazie ai report generati periodicamente, LECS è un dispositivo in grado di essere conforme al GDPR.

- **Sistemi Industriali**: LECS supporta tutti i protocolli delle macchine 4.0 e mette in sicurezza i dati come richiesto dai certificati di qualità.
- **Sentinella Passiva**: LECS agisce come una trappola strategica di rete integrabile anche con altri dispositivi di sicurezza già esistenti in quanto agisce come una sentinella e da trappola nascosta all'interno della rete. Confuso con gli host, il dispositivo analizza le minacce in tempo reale dall'interno della rete classificandole in base alla gravità di impatto. Monitorando il traffico in maniera passiva LECS assicura anche la massima privacy difendendo l'infrastruttura di rete da minacce interne ed esterne.
- **Impact Isolation Response**: In caso di minaccia ad alto impatto interviene il motore di scollegamento a doppia natura secondo procedura brevettata ed inviando in maniera immediata notifica di alert all'utente. In seguito alla contromisura, il dispositivo ripristina il collegamento di rete, verificando la presenza di ulteriori minacce imparando e parametrizzando in cloud i dati della minaccia.
- **Log Storage**: LECS funge da log store a lungo termine conservando tutte le statistiche di monitoraggio. Tutti i log, sia di rete che relativo agli attacchi rilevati dal dispositivo sono comodamente visibili dall'apposita Web App che tiene traccia di ciò che succede nella rete.

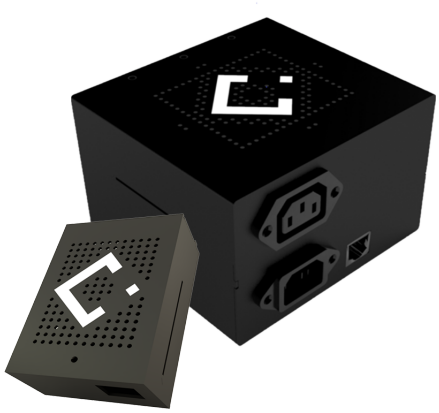

Figura 1.2: Dispositivo LECS

1.2 Obiettivi e struttura della tesi

L'obiettivo della tesi è quello di effettuare dei task di data analysis e notification relativi ai log di sicurezza generati da LECS e notarizzare i log riguardanti gli attacchi più critici all'interno della blockchain in modo tale che queste transazioni risultino immutabili. La tesi è strutturata in diversi capitoli descritti nel seguente elenco:

- Il capitolo 2 descrive a livello teorico le tecniche e le architetture utilizzate durante la tesi.

- Il capitolo 3 descrive le tecnologie utilizzate per lo sviluppo dei vari task.
- Il capitolo 4 descrive brevemente il funzionamento dell'intera infrastruttura dei dispositivi LECS e come l'architettura del sistema di notifica e notarizzazione che si sono realizzati sono stati integrati in questa infrastruttura.
- Il capitolo 5 descrive i task di data analysis eseguiti durante lo svolgimento della tesi.
- Il capitolo 6 descrive i passaggi eseguiti per la creazione del sistema di notarizzazione all'interno di una permissioned blockchain.
- Il capitolo 7 descrive le diverse considerazioni effettuate per quanto riguarda la sicurezza del sistema di notarizzazione.

Capitolo 2

Tecniche e architetture utilizzate

2.1 Anomaly Detection

L'anomaly detection è un task che permette di individuare pattern non idonei rispetto al comportamento normale dei dati. Esistono principalmente tre diverse classi di anomalie:

1. **Anomalie Puntuali:** Se un dato assume dei valori anomali rispetto al resto della popolazione in maniera puntuale.
2. **Anomalie Contestuali:** Se un dato è anomalo rispetto ad un determinato contesto che può essere temporale o spaziale. Dunque un dato il quale non risulta anomalo rispetto ad altri, ma anomalo rispetto al contesto in cui viene misurato. Per fare un esempio un dato relativo alle precipitazioni in millimetri con un valore elevato può essere un' anomalia contestuale se registrato durante la primavera.
3. **Anomalie Correlate:** Se più dati correlati tra loro risultano anomali, dunque i singoli dati non sono anomali, ma il loro verificarsi insieme rappresenta una sequenza anomala.

Le tecniche di anomaly detection si concentrano principalmente nel rilevamento di anomalie puntuali e possono essere di diverse tipologie:

- **Tecniche basate sulla classificazione**: Queste tecniche prevedono l'addestramento di un modello predittivo in grado di classificare nuovi dati come anomalie. Lo svantaggio principale di queste tecniche sono la necessità di avere a disposizione dati etichettati del comportamento normale ed anomalo dei dati per la fase di training e test del modello, i quali non sono sempre disponibili.
- **Tecniche basate sul Nearest Neighboar**: Queste tecniche si basano sull'assunzione che le istanze di dati normali si verificano in vicinati densi, mentre le anomalie si verificano lontano dai loro vicini più prossimi. Le tecniche di rilevamento delle anomalie basate sul nearest neighbor richiedono una distanza o una misura di somiglianza definita tra due istanze di dati la quale può essere diversa nel caso in cui si stiano analizzando

attributi continui o categorici. Lo svantaggio di tale tecnica è che richiede un complessità che è $\mathtt{O}(N^2)$.

- **Tecniche basate sul Clustering:** Sono tecniche di tipo unsupervised le quali non richiedono la disponibilità dei dati etichettati. La complessità computazionale del training di una tecnica di rilevamento delle anomalie basata sul clustering dipende dall'algoritmo di clustering utilizzato per generare i cluster dai dati. Così tali tecniche possono avere una complessità quadratica se la tecnica di clustering richiede il calcolo delle distanze a coppie per tutte le istanze dei dati, o lineare quando si usano tecniche euristiche come k-means o tecniche di clustering approssimativo. La fase di test delle tecniche basate sul clustering è veloce, poiché comporta il confronto di un'istanza di test con un piccolo numero di cluster.

- **Tecniche basate sulla teoria dell'informazione:** Queste tecniche analizzano il contenuto informativo di un insieme di dati usando diverse misure derivate dalla teoria della'informazione come la complessità di Kolomogorov o l'entropia. Tali tecniche assumono che le anomalie nei dati inducono irregolarità nel contenuto informativo dell'intero insieme di dati.

- **Tecniche basate sulla statistica:** Le tecniche di rilevazione statistica delle anomalie si basano sulla seguente assunzione in cui le istanze di dati normali si verificano nelle regioni ad alta probabilità di un modello stocastico, mentre le anomalie si verificano nelle regioni a bassa probabilità del modello stocastico. Le tecniche statistiche adattano un modello statistico (di solito per il comportamento normale) ai dati forniti e poi applicano un test di inferenza statistica per determinare se un'istanza non esaminata in precedenza appartiene a questo modello o no. Le istanze che hanno una bassa probabilità di essere generate dal modello appreso, sulla base della statistica di test applicata, sono dichiarate come anomalie. Lo svantaggio chiave delle tecniche statistiche è che si basano sull'assunzione che i dati siano generati da una particolare distribuzione. Questa assunzione spesso non è vera, specialmente per insiemi di dati reali ad alta dimensione.

- **Tecniche spettrali:** Le tecniche spettrali cercano di trovare un'approssimazione dei dati usando una combinazione di attributi che catturano la maggior parte della variabilità nei dati. Tali tecniche si basano sulla sull'assunzione secondo la quale i dati possono essere inseriti in un sottospazio dimensionale inferiore in cui le istanze normali e le anomalie appaiono significativamente diverse. Quindi l'approccio generale adottato dalle tecniche di rilevamento delle anomalie spettrali è quello di determinare tali sottospazi in cui le istanze anomale possono essere facilmente identificate. Diverse tecniche utilizzano la Principal Component Analysis (PCA) per proiettare i dati in uno spazio di dimensioni inferiori.

Di seguito sono descritte due tecniche utilizzate durante il progetto.

2.1.1 L'Entropia nella teoria dell'informazione

La teoria dell'informazione è il campo dell'informatica che si occupa dello studio e dell'analisi matematica dei fenomeni relativi alla misurazione, memorizzazione e trasmissione delle informazioni [5]. Tra le metriche più importati utilizzate per misurare la quantità di informazione c'è l'entropia. Nella teoria dell'informazione, l'entropia di una variabile casuale è il livello di "incertezza" o "sorpresa" inerente ai possibili esiti della variabile. Secondo la teoria dell'informazione, un messaggio porta tanta più informazione quanto più questo è inaspettato, dunque maggiore è la probabilità di un messaggio, più è bassa l'informazione che esso contiene. Il concetto di entropia dell'informazione è stato introdotto da Claude Shannon nel suo articolo del 1948 "A Mathematical Theory of Communication" [6]. Shannon nel suo articolo afferma che un sistema di comunicazione è composto da tre elementi: una sorgente di dati, un canale di comunicazione e un ricevitore. Secondo Shannon, il ricevitore è in grado di identificare quali dati sono stati generati dalla fonte, in base al segnale che riceve attraverso il canale. Shannon considerò vari modi per codificare, comprimere e trasmettere messaggi da una fonte di dati e dimostrò che l'entropia rappresenta un limite matematico assoluto alla capacità di comprimere senza perdite i dati della sorgente su un canale perfettamente privo di rumore. Supponiamo un messaggio sia composto da un campione di una variabile aleatoria discreta `X` a n valori con probabilità p(x), il livello medio di "inaspettatezza" del messaggio sarà pari a:

$$\sum_{i=1}^{n} p(x_i) \cdot \log \frac{1}{p(x_i)} \tag{2.1}$$

Definiamo l'entropia di una variabile aleatoria discreta `X` come:

$$H[X] = -\sum_{i=0}^{n} p(x_i) \cdot \log_b p(x_i) \tag{2.2}$$

La misura di entropia dunque può essere utilizzata in diversi contesti a partire dalla teoria della probabilità fino alla sicurezza informatica. Durante la tesi la metrica di entropia è stata utilizzata per un duplice scopo: sia per calcolare la variabilità dei pacchetti di un segmento di rete monitorato dal dispositivo LECS, sia per rilevare eventuali anomalie. Infatti dato un flusso di pacchetti l'entropia può essere utilizzata per valutare la stabilità della rete, in quanto supponendo di calcolare il loro flusso medio ad intervalli regolari, se tale flusso risulta costante il valore di entropia calcolato sarà pari a 0. Nel caso in cui invece le diverse rilevazioni del flusso medio si discostino tra loro il valore di entropia aumenta fino ad un valore massimo pari a $log(k)$ dove k rappresenta il numero di rilevazioni considerate. Per quanto riguarda le anomalie l'entropia è stata utilizzata per calcolare eventuali errori nei pacchetti, in questo caso un valore diverso da 0 indica un discostamento dei pacchetti rispetto ad un comportamento normale in assenza di errori e dunque una potenziale anomalia di rete.

2.1.2 K-Means Clustering

Il K-Means è una tecnica di clustering di tipo partizionativo. L'algoritmo va a minimizzare la varianza intra-gruppo dei vari cluster, i quali sono identificati da un punto medio chiamato centroide. Una volta creati in maniera casuale K partizioni ognuna con un suo punto di riferimento, l'algoritmo segue una procedura iterativa:

1. Ogni elemento del dataset è associato alla partizione più vicina formando K cluster.

2. Il centroide di ogni cluster viene calcolato e diventa il nuovo punto di riferimento per i passi successivi.

La procedura viene iterata finché la posizione dei centroidi non converge.

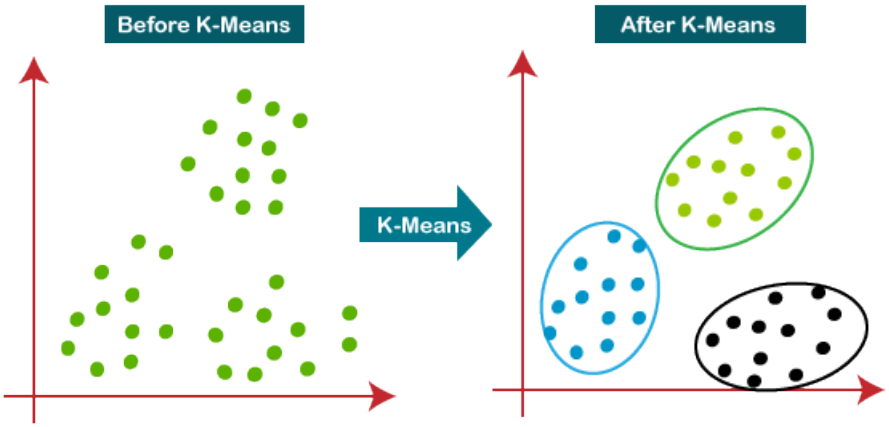

Figura 2.1: Esempio di K-Means Clustering

Tale algoritmo può essere utilizzato per l'anomaly detection nel caso in cui le caratteristiche puntuali di un dato normale ed uno anomalo siano molto differenti tra loro e dunque appartenenti a due cluster differenti. La difficoltà in questo caso è quella di definire il numero K di cluster che il modello dovrà generare, in quanto non è detto che tutti i comportamenti normali siano rappresentati da un unico cluster.

2.2 Blockchain

Nel 2009 Satoshi Nakamoto presenta per la prima volta il Bitcoin, una nuovo sistema per la gestione decentralizzata delle transazioni. Nakamoto afferma che tramite l'uso di reti distribuite è possibile definire un nuovo sistema monetario con una gestione di emissione non più centralizzato da una banca centrale, ma bensì distribuito. Dà lì a pochi anni tale sistema è stato fondamentale nella definizione delle criptovalute. L'idea alla base delle criptovalute è la gestione di emissione della moneta e autorizzazione delle transazioni in maniera decentralizzata. A differenza dei classici sistemi monetari la validità di una transazione

o di un contratto, infatti non è più centralizzata tra i membri della transazione o da un' entità centrale autorevole, ma tale validità è gestita in maniera decentralizzata (figura 2.2) e sicura grazie all'utilizzo di reti peer-to-peer e di primitive crittografiche. Tali sistemi infatti fanno uso di un registro pubblico (ledger) in cui tutte le transazioni sono registrate in modo sequenziale. Tale registro, sequenziale e immutabile, è detto «blockchain». In generale la blockchain può essere definita come un insieme di tecnologie per la gestione decentralizzata delle transazioni. Queste nuove tecnologie rappresentano uno strumento di storage alternativo ai classici database ed al cloud per la supervisione e gestione delle transazioni. La blockchain appartiene alla più ampia famiglia di tecnologie a Distributed Ledger (DLT) le quali si basano su sistemi di registro distribuito. Un DLT è un tipo di database condiviso, replicato e sincronizzato tra i membri di una rete decentralizzata (rete peer-to-peer), il quale registra le transazioni, come lo scambio di beni o dati, tra i partecipanti della rete. Ciò che rende ogni DLT diverso sono:

- Una struttura dati.
- Un protocollo di registrazione delle transazioni.
- Un algoritmo di consenso per la validità delle transazioni.

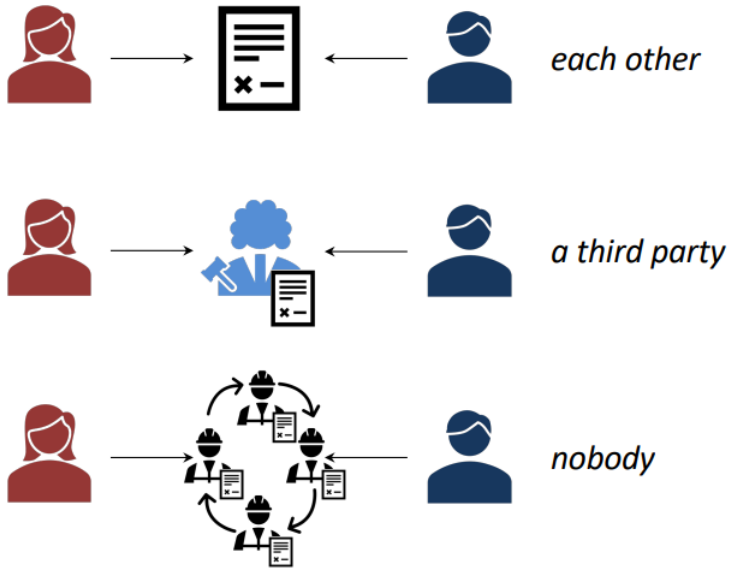

Figura 2.2: Gestione decentralizzata delle transazioni

Oltre alle classiche criptovalute la blockchain è uno strumento utile sia per la sicurezza che per la dematerializzazione dei documenti. Facendo un'analogia con il libro mastro, la blockchain è un registro distribuito il quale garantisce diverse proprietà:

- **Autenticità**: Le transazioni scritte nel libro mastro sono valide ed autentiche poiché tutti gli importi delle transazioni sono controllati e validati dalla rete.
- **Integrità**: Le transazioni all'interno del libro mastro sono immutabili e non possono essere cancellate. Nel caso in cui dovessi eseguire una modifica questa sarà inserita come una nuova transazione.
- **Disponibilità**: Il registro essendo distribuito tra i diversi nodi della rete ha un elevata disponibilità.
- **Accountability**: Le transazioni sono trasparenti e tracciabili.
- **Anonimato**: Gli utenti sono identificati da chiavi crittografiche pubbliche.

In generale l'architettura delle tecnologie a registro distribuito e dunque l'uso della blockchain possono portare a diversi vantaggi nell'ambito della cybersecurity. Una blockchain, infatti permette di applicare diverse tecniche di difesa:

- **Isolamento:** Ogni nodo è fisicamente isolato dagli altri. Per poter prendere il controllo l'attaccare deve controllare il 50%+1 dei nodi.
- **Monitoraggio:** Ogni nodo monitora le transazioni prima di approvarle.
- **Offuscamento:** Nell'esecuzione delle transazioni e nella struttura dati della blockchain sono utilizzate diverse primitive crittografiche.

Oltre alle varie tecniche di difesa, le tecnologie blockchain aumentano in generale l'affidabilità del sistema e quindi la sua dependability (fidatezza). Inoltre la blockchain garantisce:

- **Ridondanza:** Ogni nodo è ridondante in quanto mantiene la stessa copia dei dati.
- **Diversità:** Ogni nodo è esattamente una replica degli altri, ma appartiene ad utenti diversi.
- **Distribuzione:** Ogni nodo è geograficamente distribuito.

2.2.1 Struttura Dati

Una blockchain dunque è un registro digitale condiviso che registra le transazioni in una rete peer-to-peer pubblica o privata. Distribuito a tutti i nodi membri della rete, il libro mastro registra in modo permanente, in una catena sequenziale di blocchi crittografici collegati, la storia delle transazioni che avvengono tra i peer della rete. Le tecnologie blockchain sono dunque caratterizzate da una lista concatenata di blocchi (figura 2.3). Ogni nuovo blocco ha una capacità di storage limitata ed è costituito da diversi elementi:

- Un puntatore caratterizzato da un valore di hash del blocco precedente, in modo tale da stabilire con certezza l'ordine dei blocchi. Il primo blocco non avendo alcun puntatore al precedente è denominato blocco *genesi*.

- Un marcatore temporale (timestamp), relativo alla creazione del blocco, ed un nonce (un numero casuale) i quali permettono di identificare in maniera univoca il blocco.

- Le transazioni le quali sono inserite in un blocco ed organizzate secondo una struttura ad albero chiamata Merkel Tree.

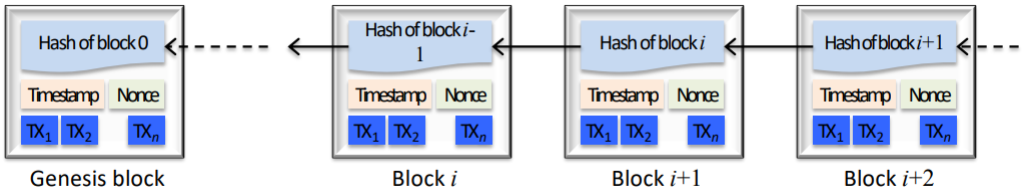

Figura 2.3: Struttura dati di una blockchain

2.2.2 Consenso

Per poter distribuire le transazioni all'interno della rete ed aggiungere un nuovo blocco alla catena è necessario che una maggioranza di nodi validatori raggiunga il consenso. A seconda del numero e della tipologia dei nodi validatori possiamo definire tre diverse tipologie di blockchain:

- **Public**: Sono reti di tipo permissionless in cui l'accesso è pubblico ed ogni nodo può partecipare alla validazione delle transazioni all'interno della rete utilizzando le proprie capacità di calcolo. In questo caso i nodi vengono chiamati *miner* in quanto le capacità di calcolo utilizzate per validare una transazione sono ricompensate.

- **Consortium**: Sono reti di tipo permissioned in cui solo i nodi appartenenti ad un consorzio di organizzazioni possono essere validatori. L'accesso alla rete può rimanere comunque pubblico.

- **Private**: Reti permissioned in cui è possibile partecipare sono nel caso in cui si è autenticati.

Nel caso di reti permissioned non è necessario ricompensare i nodi validatori e dunque non è presente alcuna reward per la validazione dei blocchi. La gestione della validazione delle transazioni deve essere gestita in maniera uguale da tutti i nodi della rete i quali utilizzano un meccanismo comune di consenso. Ogni blockchain dunque, utilizza un proprio protocollo per gestire il consenso. Tra i protocolli più utilizzati ci sono:

- **Proof of Work**: Utilizzata nelle reti di tipo permissionless in cui tutti i nodi miner per poter validare una transazione devono risolvere un puzzle crittografico. Il principale svantaggio di questo tipo di meccanismo è l'elevato consumo elettrico di energia dei nodi miner.

- **Proof of Stake**: Utilizzato nelle reti permissionless in sostituzione del PoW. Per poter partecipare come validatore, un nodo deve dimostrare di possedere una certa quantità di moneta. In questo caso non è presente alcuna ricompensa per le generazione di un nuovo blocco, ma gli utenti vengono ricompensati con una cosiddetta transaction fee. Il principale svantaggio è che solo chi possiede una certa quantità di moneta può effettivamente partecipare alla validazione, per questo motivo sono diverse le varianti suggerite in questi ultimi anni (es. selezione randomica, selezione coin-age based).
- **Bizantine Fault Tolerance**: Meccanismo di consenso utilizzato nelle reti di tipo permissioned. Il vantaggio principale di questo tipo di meccanismo è che la rete è robusta rispetto ad numero di nodi malevoli. Secondo questo meccanismo ogni volta che un nuovo blocco deve essere forgiato, secondo alcune regole viene definito un nodo validatore primario mentre gli altri nodi sono chiamati nodi di backup. Un client che deve eseguire una transazione la invia al nodo primario il quale trasmette in multicast la richiesta a tutti gli altri nodi di backup. Tutti i nodi inviano una risposta al client che è in ascolto. La transazione sarà considerata valida quando il client riceve il $33.3\% + 1$ di risultati uguali dai vari nodi. Una implementazione del meccanismo è data dal protocollo *IBFT* (Instanbul Pratcial Fault Tolerance) il quale rende la rete tollerante fino ad un numero f di nodi malevoli in una rete composta da $3f + 1$ nodi. La caratteristica principale dei protocolli di tipo BFT è la tolleranza della rete rispetto ad un numero di nodi malevoli. Data una rete composta da $3f + 1$ nodi, il protocollo assicura che questa sia tollerante rispetto ad un numero f di nodi malevoli.
- **Proof of Authority**: Protocollo utilizzato nelle reti permissioned nel quale solo nodi autorizzati all'interno della rete possono partecipare come nodi validatori. Molti algoritmi basati sul BFT possono essere utilizzati in queste tipologie di reti.

2.2.3 Transazioni

Ciascun utente per poter eseguire una transazione deve possedere una coppia di chiavi pubblica-privata, ed un portafoglio elettronico (wallet). Ogni portafoglio è identificato da un indirizzo il quale è stato generato a partire dalla chiave pubblica dell'utente. Per generare una transazione l'utente deve dimostrare di possedere la chiave privata corrispondente al proprio indirizzo pubblico. Ogni wallet è anonimo poiché non identifica in maniera univoca un utente in quanto non è possibile risalire allo stesso a partire dall'indirizzo pubblico. La figura 2.4 mostra un esempio di transazione:

1. L'utente Alice inizializza una transazione utilizzando il suo wallet.
2. La transazione è trasmessa ai nodi validatori ed inserita nella lista delle transazioni in *pending*.
3. Periodicamente i nodi validatori combinano le transazioni in attesa per forgiare un nuovo blocco.

4. I nodi validatori processano il nuovo blocco e raggiungono prima o poi un consenso. Una volta raggiunto il consenso la blockchain è aggiornata ed alcuni nodi sono ricompensati.

5. L'utente Bob legge la transazione utilizzando il suo wallet.

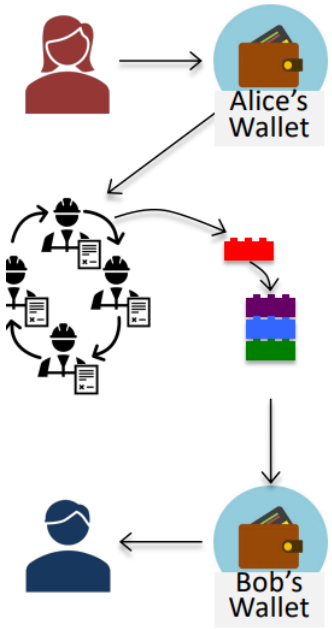

Figura 2.4: Esempio di transazione

2.2.4 Smart Contracts

Uno smart contract è una raccolta di codice (funzioni) e dati (stato) scritte in un linguaggio di programmazione la cui logica segue dei termini contrattuali [7]. Per capire il funzionamento di uno smart contract è importante definire il significato di contratto. Un contratto è un accordo tra due o più parti che stabilisce delle regole da rispettare. Una delle principali problematiche dei contratti è che possono essere soggetti ad interpretazione. A differenza di un normale contratto, uno smart contract è in grado di realizzarsi e di rispettare le clausole in modo autonomo e automatico, senza intermediari o mediatori. Essendo uno "script" scritto in un linguaggio di programmazione evita il problema dell'interpretazione personale poiché i termini del contratto sono decisioni e comandi all'interno del codice che li restituisce; inoltre uno smart contract è valido senza dipendere dalle autorità. Ciò è dovuto alla sua natura: è un codice visibile a tutti, che non può essere modificato in quanto presente sulla tecnologia blockchain. Questo gli conferisce un carattere decentralizzato, immutabile e trasparente. Gli smart contract [8] dunque consentono agli sviluppatori di creare app che sfruttano la sicurezza, l'affidabilità e l'accessibilità della blockchain, offrendo sofisticate funzionalità peer-to-peer.

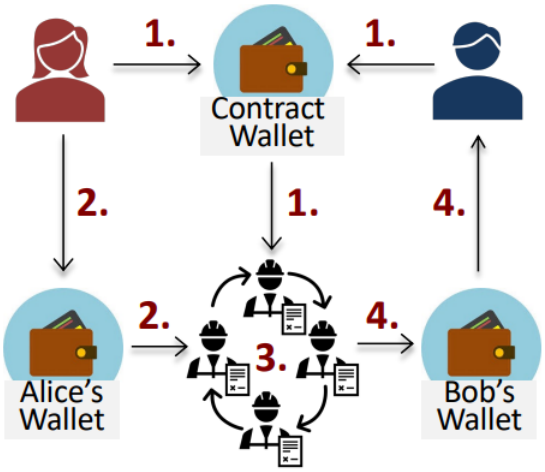

Figura 2.5: Ciclo di vita di uno smart contract

La figura 2.5 mostra il ciclo di vita di un semplice smart contract che esegue il passaggio di fondi da un account ad un altro:

1. I termini del contratto sono concordati da tutte le controparti e scritte in un linguaggio di programmazione. Una volta testato il suo funzionamento lo smart contract viene registrato all'interno di una blockchain. Ogni nodo della rete memorizza una copia dello smart contract e del suo stato attuale.

2. Un evento determina l'esecuzione di una clausola del contratto. All'interno della blockchain questo evento è rappresentato come una transazione.

3. Lo smart contrat viene eseguito in maniera automatica su tutti i nodi della rete secondo i termini contrattuali, e la transazione è validata quando si è raggiunto un consenso sull'esito da parte della maggioranza dei nodi. Questo meccanismo è molto importante in quanto consente un esecuzione sicura della clausola, senza che sia necessaria un'autorità centrale.

4. Il pagamento viene completato e la transazione viene convalidata e registrata all'interno di un blocco.

Capitolo 3

Tecnologie e strumenti utilizzati

3.1 Python

Python è un linguaggio di programmazione di alto livello orientato agli oggetti e multi-paradigma. Sebbene Python venga in genere considerato un linguaggio interpretato, il codice sorgente non viene convertito direttamente in linguaggio macchina. Infatti passa prima da una fase di pre-compilazione in bytecode evitando così di reinterpretare ogni volta il sorgente e migliorando le prestazioni. Inoltre è possibile distribuire programmi Python direttamente in bytecode, saltando totalmente la fase di interpretazione da parte dell'utilizzatore finale e ottenendo programmi Python a sorgente chiuso [9]. La sua semplicità di sintassi e modularità hanno reso Python ad oggi uno dei linguaggi di programmazione più utilizzato dagli sviluppatori di tutto il mondo (figura 3.1).

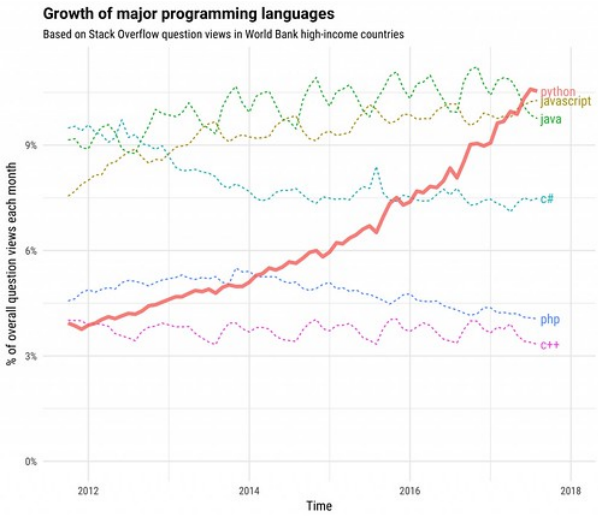

Figura 3.1: Andamento Python [10]

Python, infatti rappresenta spesso la lingua preferita dagli sviluppatori e data scientist che hanno bisogno di applicare tecniche statistiche o di analisi dei dati nel loro lavoro. La combinazione di sintassi coerente, i tempi di sviluppo più brevi e la flessibilità rendono tale linguaggio adatto allo sviluppo di modelli di

previsione sofisticati che possono essere collegati direttamente ai sistemi di produzione ed apprendimento automatico. Questo è possibile anche grazie all'ampio set di librerie, ossia insiemi di routine e funzioni scritte che svolgono un determinato compito, che lo sviluppatore può richiamare a seconda delle necessità. Sono diverse le librerie utilizzate nella tesi:

- **Pandas:** è una libreria software per la manipolazione e l'analisi dei dati. In particolare, offre strutture dati (DataFrames e Series) e operazioni per manipolare tabelle numeriche e serie temporali. La libreria è stata utilizzata durante i task di data analysis per rappresentare i log in dataframe ed effettuare diverse operazioni utili.

- **Sklearn:** è una libreria utilizzata principalmente per analisi di tipo predittivo; in particolare è possibile effettuare task di classificazione, regressione e clustering. La libreria è stata utilizzata per generare dei modelli tramite l'algoritmo di clustering KMeans il quale è utilizzato per rilevare eventuali anomalie di rete.

3.2 Ethereum

Ethereum è una piattaforma blockchain pubblica e di tipo permissionless progettata per l'esecuzione di smart contracts [11]. Tra le caratteristiche principali dell'architettura sono l'assenza di meccanismi di privacy delle transazioni le quali hanno un costo. Ethereum, infatti utilizza una propria criptovaluta chiamata *ETH*, ma permette anche la creazione di custom asset chiamati *tokens*.

Figura 3.2: Logo di Ethereum

Poiché la rete Ethereum permette l'esecuzione di smart contract, questa deve possedere un proprio stato il quale può cambiare da una transazione all'altra. Per questo motivo tutti i nodi per poter partecipare alla rete necessitano di una *Ethereum Virtual Machine* (EVM). La Macchina Virtuale di Ethereum è un computer virtuale e globale nel quale lo stato è memorizzato e concordato da ogni partecipante della rete Ethereum [12]. La rete Ethereum infatti mantiene traccia del proprio stato grazie all'utilizzo di una grande struttura dati che contiene non solo tutti i conti dei partecipati ma anche uno stato della macchina, che può cambiare da blocco a blocco secondo una serie predefinita di regole e su cui è possibile eseguire del codice arbitrario. Le regole specifiche di cambio stato sono appunto definite dall'EVM.

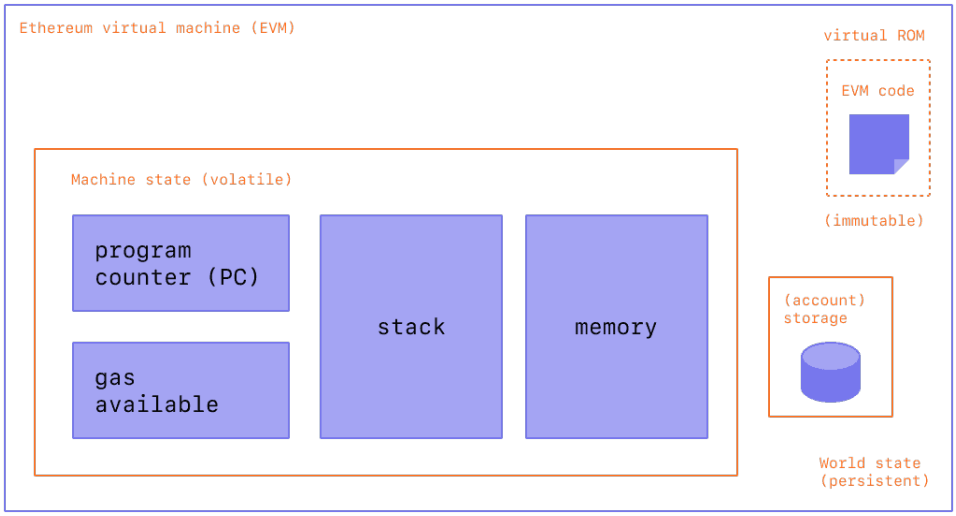

Figura 3.3: Ethereum Virtual Machine

Per quanto riguarda i meccanismi di consenso implementati da Ethereum, dato l'elevato consumo elettrico dell'algoritmo Proof of Work il quale richiedeva ai diversi nodi minatori di risolvere complicati puzzle crittografici per poter forgiare un nuovo blocco, a partire da settembre 2022 la piattaforma utilizza un meccanismo basato su un algoritmo di consenso di tipo Proof of Stake. In tale meccanismo di consenso chiunque voglia partecipare come nodo validatore e dunque aggiungere nuovi blocchi alla catena deve possedere un certo valore di ETH. Un validatore è un nodo della rete Ethereum il quale è responsabile di verificare che i nuovi blocchi propagati sulla rete siano validi ed, occasionalmente, di crearne di nuovi. Per partecipare come validatore, infatti un utente deve depositare in staking 32 ETH in un contratto di deposito. Depositando i propri ETH, l'utente si unisce ad una coda di attivazione che permette di limitare la partecipazione di nuovi validatori all'interno della rete. Una volta attivati, i validatori ricevono nuovi blocchi dai peer sulla rete di Ethereum. Le transazioni consegnate nel blocco sono eseguite e la firma del blocco viene verificata per assicurarsi che questo sia valido. Il validatore infine invia poi alla rete un voto a favore di quel determinato blocco. Una volta raggiunta la maggioranza dei voti il blocco e le relative transazioni saranno considerate valide ed il nuovo blocco è inserito nella blockchain. Per poter individuare sia gli utenti che gli smart contract l'architettura Ethereum utilizza un indirizzo identificativo di 20 byte. All'interno della piattaforma, infatti è possibile creare degli account i quali tengono traccia del proprio saldo. Gli account possono essere di due tipi:

- **Externally Owned Account (EOA):** Account di un determinato utente costituito da una propria chiave privata.

- **Contract Account:** Indirizzi degli smart contract i quali sono associati e controllati da un determinato codice.

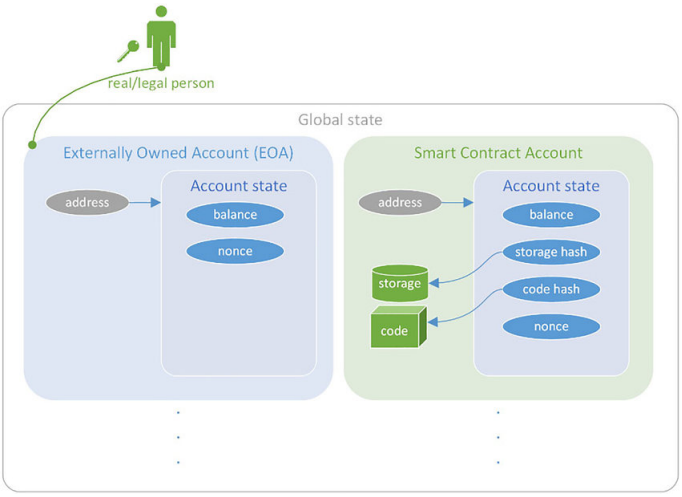

Figura 3.4: Account Ethereum

Oltre al saldo in Wei (10^{18} Wei corrispondono ad un ETH), lo stato di un account contiene diverse informazioni come il *nonce*, cioè il numero transazioni inviate da quell'account od il numero di contratti creati nel caso di contract account. Nel caso di contract account lo stato è associato anche ad uno *storage hash*, il quale rappresenta un hash dell'identificatore dello storage del contratto, ed un *code hash* il quale rappresenta l'hash del codice presente nel contratto.

3.2.1 Gas Fee

All'interno della piattaforma Ethereum ogni partecipante che trasmette una richiesta di transazione deve offrire un certo importo di ETH alla rete a titolo di ricompensa. La rete infatti elargirà tale ricompensa a chiunque svolga il lavoro effettivo verificando, eseguendo e trasmettendo la transazione all'interno della blockchain. Definiamo *Gas* come l'unità di misura della quantità di sforzo computazionale necessario per eseguire una determinata azione all'interno di Ethereum [13]. Poichè ogni transazione necessita di risorse di calcolo per essere eseguita, questa richiede una commissione. Ad ogni transazione infatti sono associati un *gas limit*, cioè il valore massimo di gas che un utente è disposto a spendere per la transazione, ed un *gas price* cioè la quantità in Wei che si è disposti a spendere per ogni unità di gas. Il prodotto di questi due parametri fornisce il *gas cost* che un utente è disposto a spendere per una determinata transazione. La figura 3.5 mostra un esempio di transazione: supponiamo che un sender voglia inviare una transazione ad un reciver, affinché questa sia eseguita è necessario che l'utente abbia abbastanza Ether nel proprio account per poter coprire il costo della gas fee e che il gas limit sia sufficiente per poter eseguire la transazione. Per quanto riguarda la gas fee questa viene depositata all'indirizzo del nodo validatore che ha depositato i propri ETH per poter validare la transazione ed inserirla in nuovi blocchi. Le gas fee rappresentano infatti un incentivo a contribuire alla rete per i validatori. Nel caso in cui del gas rimanga inutilizzato durante una transazione questo infine viene restituito al mittente.

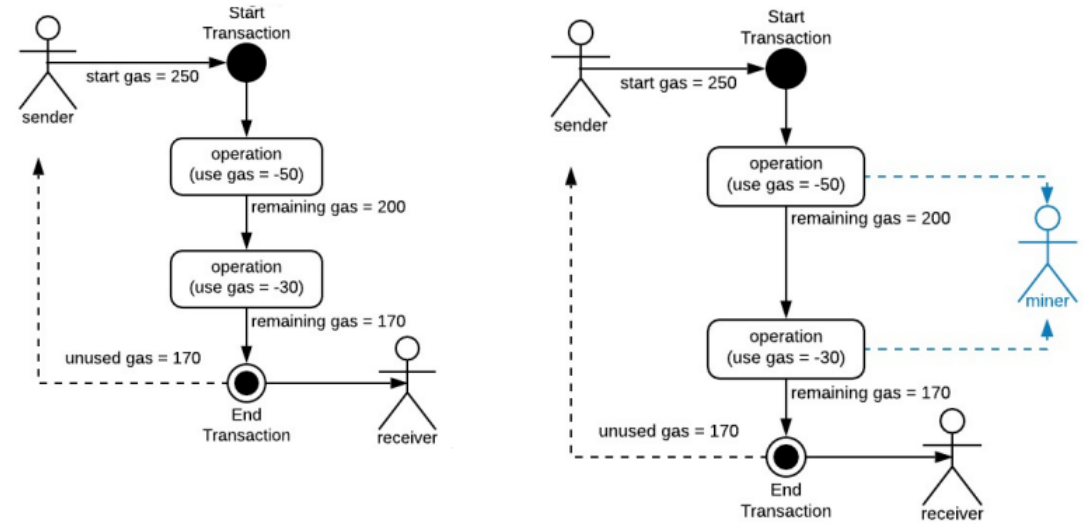

Figura 3.5: Esempio di transazione corretta

Nel caso in cui l'utente non abbia a disposizione abbastanza gas per poter eseguire una determinata transazione (figura 3.5), quest'ultima viene considerata non valida. I cambiamenti di stato che erano stati eventualmente eseguiti vengono invertiti e la transazione non riuscita viene registrata. Poiché la capacità di calcolo è già stata spesa dalla rete, l'utente non viene rimborsato del gas speso.

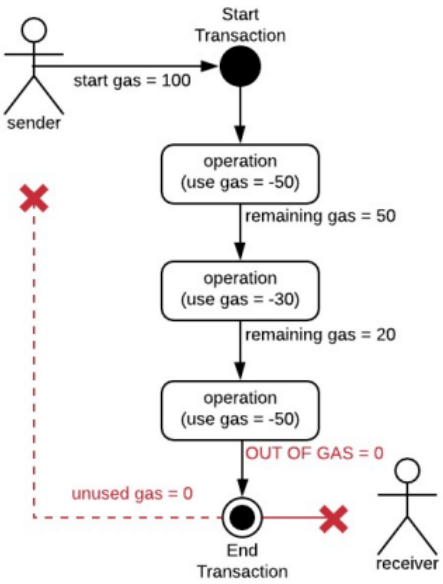

Figura 3.6: Esempio di transazione errata

3.2.2 DAPPs

Oltre allo scambio di fondi da un indirizzo all'altro, la piattaforma Ethereum può essere utilizzata per sviluppare applicazioni decentralizzate [14]. Un' applicazione decentralizzata (Dapp) è un applicazione la quale viene eseguita all'interno di una rete decentralizzata peer-to-peer e che sfrutta le potenzialità degli smart contract per eseguire diverse operazioni. A differenza di una semplice applicazione il codice backend rappresentato da uno smart contrat è eseguito in una rete decentralizzata e non all'interno di un server centralizzato. Una dapp può avere codice frontend e interfacce utente scritti in qualsiasi linguaggio (come qualsiasi app) che grazie all'uso di librerie come ad esempio Web3 possono interagire con il backend collegandosi alla blockchain ed interagendo con lo smart contract. La piattaforma Ethereum inoltre, permette l'uso degli smart contract in maniera trasparente in modo simile all'uso di API. Tutte le Dapp sono eseguite in nella EVM, in questo modo se lo smart contract contiene un bug, quest'ultimo non ostacolerà il normale funzionamento della rete.

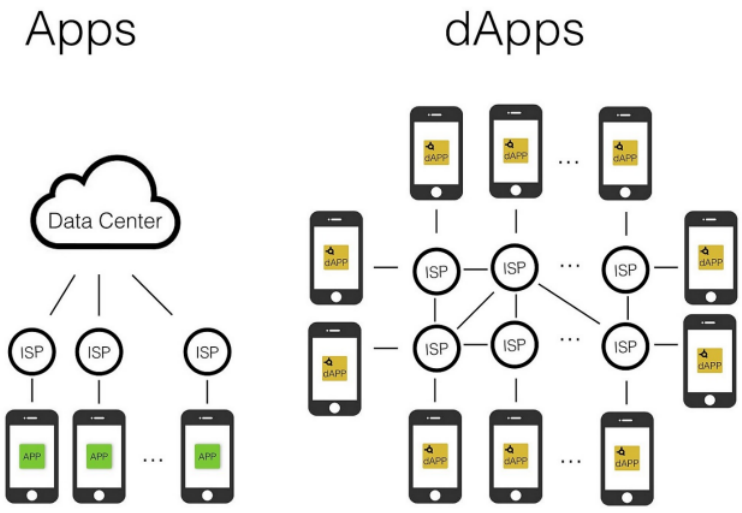

Figura 3.7: Esempio di DAPP

Lo sviluppo di daap presenta diversi vantaggi:

- **Decentralizzazione:** Operando su reti peer-to-peer pubbliche e decentralizzate, nessun individuo o gruppo detiene il controllo. Una volta distribuito il contratto intelligente sulla blockchain, l'intera rete potrà sempre servire i client che cercano di interagire con il contratto. Gli attori malevoli quindi non possono lanciare attacchi di tipo denial-of-service verso dapp singole.
- **Integrità:** I dati conservati sulla blockchain sono immutabili grazie all'uso di primitive crittografiche. Gli attori malevoli dunque non possono falsificare le transazioni o altri dati che sono già stati resi pubblici.
- **Privacy:** Non è necessario fornire un'identità reale per distribuire una dapp o interagirvi, ma è sufficiente utilizzare il proprio account Ethereum.

3.3 Hyperledger Besu

Hyperledger Besu è un client Ethereum open source il quale può essere eseguito sia sulla rete pubblica Ethereum sia su reti private autorizzate [15]. In generale un client Ethereum è un software che implementa il protocollo Ethereum il quale è caratterizzato da un ambiente di esecuzione per l'elaborazione delle transazioni così come un archivio dei dati relativi alla loro esecuzione. Il client dispone di strumenti utili sia ad implementare delle reti peer-to-peer (P2P) per far comunicare i vari nodi della blockchain tra loro, sia di API utili agli sviluppatori per poter interagire con la blockchain.

Figura 3.8: Logo di Hyperledger Besu

Come ogni rete Ethereum il client dispone di una Ethereum Virtual Machine (EVM) per l'esecuzione degli smart contracts. Besu utilizza due tipi di algoritmi di consenso:

- **Proof of Authority (PoA)**: Besu implementa diversi protocolli PoA come ad esempio IBFT, QBFT e Clique i quali sono utilizzati quando i partecipanti alla blockchain si conoscono ed il livello di fiducia tra di loro è elevato. Le transazioni e i blocchi sono convalidati da accounts approvati, noti come validatori, che a turno, creano il blocco successivo. I validatori esistenti propongono e votano per aggiungere o rimuovere altri validatori.
- **Proof of Stake(PoS)**: Il meccanismo PoS sceglie casualmente i validatori per proporre o convalidare i blocchi. I nodi proponenti sono responsabili della proposta di nuovi blocchi di consenso, mentre i nodi validatori sono responsabili della convalida dei blocchi proposti. Quest'ultimi sono ricompensati per aver proposto e attestato blocchi di consenso che alla fine vengono inclusi nella blockchain, mentre vengono penalizzati in caso di comportamento dannoso.

La figura 3.9 mostra l'intera l'architettura dell'execution core di Besu la quale fornisce diverse funzioni utili [16]:

- **La macchina virtuale di Ethereum (EVM):** l'EVM consente la distribuzione e l'esecuzione di smart contract Ethereum.
- **Algoritmi di consenso:** oltre a un algoritmo proof-of-work, Besu dispone anche di diversi protocolli proof-of-authority, che si prestano bene all'uso in consorzi blockchain o altre reti private in cui i partecipanti si conoscono.
- **Rete P2P:** sfruttando i protocolli di rete devp2p di Ethereum, Besu è in grado di facilitare la comunicazione tra i clienti.

- **Discovery:** un protocollo basato su UDP (user datagram protocol).
- **RLPx:** protocollo basato su TCP con due sottoprotocolli: Ethereum Wire Protocol e IBF sub protocol per IBFT2.0.
- **Storage:** Hyperledger Besu utilizza un database RockDB a valore-chiave per conservare localmente i dati della catena. I dati sono suddivisi in due sottocategorie: i dati della blockchain e i dati del world state. In particolare i dati della blockchain contengono i block headers, i block bodies (transazioni ordinate), le ricevute delle transazioni ed i transaction logs. I dati del world state, invece sono una mappatura dagli indirizzi agli accounts: gli account esterni contengono ether balance, mentre gli smart contracts contengono codice eseguibile.
- **Permissioning:** pur sfruttando la mainnet pubblica di Ethereum, Besu ha la capacità di mantenere private le transazioni tra le parti coinvolte consentendo la partecipazione solo a nodi ed account specifici, abilitando l'autorizzazione del nodo e/o l'autorizzazione dell'account sulla rete.
- **Privacy:** Tramite il gestore di transazioni private Tessera è possibile garantire la privacy delle transazioni tra le controparti.
- **API:** il client fornisce API JSON-RPC per la mainnet Ethereum basandosi su protocolli HTTP, WebSocket o API GraphQL per l'iterazione con l'utente e gli smart contract.
- **Monitoraggio:** Hyperledger Besu utilizza strumenti come Prometheus o Block Explorer per consentire agli utenti di monitorare le prestazioni dei nodi e della rete.

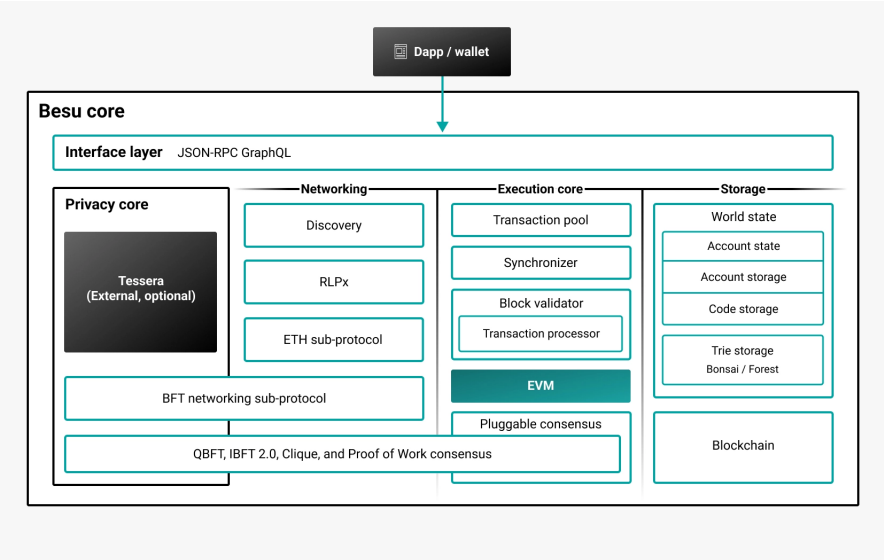

Figura 3.9: Architettura Besu [16]

Durante il tirocinio si è scelto di utilizzare il client Besu in quanto presenta la possibilità di creare reti di tipo permissioned tramite l'implementazione di algoritmi di consenso di tipo PoA. Inoltre la blockchain Besu è configurabile grazie ad un ampio set di opzioni le quali permettono di impostare la connessione e gestire le transazioni. Per quanto riguarda l'algoritmo di consenso, in questo caso si è utilizzato l'algoritmo QBFT (Quorum Bizantine Fault Tolerance). Si è scelto un algoritmo di tipo QBFT in quanto, insieme al protocollo IBFT2.0, è quello consigliato dal team di Hyperledger Besu per la creazione e la gestione di production-network. Inoltre a differenza di un protocollo di tipo Clique, il protocollo ha una finalità immediata poiché non presenta fork della blockchain e tutti i blocchi sono validati ed inseriti nella main chain. Inoltre QBFT offre una maggiore ridondanza in quanto per il funzionamento della rete sono necessari un minimo di 4 validatori per poter generare blocchi, a discapito del singolo validatore richiesto da Clique.

3.4 Solidity

Solidity [17] è un linguaggio di alto livello per lo sviluppo di smart contracts da eseguire sulla EVM (Ethereum Virtual Machine). Solidity supporta diverse caratteristiche come una tipizzazione dei dati statica, l'uso di librerie e la possibilità di definire tipi di dati strutturati.

Figura 3.10: Logo di Solidity

Il linguaggio riprende alcuni dei paradigmi della programmazione ad oggetti come l'incapsulamento e l'ereditarietà. I contratti in Solidity, infatti sono simili alle classi dei linguaggi object-oriented. Ogni contratto è costituito da diversi elementi [18]:

- **Variabili di stato**: Sono le variabili memorizzate all'interno del contratto. In analogia con le classi, le variabili di stato rappresentano gli attributi della classe.

- **Funzioni**: Unità eseguibili di codice definite internamente o esternamente al contratto le quali accettano parametri in input e possono ritornare variabili.

- **Modificatori:** I modificatori possono essere utilizzati per modificare il comportamento delle funzioni in maniera dichiarativa, come ad esempio per verificare una condizione prima di eseguire una determinata funzione. I modificatori sono definiti tramite la parola chiave `modifier`, mentre le condizioni possono essere facilmente definite tramite il costrutto `require`, il quale prende in input la condizione da verificare ed il messaggio di errore da visualizzare in caso di condizione non verificata (in questo caso la require esegue il revert della transazione che fallisce). Il modificatore è stato utilizzato durante la tesi in modo tale che solo il proprietario del contratto possa eseguire le diverse clausole (esempio figura 3.11).

- **Eventi:** Gli eventi Solidity forniscono un'astrazione delle funzionalità di logging dell'EVM. Le applicazioni possono sottoscrivere e ascoltare questi eventi attraverso l'interfaccia RPC di un client Ethereum. Quando vengono chiamati, gli argomenti dell'evento sono memorizzati nel log della transazione. Questi log sono associati all'indirizzo del contratto e sono incorporati nella blockchain stessa e vi rimangono finché un blocco è accessibile. Durante la tesi gli eventi sono stati utilizzati per registrare una notarizzazione avvenuta con successo.

```
    modifier onlyOwner {
        require(
            msg.sender == owner,
            "Only owner can call this function."
        );
        _;
    }
}
```

Figura 3.11: Esempio di modifier

Per quanto riguarda l'incapsulamento, sia le variabili di stato che le funzioni possono avere diversi livelli di visibilità:

1. Il livello *public* indica che la variabile o la funzione possono essere richiamate sia internamente che esternamente tramite message calls.

2. Il livello *internal* indica funzioni e variabili le quali possono essere accedute all'interno del contratto in cui sono state definite e nei contratti che ne derivano (In analogia con le classi i contratti derivati sono le sottoclassi del contratto dunque si sfrutta l'ereditarietà).

3. Il livello *private* indica funzioni e variabili accessibili solo all'interno del contratto stesso.

4. Solo per le funzioni è possibile definire il livello *external* il quale permette alle funzioni di interagire come se fossero delle interfacce e quindi di essere richiamate solo da altri contratti e da transazioni esterne.

5. Solo per le funzioni è possibile definire il livello *view* il quale affiancato agli altri modificatori di visibilità indica una funzione in solo lettura la quale quindi non modifica in alcun modo le variabili di stato ma offre solo funzionalità di visualizzazione.

Per quanto riguarda i dati Solidity utilizza diversi tipi di dato i quali possono essere:

- **Value Types:** Le variabili di questo tipo sono sempre passate per valore cioè sono copiate durante l'assegnamento o quando sono passate in input alle funzioni. Oltre ai classici tipi numerici, booleani, stringhe ed enum, Solidity definisce i tipi address i quali rappresentano gli indirizzi Ethereum e Quorum. Come tutti gli indirizzi sono costituiti da 20 byte e possono essere di due tipologie: *address* ed *address payable*. Il primo non può ricevere valuta ETH dallo smart contract, mentre il secondo permette l'utilizzo delle funzioni send e transfer per l'ottenimento di fondi.

- **Reference Types:** Le variabili di questo tipo sono sempre passate per riferimento. Includono tipi di dato come structs, array e mappings. Nell'uso di questi tipo di dati è necessario specificare la locazione di memoria in cui il dato è immagazzinato tramite tre diverse parole chiave:

    1. **Memory:** Il lifetime di questi dati è limitata alla chiamata di una funzione.
    2. **Storage:** Indica la locazione di memoria dove il dato è memorizzato, il lifetime di questi dati è limitato alla durata del contratto.
    3. **Calldata:** È valido solo per i parametri delle funzioni esterne del contratto. È un'area non modificabile, non persistente, in cui sono memorizzati gli argomenti delle funzioni.

In conclusione Solidity è un linguaggio molto utile per la definizione di smart contract, durante la tesi il linguaggio e le sue funzionalità sono state utilizzate per sviluppare uno smart contract il quale gestisce la notarizzazione dei log di sicurezza.

3.5 Truffle Suite

La suite di truffle [19] è un insieme di tools per lo sviluppo e l'integrazione di smart contracts. Truffle mette a disposizione un ambiente di sviluppo, un framework di test per le blockchain che utilizzano la macchina virtuale di Ethereum (EVM) in modo da semplificare lo sviluppo, il test e l'intergrazione degli smart contracts.

Figura 3.12: Logo di Truffle

Tra le principali funzionalità messe a disposizione da Truffle ci sono:

- Compilazione, distribuzione e gestione binaria degli smart contracts.
- Distribuzione transazioni tramite Metamask, un portafoglio elettronico in grado di gestire i propri account.
- Test automatizzati degli smart contract per uno sviluppo più rapido.
- Gestione della rete per il deploy su qualsiasi numero di reti pubbliche e private.

E' possibile installare la suite tramite il package npm di Node.js utilizzando il comando *npm install -g truffle*. All'interno della suite è presente Ganache [20], un software open-source in grado di creare una blockchain personale in maniera semplice ed immediata per lo sviluppo ed il test di applicazioni decentralizzate. Il software è disponibile sia in versione CLI che con interfaccia grafica.

Figura 3.13: Logo di Ganache

L'applicazione una volta avviata permette di creare una blockchain ethereum locale e privata, e di specificare diverse opzioni come il numero di account da generare con il relativo balance, un file di configurazione truffle con il quale è possibile interagire ed effetture il deploy di eventuali smart contract sviluppati ed altre opzioni.

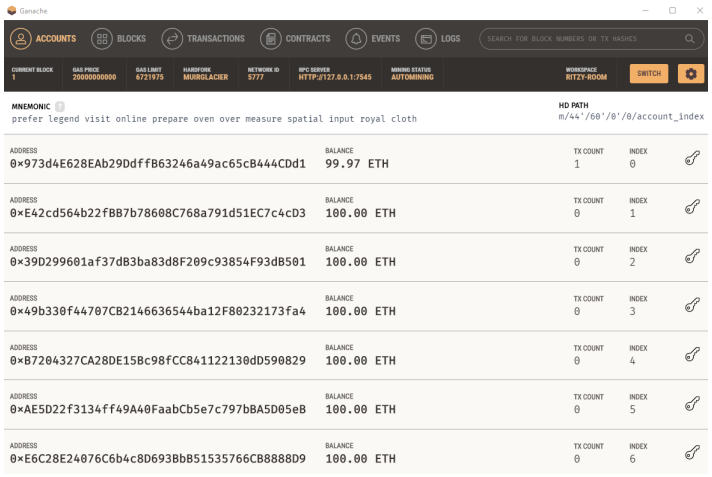

Figura 3.14: Esempio di blockchain ed account generati da ganache

La suite di Truffle e Ganache sono stati utilizzati sia durante la fase di sviluppo e test dello smart contract sia per il successivo deploy dello stesso sulla blockchain permissioned Hyperledger Besu in fase di pre-produzione.

3.6 Web3

Web3 [21] è un insieme di librerie le quali permettono di interagire con il proprio nodo blockchain remoto o locale in maniera semplice ed intuitiva tramite l'utilizzo di diversi protocolli come HTTP, IPC o WebSocket. La libreria è stata utilizzata per eseguire le transazioni ed integrare il firmware python del dispositivo LECS alla blockchain in modo da richiamare ed utilizzare lo smart contract per la notarizzazione automatica dei log.

Capitolo 4

System Architecture

La tecnologia di LECS si basa su 3 engine i quali tramite l'uso di tecniche di machine learning proteggono un segmento di rete eseguendo sia la detection che la classificazione delle minacce e delle anomalie in tempo reale, agendo con contromisure e risposte mirate nei casi di minacce critiche [4]. Tutti e tre gli engine agiscono in maniera sinergica e parallela in modo da fornire un intero ecosistema di difesa di un segmento di rete. I tre engine sono descritti di seguito:

- **Specto**: E' un modello di Hidden Analysis Classification Logging (HACL) che gestisce in maniera automatica le minacce in real-time. L'approccio utilizzato dall'algoritmo è un approccio di tipo stealth che permette di lavorare in maniera semplice e parallela all'analisi e la classificazione delle minacce all'interno della rete, senza ostacolare la normale operatività della stessa.

- **Raises**: Nel caso in cui venisse rilevata una criticità ad alto impatto, la contromisura automatica interviene in maniera procedurale ed adattiva, utilizzando diversi step di risposta fino ad arrivare all'ultima difesa di risposta energetica di contromisura elettronica brevettata nei casi più critici. In questo modo è possibile effettuare air gap energetici in maniera completamente automatica per isolare e mettere in sicurezza il segmento di rete attaccato.

- **Tiresia**: Tutte le operazioni di detection e classificazione sono supportate dal un algoritmo intelligente, che in maniera continua apprende e migliora le capacità di previsione aggiornando in maniera periodica i dati relativi alle minacce. L'algoritmo esegue delle vere e proprie predizioni di cyber threat forecast, aumentando in questo modo notevolmente le capacità di detection dell'intero ecosistema.

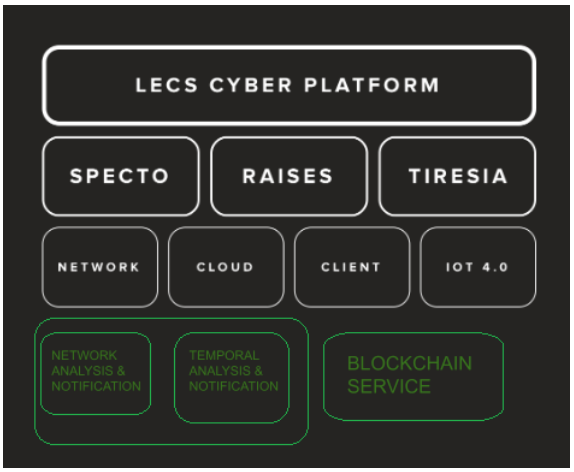

Figura 4.1: LECS Cyber Platform

La figura 4.1 mostra l'intera piattaforma di LECS. Durante la tesi si sono sviluppati diversi nuovi moduli che sono mostrati in verde i quali si occupano del sistema di analisi, notifica e notarizzazione dei log di sicurezza generati dal dispositivo. I vari task sviluppati sono inseriti in un' architettura di sistema la quale può essere rappresentata ad alto livello dal diagramma in figura 4.2. Il sistema è composto da diverse entità:

- **LECS Device:** Blackbox per il monitoraggio e la protezione delle infrastrutture critiche. Agisce come sentinella andando a monitorare l'infrastruttura di rete a cui è collegato e generando eventi di sicurezza relativi a possibili minacce e/o atacchi.

- **Blockchain Service:** Servizio di notarizzazione attivo su una architettura blockchain di tipo permissioned.

- **Event Analysis Engine:** Infrastruttura server centrale composta da un database in cui sono immagazzinati i log di sicurezza e di rete generati dal dispositivo. Oltre alle capacità di storage il server esegue attività di controllo temporale delle minacce rilevate dal dispositivo e analisi dei dati di rete per poter notificare l'utente.

- **User:** Utente al quale è associato un dispositivo LECS. Oltre ad essere notificato di eventuali minacce rilevate, ha accesso ad una dashboard riguardante statistiche di rete ed altre informazioni. La dashboard è aggiornata in tempo reale con gli ultimi dati immagazzinati nel database.

L'intera architettura può essere associata alle funzionalità di un SIEM in quanto il dispositivo agisce come una sentinella in grado di generare eventi di sicurezza, mentre l'event analyisis engine agisce sia come strumento di storage permanente che di analisi.

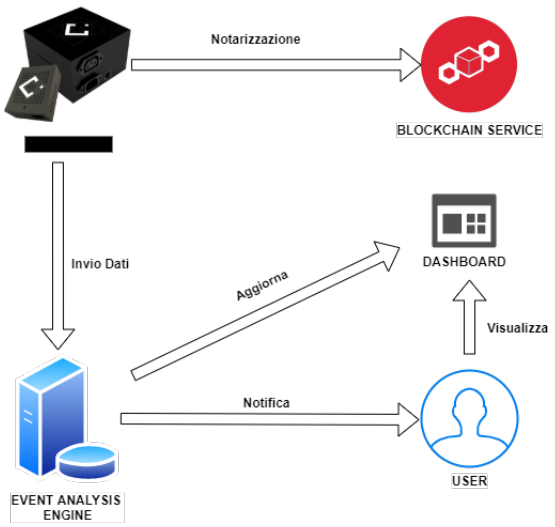

Figura 4.2: System Architecture

Per quanto riguarda il sistema di notifica, un rilevamento di una determinata sequenza temporale nelle classificazioni o un anomalia di rete sono gestiti all'interno del server. Un esempio di sequenza è descritta nel diagramma 4.3.

1. Il dispositivo monitora l'infrastruttura a cui è collegato ed invia in real-time diverse informazioni. Oltre ai log di sicurezza relativo alle minacce classificate tramite l'uso di Tiresia e Specto, il dispositivo scansiona la rete. Tramite la funzione send packet queste informazioni sono inviate ed immagazzinate in diverse tabelle di un database server.

2. Le informazioni relative alle caratteristiche di rete individuate sono utilizzate per calcolare indici come ad esempio l'entropia o per poter individuare anomalie nei pacchetti scambiati tramite l'uso del KMeans. Per quanto riguarda invece le classificazioni delle eventuali minacce rilevate dal dispositivo, esse sono analizzate in modo tale da verificare il loro andamento temporale. Un aumento eccessivo delle minacce e/o possibili attacchi in un determinato arco temporale, infatti può risultare in una maggiore vulnerabilità della rete.

3. Nel caso in cui si sia rilevata una sequenza temporale anomala nelle classificazioni oppure un' anomalia di rete rispetto al modello KMeans sviluppato, il sistema invia una notifica all'utente.

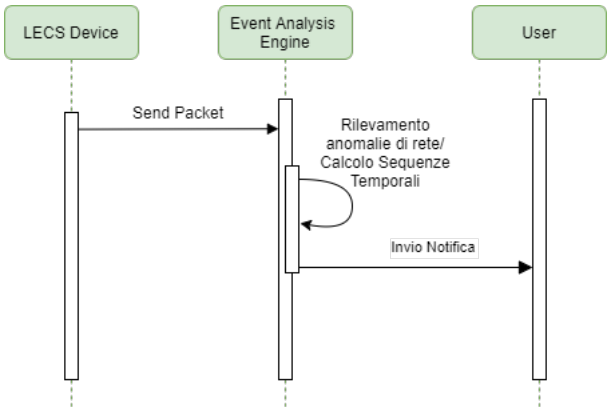

Figura 4.3: Analisi di Dati

Nel caso di attacchi critici di livello 1 il dispositivo, oltre ad inviare i dati dell'attacco al server che andrà immediatamente a notificare l'utente, andrà a notarizzare i dati nella blockchain. In questo modo i dati relativi all'attacco saranno resi immutabili e certificati all'interno di una transazione.

1. Il dispositivo rileva una minaccia critica di livello 1 ed invia i dati dell'attacco al server.

2. Gli stessi dati sono utilizzati per richiamare una funzione di uno smart contract che si occupa della notarizzazione degli attacchi. La transazione è inserita nel pool di quelle in attesa e sarà prima o poi validata.

3. Il dispositivo tramite l'engine Raises esegue una contromisura elettronica brevettata inviando in maniera immediata una notifica di alert.

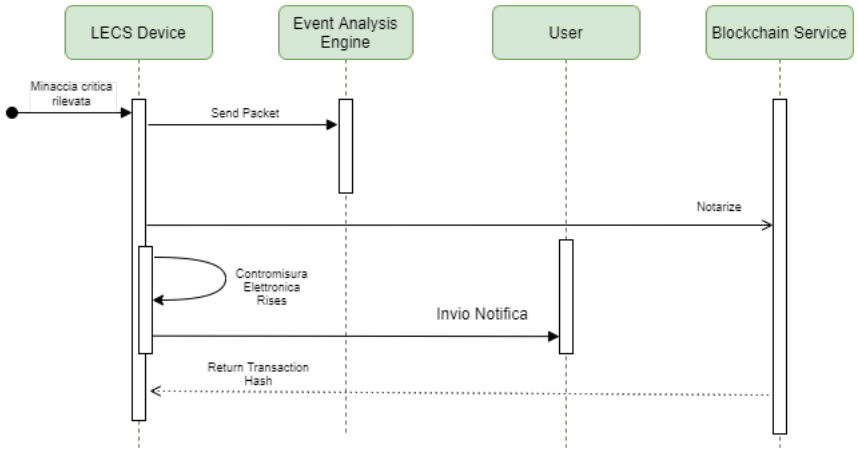

Figura 4.4: Notarizzazione log in Blockchain

Capitolo 5

Log Data Analysis

L'analisi delle caratteristiche e delle anomalie è stato effettuato prendendo in considerazione dei log di rete relativi ai pacchetti. Il dispositivo LECS tramite il tool ethtool legge diverse informazioni dai driver della scheda di rete o dal sistema operativo e li invia in real-time ad un server il quale li memorizza in un database. I campi utilizzati nell'analisi sono descritti nella seguente tabella:

Nome	**Tipo**	**Descrizione**
idlog	Stringa	Identificativo del log
rx_64	Stringa	Numero incrementale di pacchetti di tipo rx64 ricevuti dal sistema
rx_65	Stringa	Numero incrementale di pacchetti di tipo rx65 ricevuti dal sistema
rx128	Stringa	Numero incrementale di pacchetti di tipo rx128 ricevuti dal sistema
rx256	Stringa	Numero incrementale di pacchetti di tipo rx256 ricevuti dal sistema
rx512	Stringa	Numero incrementale di pacchetti di tipo rx512 ricevuti dal sistema
rx1024	Stringa	Numero incrementale di pacchetti di tipo rx1024 ricevuti dal sistema
rx1522	Stringa	Numero incrementale di pacchetti di tipo rx1522 ricevuti dal sistema
Time	Datetime	Data ed ora della rilevazione
fkSeriale	Stringa	Codice identificativo del dispositivo LECS
rx_error	Number	Numero incrementale di pacchetti errati al momento della rilevazione
PPS	Number	Numero di pacchetti al secondo ricevuti dal sistema al momento della rilevazione
BPS	Number	Numero di byte al secondo ricevuti dal sistema al momento della rilevazione
rxTot	Number	Numero di pacchetti totali ricevuti dal sistema al momento della rilevazione

rx_UDP	Number	Numero incrementale di pacchetti di tipo UDP ricevuti dal sistema
rx_TCP	Number	Numero incrementale di pacchetti di tipo TCP ricevuti dal sistema
rx_ICMP	Number	Numero incrementale di pacchetti di tipo ICMP ricevuti dal sistema
rx_PKT2	Number	Numero di pacchetti totali ricevuti dal sistema al momento della rilevazione

Tabella 5.1: Campi utilizzati dei log di rete

L'obiettivo del task di log analysis è quello di calcolare alcune statistiche di rete e di rilevare delle eventuali anomalie relative ai pacchetti ricevuti.

5.1 Calcolo Entropia Pacchetti Medi

Il primo task di analisi è quello relativo al calcolo dell'entropia del numero di pacchetti e della relativa connessione. Tale funzionalità fa parte del modulo Network analysis & anomaly notification di figura 4.1 che si occupa di analisi dei dati di rete e successiva notifica anomalie. Il calcolo sarà eseguito ogni dieci minuti e per ogni LECS identificato dal campo fkSeriale e successivamente inserito in una nuova tabella *entropy* creata all'interno del database. La tabella sarà composta dai seguenti campi:

Nome	**Tipo**	**Descrizione**
idEntropy	Stringa	Identificativo auto-incrementale relativo al calcolo di un' entropia
fkSeriale	Stringa	Identificativo del dispositivo LECS di cui si è calcolata l'entropia
Valore	Double	Valore di entropia calcolato
Time	Datetime	Data ed ora del calcolo dell'entropia
Tipo	Stringa	Tipo di pacchetti di cui si è calcolata l'entropia

Tabella 5.2: Campi della tabella relativa al calcolo dell'entropia

Per poter eseguire il calcolo dell'entropia è stato creato un script python il quale una volta connesso al database, estrae i dati tramite una query, la quale è stata definita in modo tale da ricavare i dati relativi agli ultimi 10 minuti. Sfruttando la funzione `read_sql_query` messa a disposizione da pandas i dati estratti sono stati inseriti all'interno di un dataframe. Poiché il calcolo dell'entropia deve essere eseguito per ogni dispositivo LECS il dataframe è stato suddiviso in componenti più piccoli andando ad effettuare un filtraggio tramite l'utilizzo del metodo *unique()*. Prima di effettuare tale divisione tuttavia il dataframe originale è stato aggiornato con nuove colonne per ogni tipo di pacchetto. Ogni colonna rappresenterà il valore medio di pacchetti ad ogni rilevazione ed il suo valore

sarà inizialmente impostato pari a 0 per poi essere successivamente aggiornato ed utilizzato nel calcolo dell'entropia nei passi successivi. Ogni dataframe ottenuto dalla fase di divisione viene elaborato dallo script il quale esegue diverse trasformazioni:

- **Calcolo Pacchetti Medi:** Poiché il numero di pacchetti tra una rilevazione e l'altra è incrementale, per il calcolo di entropia è necessario calcolare il numero medio intero di pacchetti al secondo per ogni rilevazione. Per effettuare tale task si è creata una funzione `crea_numero_pacchetti_medi` la quale prende in input la tipologia di pacchetti di cui si vuole calcolare il numero medio ed aggiorna la relativa colonna del dataframe creata precedentemente. Il numero di pacchetti medi per ogni rilevazione sarà calcolato come la divisione della differenza tra i pacchetti del record attuale e quello precedente ed il tempo trascorso tra una rilevazione e l'altra in secondi (calcolato come il deltatime tra le due rilevazioni). Il calcolo è riassunto dalla seguente formula:

$$MeanPackets = \frac{packets[i] - packets[i-1]}{get\_seconds(time[i] - time[i-1])} \tag{5.1}$$

  La funzione sarà eseguita per ogni tipologia di pacchetti.

- **Calcolo Entropia:** Una volta calcolato il numero di pacchetti medi si è passati al calcolo dell'entropia tramite una funzione la quale implementa la formula 2.2. Per ogni tipo di pacchetti tramite la struttura collection e la funzione counter messa a disposizione da python, viene calcolata la probabilità relativa del numero medio di pacchetti rilevati. Tale probabilità è calcolata tramite un' approccio empirico e dunque per ogni tipologia di pacchetto viene calcolata la frequenza relativa del numero di pacchetti medi sul totale delle rilevazioni. Siano n il numero totale di rilevazioni che possono assumere k possibili valori $v_1, v_2, ..., v_k$ (in questo caso rappresentati dal numero di pacchetti medi rilevati). La frequenza relativa del valore v_i indicata con $f_r(v_i)$ può essere calcolata tramite la seguente formula:

$$f_r(v_i) = \frac{f(v_i)}{n} \tag{5.2}$$

  Dove $f(v_i)$ rappresenta la frequenza con cui un determinato valore è presente nell'insieme delle rilevazioni considerate. Una volta ottenute tutte le frequenze relative il calcolo di entropia è stato eseguito tramite la funzione `scipy.stats.entropy`. Una volta ottenuto l'entropia per ogni tipologia di pacchetti, questi sono inseriti nella tabella del database tramite una insert la quale prende in input i valori del seriale, il valore di entropia calcolato ed il datetime relativo all'orario in cui è stato effettuato il calcolo ottenuto tramite il comando date.datetime() di python.

La funzione di calcolo dell'entropia viene eseguito ogni dieci minuti da un thread python; in questo modo il calcolo sarà automatico ed aggiornato in tempo reale. I dati ottenuti sono utilizzati per la realizzazione di una dashboard la quale,

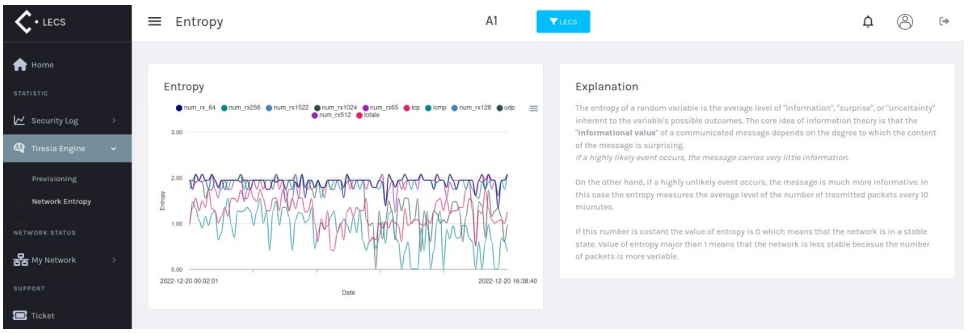

Figura 5.1: Network Entropy Dashboard

una volta estratto il valore di *fkSeriale* relativo all'utente autenticato, tramite una query genera un grafico a linee il quale riporta l'andamento dell'entropia nell'ultimo giorno per ogni tipologia di pacchetto (figura 5.1).
Poiché il calcolo sarà eseguito ogni dieci minuti il valore di entropia oscillerà tra 0 e 2 in quanto le rivelazioni per ogni LECS in quel periodo di tempo sono mediamente pari a 8/9 record. Infatti secondo la teoria di Shannon il valore massimo di entropia sarà pari al valore di *log(k)* dove k sono il numero di rilevazioni considerate. Il calcolo dell'entropia può anche essere utilizzato per rilevare eventuali anomalie di rete. Infatti il campo ***rx_ error*** del log può essere un indice per rilevare una eventuale anomalia nei pacchetti ricevuti. Il valore infatti è pari inizialmente a 0 nel caso di un comportamento normale di rete, ma incrementa se c'è un errore nei pacchetti fino a tornare stabile e fisso al valore raggiunto una volta che la rete è tornata ad un comportamento stabile. L'entropia così come è stata definita dunque può essere un indice in quanto se è presente un errore il suo valore calcolato non sarà pari a 0 come nel caso del comportamento normale ma avrà un incremento in quanto i valori di *rx_ error* non sono più stabili, ma variano tra una rilevazione e l'altra. Tale indice può essere molto utile nel caso di reti di piccola e media dimensione o nei macchinari industriali in cui gli errori nei pacchetti avviene molto raramente o nei casi di attacchi di tipo DoS. Tale statistica è stata estratta andando a monitorare il comportamento di diverse reti reali di diverse dimensioni per circa un mese. Infatti andando ad analizzare ad esempio dati relativi ai dispositivi collegati a macchinari industriali, circa il 95% della statistica di entropia *rx_ error* assume un valore pari a 0. Nel caso di reti a maggiore dimensioni invece è probabile che alcuni dei pacchetti siano errati dunque il valore di entropia di *rx_ error* risulterà comunque diverso da zero, tuttavia nonostante ciò avvenga con maggiore frequenza rispetto alle reti a minor dimensione, l'entropia calcolata in un intervallo di dieci minuti assume mediamente un valore mediamente minore di 1, mentre nel caso di attacchi tale valore avrà maggior probabilità di essere superiore ad 1. Ad esempio una rete costituita da 500 host presenta circa l'80% dei valori di entropia compresi tra 0 e 1.

5.2 K-Means Anomaly Detection

Il secondo task di analisi è quello di creare dei modelli di clustering i quali possono essere utilizzati per rilevare eventuali anomalie di rete relative ai pacchetti e ai byte scambiati al secondo. Anche questo nuovo task è una delle funzionalità definite nel modulo Network analysis & notification di figura 4.1. Uno studio [22] mostra come il numero di pacchetti e di byte al secondo possono essere utilizzati dall'algoritmo KMeans per poter rilevare anomalie di rete ed in particolare ping flood o attacchi di tipo DoS (figura 5.5). Secondo questo approccio i dati relativi al traffico di rete normale ed anomalo si distribuiscono in due cluster differenti i quali possono essere utilizzati per rilevare anomalie nei nuovi record relativi al flusso della rete. Per il rilevamento delle anomalie, è possibile utilizzare metriche come le distanze euclidee dai centroidi dei cluster della classe di traffico corrispondente. Un oggetto, infatti viene classificato come normale se è più vicino al centroide del cluster normale rispetto a quello anomalo e viceversa.

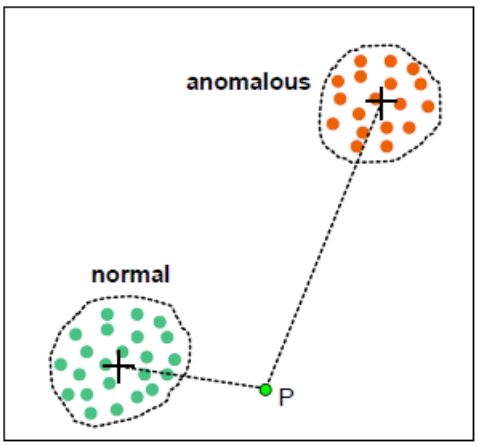

Figura 5.2: Esempio di modello di K-Means per anomaly detection [22]

Durante la tesi tale approccio è stato testato andando a controllare in real-time i nuovi dati di rete rilevati dal dispositivo e assegnandoli ad uno dei due cluster del modello. Il flowchart in figura 5.4 mostra i vari passaggi eseguiti per generare il modello di clustering:

1. Per generare i modelli di cluster utilizzati per la detection, tutti i dati relativi al traffico dell'ora precedente sono utilizzati per definire il primo dei due cluster. Non avendo a disposizione dati relativi al traffico anomalo per ogni rete analizzata, questi sono stati generati a partire dai dati a disposizione andando ad utilizzare la funzione random.uniform di python con un intervallo compreso tra 1.2 e 2.

2. Una volta generati i dati anomali i modelli per ogni utente sono addestrati: scelto un k arbitrario pari a 2 l'algoritmo Kmeans associa sia i dati reali che quelli anomali generati ad uno dei due cluster, Il modello di clustering generato può dunque essere utilizzato per rilevare eventuali nuovi

dati anomali rispetto al comportamento usuale della rete. Tuttavia un'anomalia può essere classificata in diversi modi. Per questo motivo sono state definite tre classi di anomalie:

- **Classe 3**: Il dato analizzato risulta appartenente al cluster normale, ma la sua distanza dal centroide risulta maggiore del valore di threshold (Ad esempio il punto in verde nel modello in Figura 5.3).
- **Classe 2**: Il dato analizzato appartiene al cluster anomalo e la sua distanza dal centroide del cluster normale è minore della distanza dei due centroidi (visualizzato in arancione in Figura 5.3).
- **Classe 1**: Il dato analizzato risulta appartenente al cluster anomalo, ma con una distanza rispetto al centroide del cluster normale maggiore della distanza dei due centroidi (visualizzato in rosso in Figura 5.3).

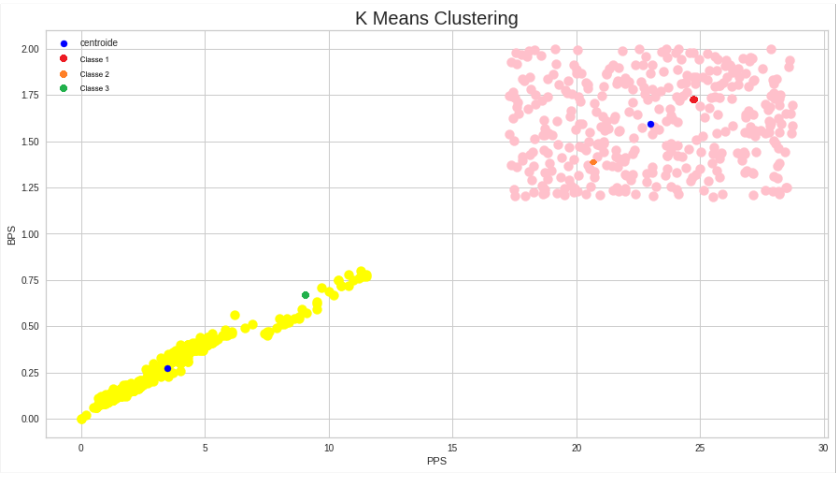

Figura 5.3: Esempio di modello di clustering generato

3. Una volta terminata la generazione dei modelli si è passati alla fase di test. Durante la fase di test si è notato che nel caso di reti a maggior dimensione e quindi più variabili è possibile che alcuni dati raccolti per la generazione dei cluster presentino già delle anomalie; portando ad una distorsione del modello generato. Per questo motivo una volta generato il modello, i dati reali utilizzati per l'addestramento sono utilizzati durante la fase di test per verificare se sono anomali in modo tale da successivamente rigenerare un modello più preciso. Infatti nel caso in cui i dati utilizzati risultino anomalie di classe 2, queste sono eliminate ed il modello è rigenerato. A

questo punto il controllo viene eseguito anche su eventuali anomalie di classe 3 le quali vengono anch'esse eliminate.

Per verificare il corretto funzionamento del sistema di rilevamento di anomalie si è simulato sia uno scan che un attacco di tipo Denial of Service (DoS) sulla rete di test a cui è collegato un LECS. Nel primo caso il sistema andrà a classificare i dati come anomalie di tipo 3 in quanto il dato non essendo troppo distante dal comportamento normale apparterrà al cluster normale, ma avrà una distanza maggiore rispetto al threshold. Nel secondo caso invece il sistema andrà a classificare i dati come anomalie di tipo 1 in quanto sarà classificato nel dato anomalo ed avrà una distanza dal comportamento normale maggiore della distanza tra i centroidi.

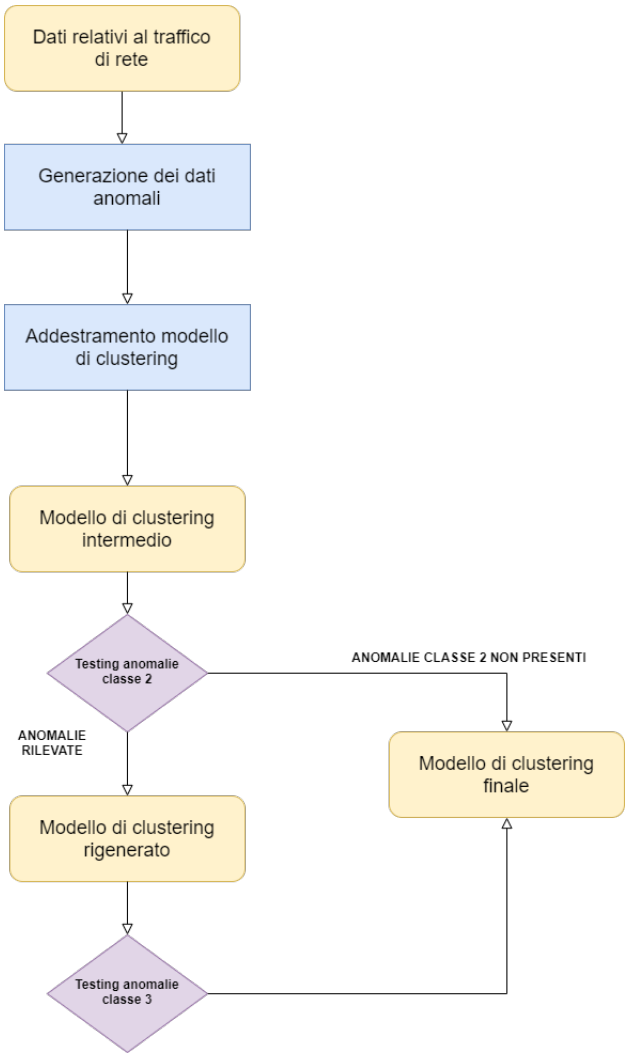

Figura 5.4: Flowchart generazione modelli di clustering

E' interessante notare che i modelli generati da questo metodo permettono di rilevare le anomalie con maggior precisione nel caso di reti di minor dimensione in quanto meno variabili durante una giornata. Per questo motivo si è deciso di

generare i modelli di clustering ogni ora in modo da confrontare il dato corrente con il modello relativo all'ora precedente. In ogni caso il task di clustering è attualmente ancora in fase di test e studio in quanto è possibile utilizzare metodi più accurati per la generazione dei cluster anomali.

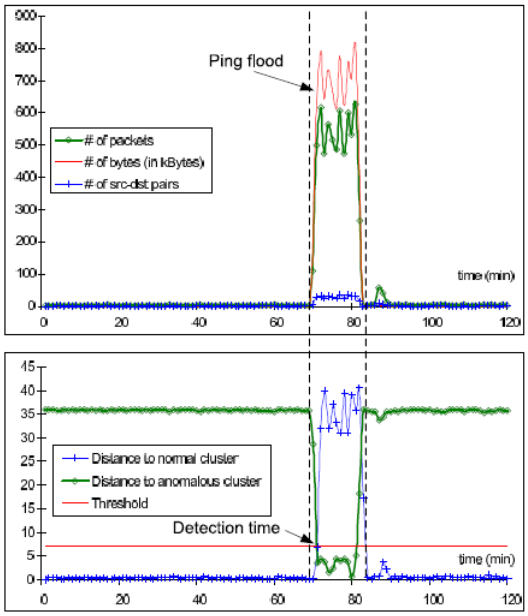

Figura 5.5: Rilevamento di ping flood con modello di clustering KMeans [22]

5.3 Log Notification

Il task di notification ha lo scopo di calcolare eventuali anomalie all'interno dell'infrastruttura monitorata da LECS e mostrare all'utente in maniera chiara e semplice l'eventuale sequenza temporale anomala rilevata. Il task rappresenta la funzionalità del modulo Temporal anlaysis and notification di figura 4.1, il quale lavora parallelamente insieme al modulo decritto nei due paragrafi precedenti. In generale i due moduli possono essere visti dall'esterno come un unico sistema di analisi e notifica. La dashboard esistente mostra all'utente in una tabella i log delle diverse classificazioni relative ad attacchi/minacce o accessi rilevati dal dispositivo i quali tuttavia non sono esemplificativi in quanto espressi in formato tabellare. Inoltre la presenza di record all'interno della tabella non è un indice di avvenuto attacco in quanto ci sono diverse classificazioni che rappresentano solo tentativi di accesso e/o esecuzione di comandi inusuali. L'anomalia infatti si presenta nel caso in cui il numero di rilevazioni di una o più particolari classificazioni dell'algoritmo di rilevamento sia superiore al numero di rilevazioni medie. Ad esempio il dispositivo controlla il numero di tentativi di accesso; nel caso in cui il numero di accessi rilevati raggruppati per ogni fascia oraria risulti molto superiore rispetto al numero medio, tale indice può essere interpretato come un tentativo di privilege escalation. Per gestire la raccolta ed elaborazione dei dati, il sistema di controllo anomalie ed il sistema di invio notifiche, è stato

sviluppato uno script python in grado di calcolare le diverse sequenze temporali e di notificare l'utente in caso di anomalie. Lo script esegue diversi passaggi descritti nei paragrafi seguenti.

5.3.1 ETL

I dati delle rilevazioni più recenti dei dispositivi LECS sono estratti, trasformati e caricati nel database. A seconda della classe di gravità con cui una minaccia è stata segnalata dal dispositivo, i dati sono salvati i tre diverse tabelle. Per ogni dispositivo sono estratti i dati delle tabelle relative alle minacce di classe 2 e 3 degli ultimi tre giorni e quelli relativi alla giornata corrente (i dati delle ore successive saranno calcolati e caricati nel database in real-time). Questi dati successivamente sono raggruppati e conteggiati per ogni classificazione e fascia oraria. Nel caso in cui durante la fase di filtraggio i dati non siano presenti, il conteggio fornirà un valore pari a 0. Una volta estratti e trasformati i dati essi saranno inseriti in una tabella. La tabella sarà composta dai seguenti campi:

Nome	**Tipo**	**Descrizione**
id	Stringa	Identificativo autoincrementale
fkSeriale	Stringa	Identificativo del dispositivo LECS
ipAttaccanti	Stringa	Lista degli IP attaccanti in quella fascia oraria
tipoAttacco	Stringa	Lista dei tipi di attacchi in quella fascia oraria
Classificazione	Stringa	Classificazione della minaccia
Date	Stringa	Data considerata
Orario	Stringa	Fascia oraria compresa tra 00 e 23
Count	Intero	Numero di volte in cui la minaccia è stata rilevata
Incrementi	Double	Incremento o decremento percentuale delle rilevazioni rispetto alla media dei giorni precedenti
Notify	Stringa	Valore stringa 'yes' o 'no' che indica se il dato deve essere notificato all'utente

Tabella 5.3: Campi della tabella increments

Prima dell'analisi vera e propria i campi incrementi e notify sono ad un valore di default pari rispettivamente a 0 e 'no'.
Anche se i dati potevano essere ottenuti direttamente utilizzando una query complessa si è deciso di trasformarli e calcolarli una sola volta e successivamente inserirli in una nuova tabella in quanto gli stessi sono poi utilizzati ogni ora. In questo modo la complessità di calcolo è minore in quanto non è necessario calcolarmi ogni volta gli aggregati, ma è sufficiente selezionarli dalla nuova tabella.

5.3.2 Detection and Notification

Ad ogni ora lo script esegue una funzione di controllo temporale. Per ogni seriale e per ogni classificazione lo script effettua un confronto tra il numero medio di rilevazioni degli ultimi tre giorni e il numero di rilevazioni attuali in quella

determinata fascia oraria calcolandone il relativo incremento/decremento percentuale ed aggiornando la tabella con il valore calcolato. Nel caso in cui il valore di incremento percentuale sia maggiore di un certo threshold, lo script invia una mail di notifica all'utente identificato dall' fkSeriale avvisando che il dispositivo ha rilevato un anomalia nell' infrastruttura monitorata. Tale avviso sarà inviato solo la prima volta nell'arco di una giornata in quanto le diverse minacce possono estendersi a più fasce orarie e troppi avvisi potrebbero essere ridondanti, inoltre in questo modo a fine giornata è possibile avere un quadro più completo dell'evoluzione della minaccia. Alla fine della giornata una funzione controlla quali utenti sono stati notificati ed invia tutte le immagini complete relative alle sequenze anomale rilevate. La figura 5.6 mostra un esempio di immagine relativa ad una sequenza anomala; come si può notare nella fascia oraria 9-10 il dispositivo ha rilevato un numero di tentativi di information leak pari a 118 il quale risulta molto maggiore rispetto alla media dei giorni precedenti, dunque l'utente sarà notificato di tale anomalia ed a fine giornata lo script invierà l'immagine della sequenza completa.

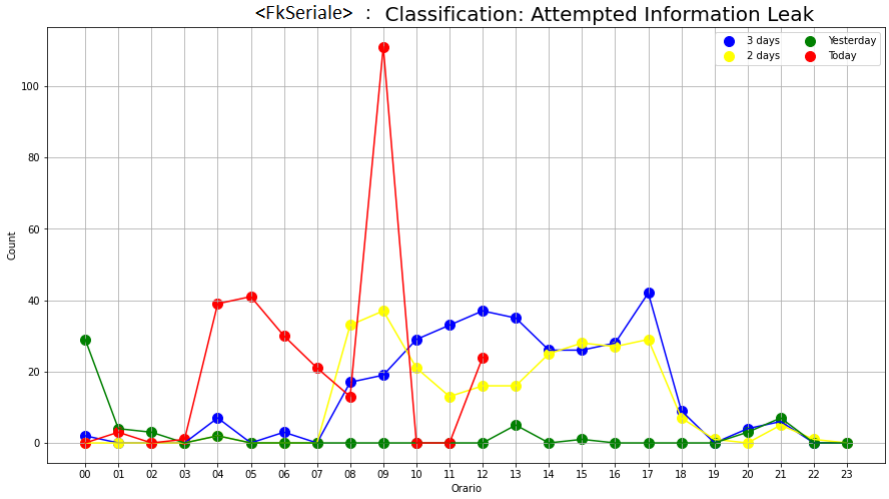

Figura 5.6: Controllo Anomalie

Capitolo 6

Log Notarization

Lo scopo del task di notarizzazione è quello di sfruttare le caratteristiche della blockchain per rendere immutabili i record relativi agli attacchi più critici, cioè quelli a maggiore impatto per l'organizzazione. Tutti gli elementi definiti in questo capitolo descrivono le funzionalità per il modulo Blockchain Service di figura 4.1. In generale la notarizzazione di un documento in blockchain consiste nel garantire l'immodificabilità dello stesso ad una certa data. Infatti una volta stabilita una data di notarizzazione, si avrà la certezza assoluta che, in quella determinata data, quel documento è diventato immodificabile e integro. Il documento notarizzato risulta anche integro in quanto è sempre possibile verificarne l'integrità; tale verifica è possibile grazie all'utilizzo delle funzioni di hash le quali permettono di associare ad ogni documento una stringa univoca di 256 bit. Nel caso in cui il documento cambi anche di un solo carattere il risultato della funzione di hash sarà diverso rispetto al valore di hash del documento notarizzato. Pertanto, quando si notarizza un documento sulla blockchain, in realtà non si invia il documento fisico, ma un suo riferimento, unico al mondo, che si esprimerà in una formula a 256 bit.

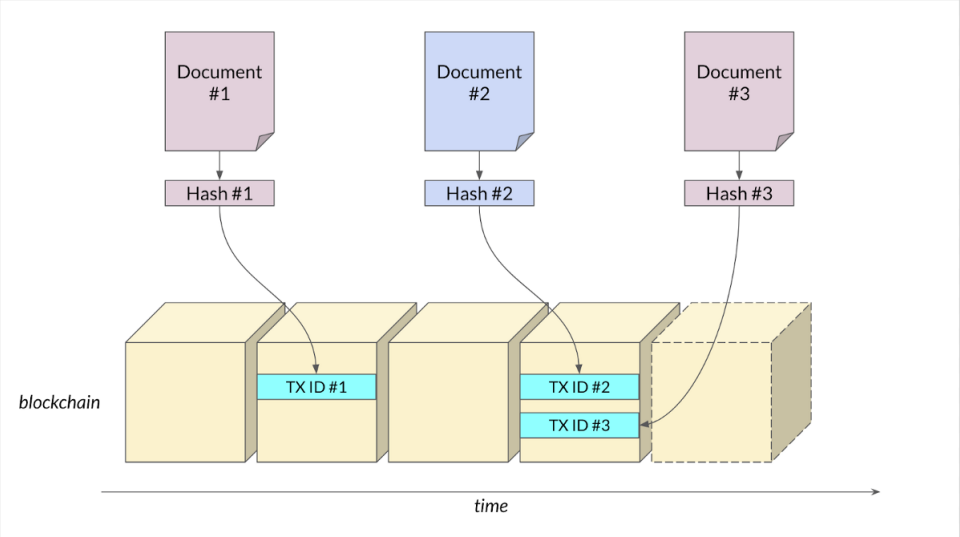

Figura 6.1: Document Notarization

I log relativi agli attacchi generati da LECS contengono diverse informazioni. Allo scopo di rendere immutabili i record degli attacchi più gravi si è deciso di andare a notarizzare solo i record relativi agli attacchi di tipo 1. I campi dei log di sicurezza sono definiti nella seguente tabella:

Campo	**Tipo**	**Descrizione**
idLog	Stringa	Identificativo incrementale del log registrato
fkSeriale	Stringa	Identificativo del dispositivo LECS che ha rilevato l'attacco
Tipo	Intero	Valore da 1 a 5 relativo alla gravità dell'attacco
Time	Stringa	Data ed ora dell'attacco
ipAttaccante	Stringa	Indirizzo IP dell'attacante
ipDestinatario	Stringa	Indirizzo IP del sistema attaccato
Opzioni	Stringa	Opzioni aggiuntive
Descrizione	Stringa	Label del tipo di attacco classificato dal dispositivo LECS
Dettaglio	Stringa	Dettagli sull'attacco

Tabella 6.1: Campi del log relativo agli attacchi

Lo sviluppo del task di notarizzazione si è svolto in diverse fasi:

- **Sviluppo Smart Contract:** Sviluppo e test locale del funzionamento dello smart contract per la notarizzazione dei log.
- **Creazione Permissioned Network:** Creazione di una blockchain di tipo permissioned e test locale sul funzionamento della rete e deploy dello smart contract.
- **Implementazione notarizzazione sul dispositivo LECS:** Test e implementazione sul dispositivo LECS (basato su ARM64) di un nodo blockchain ed integrazione notarizzazione log nel firmware.

6.1 Sviluppo dello Smart Contract

Nella prima fase si è fatto uso del linguaggio Solidity e della suite di Truffle per lo sviluppo ed il test dello smart contract, il quale si occuperà della notarizzazione dei log (La struttura del contratto è data dal diagramma delle classi in figura 6.2). All'interno dello smart contract sono definite diverse strutture dati:

- L'attributo *owner* indica il proprietario dello smart contract, il quale sarà associato una volta che lo stesso sarà deployato sulla rete.
- Un mapping *proofs* per la notarizzazione il quale associa un valore hash a un valore booleano. Infatti ogni log da notarizzare sarà trasformato in un valore di hash e inserito nel mapping con un valore pari a true, in questo modo il suo valore sarà immutabile ed è sempre possibile controllare che un determinato log sia stato notarizzato all'interno della blockchain utilizzando la funzione check_document definita nel contratto.

- Una struct contenente i campi definiti in tabella 6.1 più l'indirizzo dell'owner. Tale struttura dati permette di tenere traccia della lista di log notarizzati per ogni utente.

- Un mapping *personal_logs*, il quale associa ad ogni dispositivo LECS identificato dal seriale, un array della struct di log notarizzati. In questo modo sarà possibile in sviluppi futuri realizzare una dashboard la quale per ogni utente mostra la lista dei log notarizzati all'interno della blockchain

Per quanto riguarda invece le funzioni dello smart contract sono state definite:

- **CalculateProof** la quale dato un log, utilizza la concatenazione dei diversi campi stringa per calcolare il valore di hash del documento da notarizzare.

- **storeProof** la quale prende il valore di hash calcolato e modifica il mapping proofs settando il valore associato pari a true.

- **Check_Document** dato un documento, ne calcola il valore di hash e restituisce il valore booleano.

- **Notarize**, la funzione centrale dello smart contract la quale dato in input i campi del log generato dal dispositivo, richiamando le altre funzioni permette di notarizzare un log andando anche ad inserire il log all'interno del mapping personal_logs di un determinato utente.

- **GetPersonalLogs**, la quale dato un determinato seriale restituisce il risultato del mapping personal_log, dunque la lista di log notarizzati da un utente.

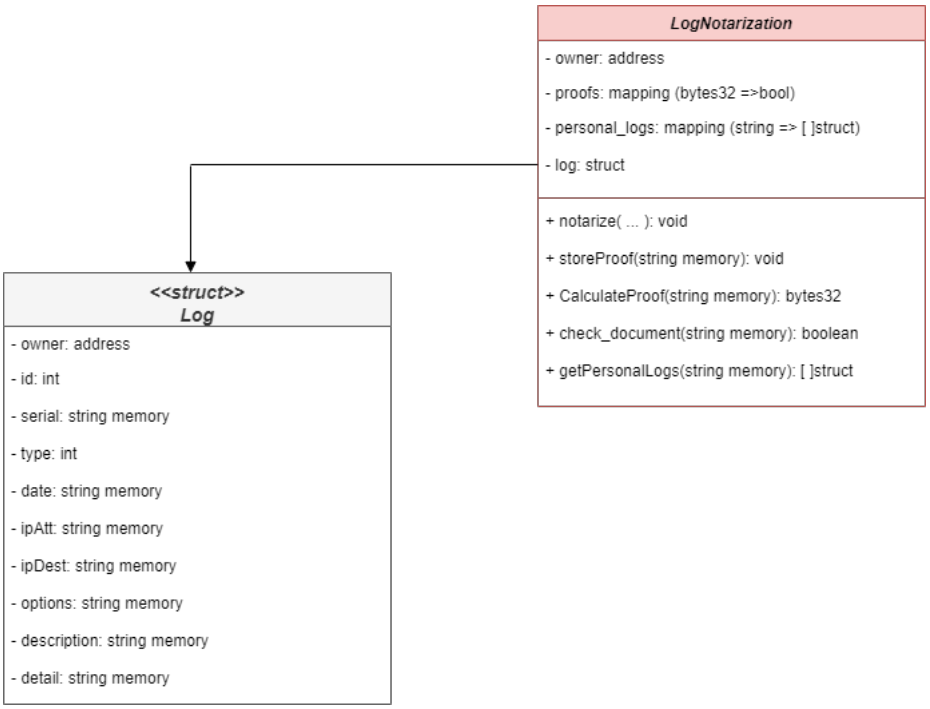

Figura 6.2: Struttura dello Smart Contract

All'interno del contratto è stato definito un modificatore il quale permette l'esecuzione delle funzioni solo al proprietario del contratto stesso (cioè di chi ha effettuato il deploy all'interno della blockchain). In questo modo eventuali attaccanti esterni per poter interagire con il contratto devono essere a conoscenza dell'indirizzo del deployer o della sua chiave privata (nel caso volesse interagire in scrittura). Una volta sviluppato il contratto si è fatto uso della suite di Truffle per effettuare un test relativo al suo corretto deployment. Tramite Ganache si è creata una blockchain personale contenente 10 account, mentre per il deploy automatico del contratto si è definito un file di configurazione `truffle-config.json` il quale specifica la versione del compilatore solc da utilizzare, la cartella del contratto e la cartella del file migrations per il deploy.

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      network_id: "*" // Match any network id
    }
  },
  compilers: {
    solc: {
      version: "^0.8.0"
    }
  }
};
```

Figura 6.3: Esempio di File di Configurazione Truffle

A questo punto si è effettuato il deploy automatico tramite il comando *truffle migrate*, il quale andrà a compilare il contratto ed ad effettuare il deploy nella blockchain locale generata da Ganache.

```
C:\Users\Utente\Desktop\notarize>truffle migrate

Compiling your contracts...
===========================
> Compiling .\contracts\Log_Notarization.solAttempt #1
> Artifacts written to C:\Users\Utente\Desktop\notarize\build\contracts
> Compiled successfully using:
   - solc: 0.8.17+commit.8df45f5f.Emscripten.clang
_ Fetching solc version list from solc-bin. Attempt #1
- Fetching solc version list from solc-bin. Attempt #1
Starting migrations...
======================
> Network name:    'ganache'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


1_initial_deploy.js
===================
_ Fetching solc version list from solc-bin. Attempt #1
   Deploying 'Log_Notarization'
   ----------------------------
   > transaction hash:    0xc46f7cee9cf183765fbfaafcad5a81b21dbf6357314a10ee0fddd2f6658d2a3a
   > Blocks: 0            Seconds: 0
   > contract address:    0x047DbD0F1C9FaF013FbEFf5133c44Dd2E622Ddd6
   > block number:        1
   > block timestamp:     1667810535
   > account:             0x973d4E628EAb29DdffB63246a49ac65cB444CDd1
   > balance:             99.97205754
   > gas used:            1397123 (0x155183)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.02794246 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.02794246 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.02794246 ETH
```

Figura 6.4: Compilazione e deploy tramite truffle

6.1.1 Test dello Smart Contract

Una volta assicurato il corretto deploy del contratto si è deciso di effettuare un test del suo funzionamento. Tra le funzionalità di Truffle infatti è possibile effettuare un test preliminare in JavaScript per verificare la corretta esecuzione di tutte le funzioni e dunque le clausole del contratto sviluppato. Il test definito nel file **testLogNotarization.test.js**, una volta effettuato il deploy utilizza la libreria web3 per poter interagire con il contratto. Le funzioni definite nello smart contract sono richiamate nel test per verificare il corretto funzionamento. I test effettuati sono diversi a partire dalla corretta notarizzazione di dispositivi LECS diversi, fino al test sul corretto revert della transazione nel caso di log già notarizzato o con un livello di sicurezza non critico. Per avviare il test è necessario salvare il file in JavaScript nella cartella test ed eseguire il comando *truffle test* nel terminale.

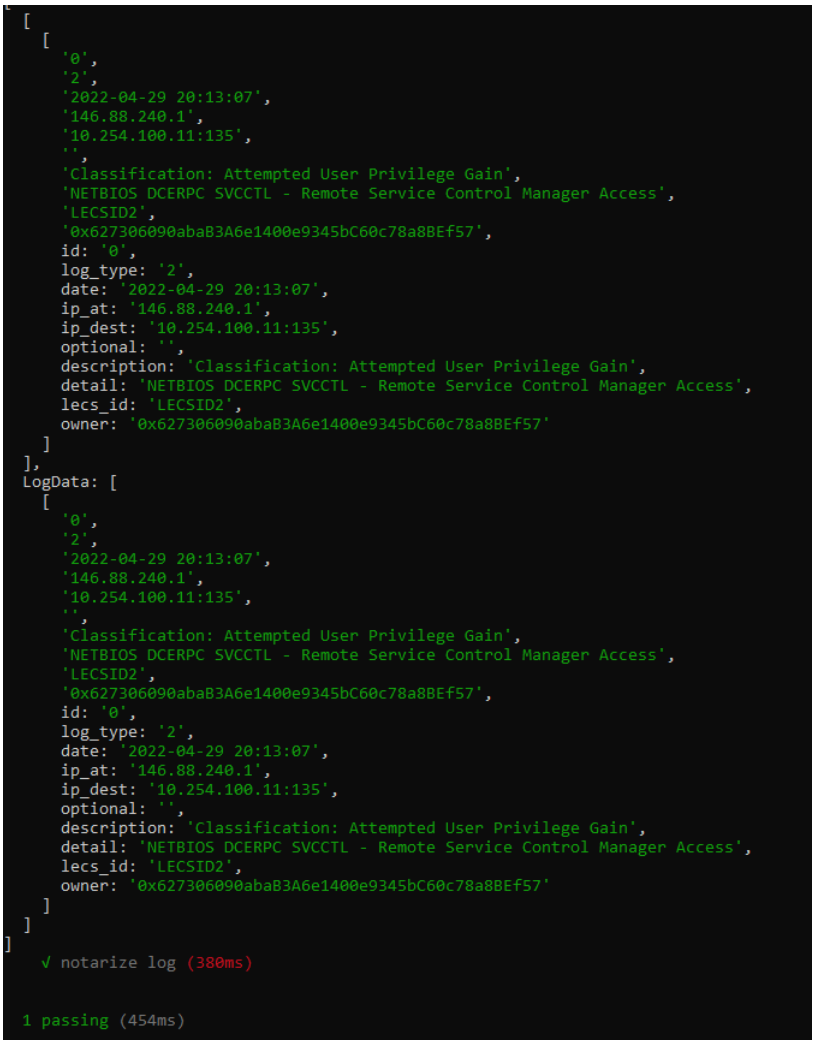

Figura 6.5: Test in JavaScript dello smart contract

6.2 Creazione Permissioned Network

Una volta terminato lo sviluppo ed il test dello smart contract, grazie all'uso di Hyperledger Besu si è creata una blockchain di tipo permissioned la quale sfrutta il protocollo di consenso QBFT (Quorum Bizantine Fault Tolerant) di tipo PoA. La rete sarà composta da 4 nodi validatori eseguiti su un server i quali andranno a validare le transazioni ed a generare i blocchi della blockchain. La rete distribuita, oltre ai 4 nodi, sarà costituita da un nodo per ogni dispositivo LECS il quale si stabilirà una connessione ai nodi del server in maniera automatica. Per semplificare la fase di discovery dei nodi e successiva connessione Besu permette di definire una lista di bootnodes, in modo tale che tutti i nodi una volta avviati si colleghino direttamente alla lista dei bootnodes senza la necessità di stabilire la connessione manualmente. L'architettura della rete P2P finale dovrebbe essere simile a quella mostrata in figura 6.19, in cui sono

presenti 4 nodi validatori collegati tra loro nel server (collegati in rosso), i quali fungono da bootnodes per gli altri nodi rappresentati dai dispositivi LECS.

6.2.1 Installazione del client

Per creare la rete privata è necessario scaricare il client besu disponibile presso la seguente repository:

`https://github.com/hyperledger/besu/releases`

Una volta selezionata la versione da utilizzare è possibile tramite il comando *curl <link-besu-client>* scaricare il client. Una volta scaricato il client ed estratto la cartella è necessario aggiungere il comando besu presente nella cartella bin nella variabile PATH d'ambiente in modo tale che lo stesso sia eseguibile in qualsiasi path. Per il funzionamento del client è necessario che il sistema abbia installato il Java Development Kit (JDK) in versione superiore alla 11, in quanto besu si basa sul linguaggio Java.

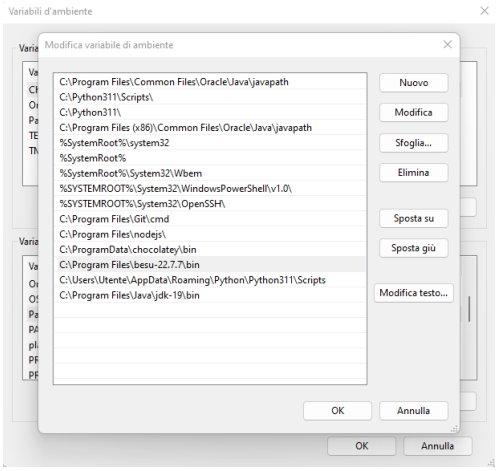

Figura 6.6: Aggiunta del comando besu alle variabili d'ambiente in sistemi Windows

6.2.2 Creazione file di genesi

Per poter creare una rete privata di tipo QBFT è necessario eseguire diversi passaggi [23]: per prima cosa è stato definito un file di configurazione *json* il quale contiene tutte le informazioni relative al blocco genesi della blockchain e sul numero di nodi da generare inizialmente. Il file contiene un campo *genesis* il quale definisce tutti le impostazioni del blocco di genesi le quali saranno salvate successivamente in un file `genesis.json`, ed un campo *blockchain* il quale specifica il numero di nodi da generare, in questo caso quattro. Una volta definito il file di configurazione è necessario eseguire il comando `besu operator generate-blockchain-config -config-file=qbftConfigFile.json -to=networkFiles -private-key-file-name=key` il quale preso in input il file di configurazione andrà a generare nella cartella networkFiles la configurazione

del blocco genesi e quattro cartelle contenti i nodi generati. Ogni nodo presenta una coppia di chiavi pubblico e private e un indirizzo [24]. Besu utilizza la coppia di chiavi per firmare e verificare le transazioni e l'indirizzo come identificativo del nodo stesso. L'indirizzo del nodo è generato creando un hash della chiave pubblica e utilizzando gli ultimi 20 byte come identificativo. Per potersi collegare ed identificare tra loro i nodi utilizzano un *enode URL* il cui formato è `enode://<id>@<host:port>` dove *id* è la chiave pubblica del nodo, escluso lo 0x iniziale, mentre *host:port* è l'host e la porta TCP su cui il nodo è in ascolto durante la fase di discovery della rete P2P. Per quanto riguarda il file del blocco genesi generato, al suo interno sono specificati diversi elementi:

- **chainID:** Le reti ethereum utilizzano due identificatori; il chain ID ed il network ID i quali nonostante spesso assumano lo stesso valore hanno due differenti usi. Le comunicazioni peer-to-peer tra i vari nodi utilizzano il network ID, mentre il processo di firma delle transazioni utlizza il chain ID. All'interno del file di genesi è possibile specificare un proprio chain ID il quale deve essere necessariamente diverso rispetto a quello standard utilizzato dalla Mainnet e dalle public testnet [25].

- **blockperiodseconds:** Poiché la rete privata utilizza un protocollo di tipo PoA è necessario che i validatori propongano un nuovo blocco anche nel caso in cui non ci siano transazioni da validare. Il campo blockperiodseconds specifica il block time, cioè il numero di secondi necessari per generare un nuovo blocco.

- **epochlength:** Numero di blocchi dopo il quale tutti i voti sono resettati.

- **requesttimeoutseconds:** Numero di secondi per ogni round di consenso prima di un cambio di round.

- **nonce:** Numero di un contatore con incremento sequenziale, che indica il numero della transazione.

- **gaslimit:** L'importo massimo di unità di gas consumabili da una transazione.

- **alloc:** Nel file di genesi è possibile definire un insieme di wallet prefounded. Il campo dunque accetta un insieme di indirizzi ed il relativo balance da allocare.

```
{
  "genesis": {
    "config": {
      "chainId": 1337,
      "berlinBlock": 0,
      "qbft": {
        "blockperiodseconds": 2,
        "epochlength": 30000,
        "requesttimeoutseconds": 4
      }
    },
    "nonce": "0x0",
    "timestamp": "0x58ee40ba",
    "gasLimit": "0x47b760",
    "difficulty": "0x1",
    "mixHash": "0x63746963616c2062797a616e74696e65206661756c7420746f6c6572616e6365",
    "coinbase": "0x0000000000000000000000000000000000000000",
    "alloc": {
      "fe3b557e8fb62b89f4916b721be55ceb828dbd73": {
        "privateKey": "8f2a55949038a9610f50fb23b5883af3b4ecb3c3bb792cbcefbd1542c692be63",
        "comment": "private key and this comment are ignored.  In a real chain, the private key should NOT be stored",
        "balance": "0xad78ebc5ac6200000"
      },
      "627306090abaB3A6e1400e9345bC60c78a8BEf57": {
        "privateKey": "c87509a1c067bbde78beb793e6fa76530b6382a4c0241e5e4a9ec0a0f44dc0d3",
        "comment": "private key and this comment are ignored.  In a real chain, the private key should NOT be stored",
        "balance": "90000000000000000000000"
      },
      "f17f52151EbEF6C7334FAD080c5704D77216b732": {
        "privateKey": "ae6ae8e5ccbfb04590405997ee2d52d2b330726137b875053c36d94e974d162f",
        "comment": "private key and this comment are ignored.  In a real chain, the private key should NOT be stored",
        "balance": "90000000000000000000000"
      }
    }
  },
  "blockchain": {
    "nodes": {
      "generate": true,
        "count": 4
    }
  }
}
```


Figura 6.7: Esempio di file di configurazione

6.2.3 Free Gas Network

In genere le transazioni in ethereum richiedono un costo computazionale il quale è misurato in gas. Il costo della transazione sarà dato dal prodotto del gas utilizzato ed il prezzo del gas. Nelle reti pubbliche, l'account che esegue una transazione paga in ether il costo del gas utilizzato ed il miner (o validatore nelle reti PoA) che inserisce la transazione in un blocco riceve il costo della transazione come ricompensa. Tuttavia, in molte reti private i partecipanti della rete gestiscono i validatori e non richiedono gas come incentivo. In questi casi dunque è necessario configurare un gas price pari a 0 in modo tale da definire una cosiddetta *free-gas network* in cui le transazioni hanno costo pari a 0. Besu offre la possibilità di definire una free-gas-network in modo tale che le reti private siano accessibili senza costi di gas [26]. Per poter definire una rete a costo 0 è necessario eseguire diverse modifiche:

1. Se il tempo di generazione dei blocchi non è un fattore critico nel file di genesi è possibile impostare il campo gasLimit al massimo accettato da Truffle (*0x1fffffffffffff*).
2. Definire nel file di genesi un campo **contractSizeLimit** il cui valore sarà il massimo supportato in byte. In questo caso il valore da impostare è pari a *2147483647*.
3. Per ogni nodo eseguito impostare l'opzione *min-gas-price* a 0.

6.2.4 Gestione del permissioning

Una rete di tipo permissioned è una rete in cui possono partecipare solo le entità autorizzate. I vantaggi di questi tipi di blockchain includono la possibilità per chiunque di unirsi alla rete dopo un adeguato processo di verifica dell'identità. Le blockchain autorizzate consentono molte funzioni, ma quella più interessante per le aziende è la **Blockchain-as-a-Service (BaaS)**, una blockchain progettata per essere scalabile per le esigenze di molte aziende. Inoltre la Blockchain-as-a-Service riduce i costi per molte aziende che possono trarre vantaggio dall'uso della tecnologia blockchain nei loro processi aziendali [27]. Hyperledger Besu permette di specificare due tipi di permissioning [28]:

1. **Node Permissioning**: L'autorizzazione avviene a livello di un nodo. Solo i nodi autorizzati possono connettersi alla rete distribuita. In questo modo è possibile gestire e controllare le connessioni di ogni nodo della rete.

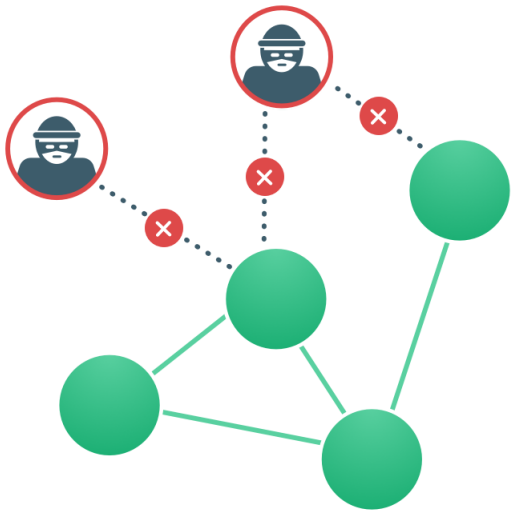

Figura 6.8: Node Permissioning

2. **Account Permissioning**; L'account permissioning permette di specificare quali account possono eseguire le transazioni all'interno della blockchain.

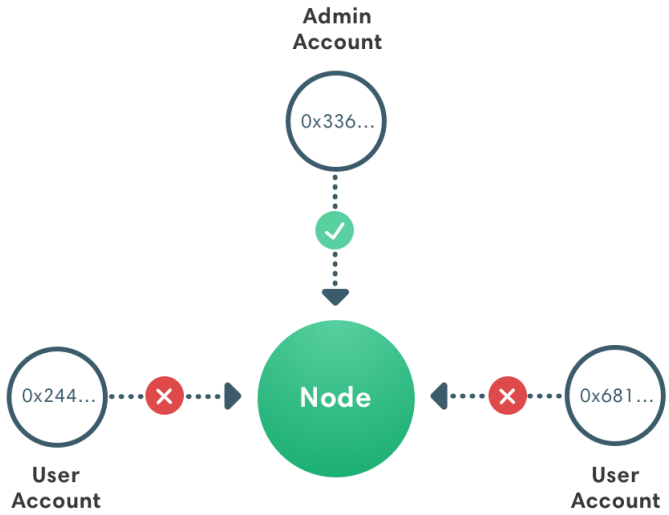

Figura 6.9: Account Permissioning

Il permissioning può essere specificato on-chain od in locale. Nel primo caso il permissioning è specificato tramite uno smart contract in esecuzione sulla rete, mentre nel secondo caso il permissioning di tipo locale lavora a livello del nodo. In questo caso il permissioning è definito per ogni nodo all'interno di un file di configurazione.

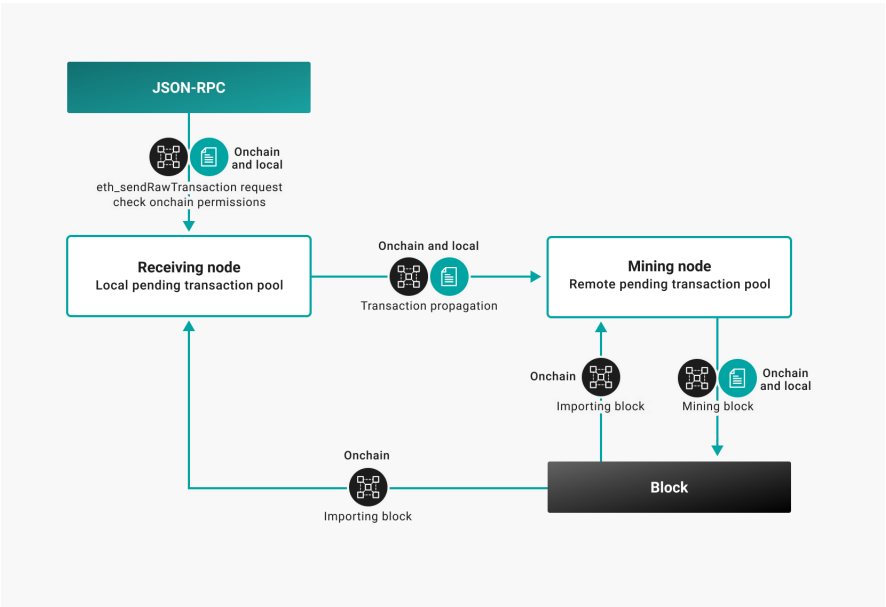

Figura 6.10: Permissioning Flow

Per avere un maggiore controllo sulle connessioni della rete P2P e degli account che possono eseguire le transazioni all'interno della blockchain si è deciso di uti-

lizzare un permissioning di tipo locale. I vantaggi sono molteplici a partire dal controllo delle connessioni tra i vari nodi della rete alla specifica degli account che possono interagire ed eseguire le transazioni all'interno della stessa. In questo caso il permissioning è stato utilizzato solo a livello di account in quanto l'autorizzazione e l'autenticazione a livello di nodo è gestita tramite l'uso di certificati (descritto nel paragrafo relativo alla connessione TLS). Utilizzando dunque un account permissionig solo l'account specificato nel file di genesi può eseguire le transazioni. Dunque grazie al permissioning c'è un controllo in profondità in quanto non solo è necessario che un eventuale attaccante conosca l'indirizzo e la chiave privata del proprietario del contratto per poter notarizzare dei log, ma quest' ultimo deve essere specificato negli account autorizzati ed il nodo deve essere autorizzato tramite certificato. Per poter specificare il permissioning locale ogni nodo della rete avrà un proprio file di permissioning il quale permette di specificare due liste:

- **nodes-allowlist**: Lista degli enode con cui è possibile stabilire una connessione P2P.

- **accounts-allowlist**: Lista degli account che possono interagire ed eseguire le transazioni all'interno della blockchain.

6.2.5 Configurazione P2P TLS

Per poter garantire sia l'autenticità che l'autorizzazione dei nodi della rete, l'architettura Besu permette di configurare connessioni TLS tra i vari nodi. In questo modo non solo i dati delle transazioni sono scambiati in maniera sicura, ma solo i nodi autorizzati possono partecipare alla rete distribuita [29]. Per poter configurare una connessione P2P è necessario che ogni nodo abbia a disposizione diversi elementi:

- Un keystore contenente il certificato e la relativa chiave del nodo.

- Un truststore contenente tutti i certificati fidati della rete con cui è possibile stabilire una connessione.

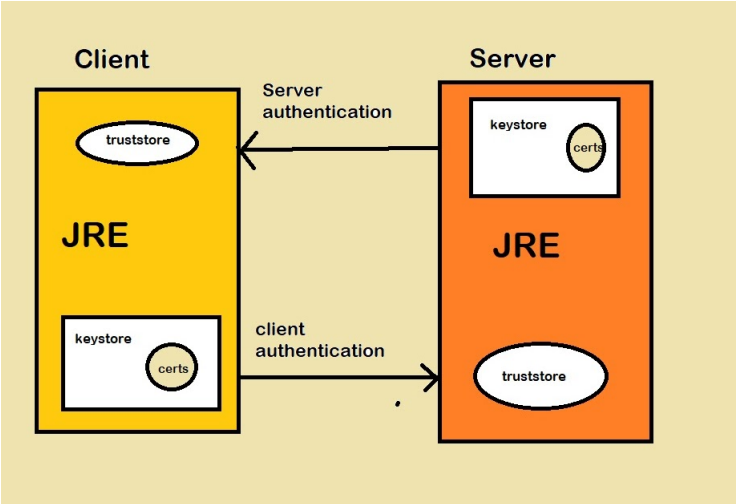

Figura 6.11: Comunicazione Client-Server tramite l'uso di truststore e keystore

Besu supporta diverse tipologie di certificati come JKS, PKCS11 e PKCS12 i quali sono utilizzati per generare i keystore ed il truststore dei nodi. La figura 6.11 mostra come un client ed un server si autentificano tra loro prima di stabilire la connessione TLS: entrambi possiedono un truststore in cui mantengono una lista di certificati validi con cui possono stabilire una connessione. Sia il client che il server si scambiano i propri certificati keystore per potersi autenticare (dunque è una mutua autenticazione); per controllare che il certificato ottenuto sia valido, questo è confrontato con il truststore: se il certificato è presente nella lista dei certificati validi allora l'autenticazione è avvenuta con successo e la connessione può essere stabilita. Poiché la rete costruita è una rete P2P non sono presenti propriamente un server ed un client, infatti i vari certificati sono gestiti in questo modo: ognuno dei 4 nodi validatori ha un proprio certificato keystore, mentre il truststore è comune a tutti e quattro i nodi. Esso contiene una lista di 5 certificati, uno per ogni nodo validatore ed uno utilizzato dai dispositivi LECS per l'autenticazione. Nel caso dei dispositivi infatti essi avranno tutti lo stesso keystore e lo stesso trustore che in questo caso contiene solamente i 4 nodi validatori del server con il quale si vuole stabilire una connessione. I vari certificati keystore ed il truststore sono stati generati tramite l'uso del tool *keytool* messo a disposizione dal Java Development Kit (JDK) [30]. Per ogni nodo sono stati eseguiti diversi passaggi descritti di seguito:

1. Il keystore del nodo è stato generato utilizzando l'opzione genkeypair.

   ```
   keytool -genkeypair -keystore mykeystore -alias alias_name -keyalg RSA -
       keysize 2048 -validity 90
   ```

   Nel comando è possibile specificare diverse impostazioni come l'alias del certificato, l'algoritmo di cifratura da utilizzare, la lunghezza della chiave e il numero di giorni in cui il certificato è valido. Una volta scelta la password per il keystore, come per ogni certificato è necessario specificare diverse informazioni come il common name (CN) cioè il nome di chi è identificato dal certificato, l'organizational unit (OU) cioè il nome dell'u-

nità dell'organizzazione, il nome dell'organizzazione (O), la località (L), lo stato (ST) ed il country code (C).

```
What is your first and last name?
  [Unknown]:  myname
What is the name of your organizational unit?
  [Unknown]:  myunit
What is the name of your organization?
  [Unknown]:  myorg
What is the name of your City or Locality?
  [Unknown]:  mycity
What is the name of your State or Province?
  [Unknown]:  mystate
What is the two-letter country code for this unit?
  [Unknown]:  IN
Is CN=myname, OU=myunit, O=myorg, L=mycity, ST=mystate, C=IN correct?
  [no]:
```

Figura 6.12: Esempio di creazione di un keystore

2. Il keystore creato può essere esaminato tramite l'opzione list, la quale mostra diverse informazioni come la data di creazione, la validità, i fingerprints del certificato e l'algoritmo utilizzato.

   ```
   keytool -list -v -keystore mykeystore
   ```

```
Your keystore contains 1 entry

Alias name: mykey
Creation date: Dec 12, 2020
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=myname, OU=myunit, O=myorg, L=mycity, ST=mystate, C=IN
Issuer: CN=myname, OU=myunit, O=myorg, L=mycity, ST=mystate, C=IN
Serial number: a6737df5ebe2c321
Valid from: Sat Dec 12 15:34:00 PST 2020 until: Fri Mar 12 15:34:00 PST 2021
Certificate fingerprints:
         SHA1: 26:CB:EB:86:A0:0D:F9:AE:17:7F:0F:C4:6D:A8:D3:CB:38:67:CA:21
         SHA256: 35:4B:35:E4:66:E9:B8:9A:F3:1A:B5:87:8A:FE:ED:71:D8:6C:F6:5F:BE:AA:C4:7E:75:4E:93:E7:1B:21:5C:46
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
```

Figura 6.13: Esempio di keystore esaminato

3. Il keystore generato viene esportato in un self-signed certificate tramite l'opzione export.

   ```
   keytool -export -keystore mykeystore -alias alias_name -file mycert.crt
   ```

4. Il certificato viene infine importato all'interno del truststore. Nel caso in cui il truststore non esista questo viene creato, altrimenti il certificato è aggiunto alla lista di certificati validi mantenuta dal truststore.

   ```
   keytool -import -keystore truststore -alias mycert -file mycert.crt
   ```

5. Tutti i certificati attualmente inseriti nel truststore possono essere esaminati tramite l'opzione list.

   ```
   keytool -list -v -keystore truststore
   ```

Tutti i vari passaggi sono stati eseguiti per ogni nodo validatore e per il futuro nodo client rappresentato dai dispositivi. In totale avremo 5 keystore e 2

truststore, il primo completo con tutti e cinque i keystore il quale sarà utilizzato dai nodi validatori, il secondo contenente solo i keystore dei nodi del server il quale sarà utilizzato dai dispositivi LECS.

6.2.6 Configurazione nodi

I nodi possono essere eseguiti specificando diverse opzioni a riga di comando. Per semplificare l'esecuzione è possibile definire un file di configurazione in formato TOML [31].

```
data-path = "data"
discovery-enabled = true
p2p-host = "<node_ip>"
p2p-port = 30303
max-peers = 500
Xdns-enabled = true
rpc-http-enabled = true
rpc-http-host = "<node_ip>"
rpc-http-port = 8545
rpc-http-api = [ "ETH", "NET", "WEB3", "ADMIN", "PERM", "QBFT",]
rpc-http-cors-origins = [ "*",]
rpc-http-max-active-connections = 65536
remote-connections-max-percentage = 100
host-allowlist = [ "*",]
permissions-nodes-config-file-enabled = true
permissions-accounts-config-file-enabled = true
genesis-file = "../genesis.json"
min-gas-price = 0
ethstats = "Server-Node:secret@<server_ip>:3000"
```

Figura 6.14: Esempio di file di configurazione

Nel file di configurazione creato sono state specificate diverse opzioni descritte di seguito:

- *data-path*: Directory in cui saranno salvati lo stato della blockchain ed il world state. La directory deve inizialmente contenere i file relativi alle chiavi del nodo generate nel passo precedente.
- *discovery-enabled*: Valore booleano che permette di ablitare o disabilitare la P2P discovery.
- *p2p-host*: Host pubblico utilizzato per accedere al nodo dall'esterno della rete nella comunicazione P2P. Di default il suo valore è pari a 127.0.0.1 cioè al localhost.
- *p2p-port*: Porte UDP e TCP in ascolto. Di default la porta di ascolto di un nodo è la 30303.
- *max-peers*: Numero massimo di peers che si possono connettere al nodo.
- *rpc-http-enabled*: Abilita o disabilita il servizio HTTP JSON-RPC. L'impostazione predefinita è false.

- *rpc-http-host*: L'host su cui HTTP JSON-RPC è in ascolto. L'impostazione predefinita è 127.0.0.1. Per limitare l'uso dell'API dall'esterno l'impostazione è lasciata predefinita.
- *rpc-http-port*: La porta TCP su cui il servizio HTTP JSON-RPC è in ascolto. L'impostazione predefinita è 8545.
- *rpc-http-api*: Elenco di API da abilitare sul canale HTTP JSON-RPC. L'impostazione predefinita è: ETH, NET, WEB3 a cui saranno aggiunte ADMIN, PERM, QBFT.
- *rpc-http-cors-origins*: Lista di domini che possono accedere al nodo utilizzando JSON-RPC.
- *rpc-http-max-active-connections*: Il numero massimo di connessioni HTTP JSON-RPC consentite. Una volta raggiunto questo limite, le connessioni in arrivo vengono rifiutate.
- *bootnodes*: Lista di indirizzi enode a cui collegarsi in fase di discovery.
- *remote-connections-max-percentage*: Percentuale di connessioni P2P remote che è possibile stabilire con il nodo.
- *host-allowlist*: Elenco di host che possono accedere all'API JSON-RPC.
- *permissions-nodes-config-file-enabled*: Valore booleano che abilita o disabilita il permissioning al livello di nodo.
- *permissions-accounts-config-file-enabled*: Valore booleano che abilita o disabilita il permissioning al livello di account.
- *genesis-file*: Path relativo al file del blocco genesi.
- *min-gas-price*: Valore minimo di gas per eseguire una transazione. In questo caso sarà posto a 0 in quanto la rete sarà una free-gas network.
- *Xp2p-tls-enabled*: Valore booleano che abilita o disabilita l'uso di TLS per la connessione tra i nodi.
- *Xp2p-tls-keystore-type*: Tipo di certificato utilizzato per il keystore.
- *Xp2p-tls-keystore-file*: Path relativo al keystore file del nodo.
- *Xp2p-tls-keystore-password-file*: Path relativo al file testuale contenente la password per sbloccare il keystore.
- *Xp2p-tls-truststore-type*: Tipo di certificato utilizzato per il truststore.
- *Xp2p-tls-truststore-file*: Path relativo al trustsotre file utilizzato dal nodo.
- *Xp2p-tls-truststore-password-file*: Path relativo al file testuale contenente la password per sbloccare il truststore.

- *tx-pool-retention-hours:* Il numero massimo di ore in cui una transazione in pending è mantenuta all'interno del transaction pool.

- *tx-pool-future-max-by-account:* Numero massimo di transazioni mantenute dal transaction pool per nodo.

- *ethstat*: Url del server ethstat per il monitoraggio del nodo.

6.2.7 Esecuzione dei nodi

La struttura di un nodo server è riassunta nella figura 6.15. Per ogni nodo avremo sia le sue chiavi pubbliche e private sia i rispettivi certificati ed i file di configurazione delle opzioni e del permissioning. Per quanto riguarda il truststore questo è comune a tutti i nodi del server e quindi è inserito nel file system allo stesso livello del file di genesi. Il nodo una volta eseguito tenterà di stabilire una connessione TLS con i nodi specificati nella lista del campo bootnodes.

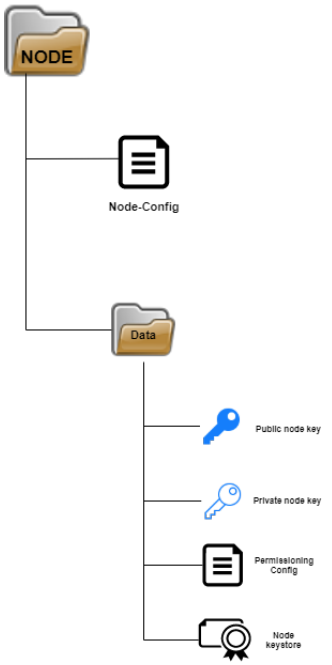

Figura 6.15: File System di un nodo Besu

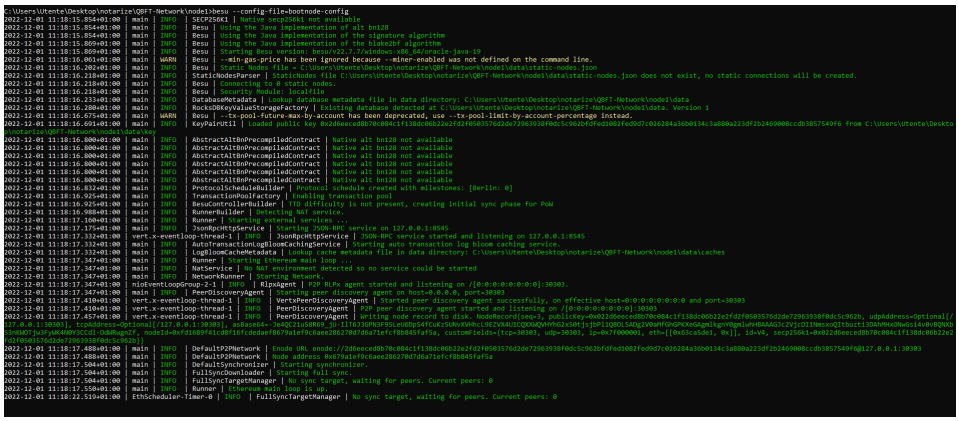

Figura 6.16: Esempio di esecuzione di un nodo Besu

Per poter eseguire un nodo è sufficiente per ognuno eseguire dalla riga di comando `besu -config-file=node-config`. Il nodo una volta eseguito tenterà di stabilire una connessione TLS con i nodi specificati nella lista del campo bootnodes.

6.2.8 Deploy dello Smart Contract

Poiché Besu non prevede la gestione interna delle chiavi private, per poter effettuare il deploy automatico dello smart contract tramite l'utilizzo della suite di Truffle è necessario installare una libreria javascript in grado di configurare un wallet [32]. La libreria è stata installata tramite npm eseguendo il seguente comando: `npm install -save @truffle/hdwallet-provider`. Una volta installato il gestore dei portafogli è necessario modificare il file di configurazione di truffle andando a creare un wallet in grado di interagire con la blockchain privata in esecuzione. All'interno del file di configurazione viene definito un provider di chiavi private il quale permette di importare l'account che si occuperà del deploy all'interno della blockchain andando a specificare la sua chiave privata (in questo caso l'account importato è quello del file di genesi). Il provider viene utilizzato nella definizione di una network la quale si connetterà ad un nodo validatore andando a specificare l'host e la porta in cui il servizio HTTP RPC è attivo. Poichè la blockchain è una free gas network nella definizione della rete si sono specificati un gasprice pari a 0 ed un gaslimit pari al valore massimo supportato. Per poter effettuare il deploy del contratto nella blockchain privata è necessario eseguire il comando `truffle migrate -network besuWallet`. Il contratto deployato e le sue funzionalità saranno inserite come una transazione in un nuovo blocco allo scadere del blocktimeperiod; inoltre il costo del deploy sarà pari a 0 in quanto si è impostata una free-gas network.

```
const PrivateKeyProvider = require("@truffle/hdwallet-provider");
const privateKey = ["<private_account_key>"];
const privateKeyProvider = new PrivateKeyProvider(privateKey, "<node_ip>");
module.exports = {
  migrations_directory: "./migrations",
  contracts_directory: './contracts/',
    networks: {
      besuWallet: {
        host:  "<node_ip>",
        port: "8545",
        provider: privateKeyProvider,
        network_id: "*",
        gasPrice: 0,
        gas: "0x1fffffffffffe"
      },
    },

    gui: {
      host: "localhost",
      port: 5000,
      network_id: "*" // Match any network id
    },

  compilers: {
    solc: {
      version: "^0.8.17"
    }
  }
}
```

Figura 6.17: Esempio di configurazione Truffle per deployment su architettura Besu

6.2.9 Monitoraggio nodi

I nodi collegati alla blockchain sono monitorati tramite l'uso dello strumento *ethstats-server* il quale mostra in una dashboard (figura 6.18) diverse informazioni. Per poter inviare i dati al server è necessario scaricare la repository presso `https://github.com/goerli/ethstats-server`. Una volta scaricata la repository è necessario definire un file `ws_secret.json` il quale salva un ws secret che sarà successivamente specificato da ogni nodo durante la connessione. Una volta definito ws secret ed impostato la repository, la dashboard può essere attivata tramite il comando `npm start`. Per connettere un nodo besu all'ethstatserver è sufficente aggiungere al file di configurazione del nodo l'opzione `ethstats="<node-name>:<jws-secret>@<ip-address>"` dove *node-name* è il nome del nodo visualizzato nella dashboard, *jws-secret* è la password necessaria affinchè il nodo si colleghi al ethserver e *ip-address* è l'indirizzo IP e relativa porta dove il server è in esecuzione. Per identificare in maniera univoca ogni dispositivo il nome del nodo visualizzato nella dashboard sarà costituito dal nome LECS-Node seguito dagli ultimi 4 caratteri del seriale identificativo del dispositivo. La dasboard mostra diverse informazioni come il numero di nodi attivi, il gas price e gas limit, il numero di blocchi all'interno della blockchain e il tempo medio di generazione dei blocchi. Per ogni nodo che si è connesso la dashboard mostra se il nodo è online ed ulteriori informazioni come il nome, il tipo (il quale mostra sia il sistema in cui il nodo è in esecuzione sia la versione di besu

e jdk utilizzata), il numero di peers a cui è connesso, il numero di transazioni in attesa di essere validate,l'ultimo blocco presente nel nodo ed altre informazioni utili. La dashboard dunque permette di visualizzare in real-time lo stato della blockchain e la visualizzazione di tutti i nodi che si sono collegati.

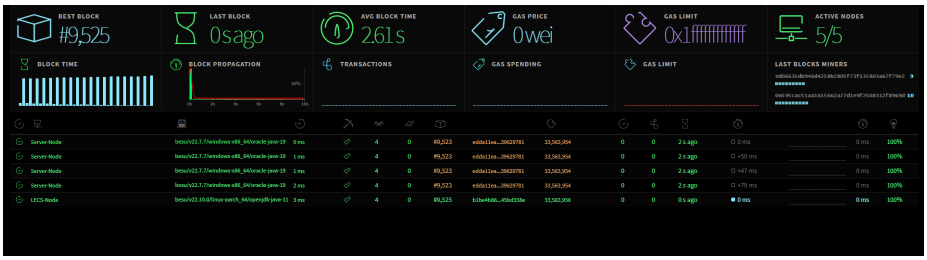

Figura 6.18: Esempio di Dashboard Ethstats-server

6.2.10 Avvio del Server

Tutti i passaggi descritti nei paragrafi precedenti sono stati testati in ambiente Windows. Una volta testata la correttezza ed il funzionamento del sistema, si è deciso di isolarlo maggiormente andando ad implementare una macchina virtuale Linux. Una volta installato il client Besu, generato i nodi con le relative certificazioni, si è deciso per comodità di sviluppare uno script python il quale permettesse di configurare le porte, le opzioni di ogni nodo e la relativa esecuzione in maniera semplice ed automatica. In questo modo nel caso in cui il server dovesse non essere disponibile, per il successivo riavvio dei servizi blockchain è sufficente eseguire lo script. Lo script python sfrutta la libreria TOML per la manipolazione dei file di configurazione e la libreria subprocess per eseguire in maniera automatica i 4 nodi validatori. Una volta eseguiti i nodi tramite il comando truffle migrate lo smart contract sviluppato è stato deployato all'interno della rete. Tutte le informazioni relative all'indirizzo e il valore dell'ABI del contratto sono memorizzati nel file `Log_Notarization.json` il quale sarà utilizzato dai dispositivi per interagire con lo smart contract.

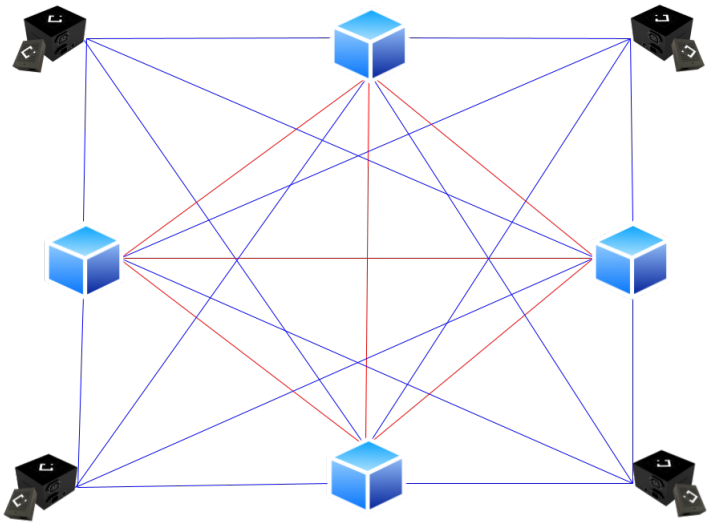

Figura 6.19: Architettura della Permissioned Blockchain

6.3 Implementazione sul dispositivo LECS

La configurazione di un nodo LECS avviene in maniera simile alla configurazione dei nodi server. Tramite l'utilizzo del protocollo SSH una volta effettuati i vari passaggi di installazione del client besu, si sono definiti dei servizi linux in modo tale che il nodo LECS creato si colleghi alla blockchain ogni volta che il dispositivo sia acceso. Tale caratteristica è fondamentale in quanto una delle peculiarità del dispositivo è il fatto di essere plug-and-play, dunque il nodo deve essere configurato e collegato in maniera automatica alla blockchain una volta collegato allo switch di rete. In particolare sono stati definiti due servizi:

- **Configure Chain:** Per poter evitare l'eventuale problema di clonazione di uno stesso nodo, il dispositivo ogni volta che viene acceso ne crea uno nuovo. Per questo motivo tutti i file di configurazione del nodo ed i certificati sono inseriti in una cartella node_files. All'avvio del dispositivo il servizio sviluppato esegue due script python. Il primo definito nell' ExecPreStart cancella il vecchio nodo ed esegue tutte le operazioni di creazione del nuovo, mentre il secondo genera il file di configurazione del nodo creato specificando tutte le diverse opzioni.

```
GNU nano 6.2                                    configurechain.service *
[Unit]
Description=Configure Blockchain
DefaultDependencies=no
After=systemd-network-wait-online.service
Wants=network-wait-online.service
[Service]
User=pi
Type=simple
ExecStartPre=/usr/bin/python /home/pi/blockchain/create_node.py
ExecStart=/usr/bin/python /home/pi/blockchain/configure.py
KillMode=process
KillSignal=SIGINT
TimeoutStopSec=90
Restart=on-failure
[Install]
WantedBy=multi-user.target
```

Figura 6.20: ConfigureChain.service

- **Start Node:** Il servizio viene avviato dopo che configure chain ha terminato la propria esecuzione e lancia il comando *besu –config-file=node-config* il quale avvia il nodo.

```
GNU nano 6.2                                    start_node.service *
[Unit]
Description=Blockchain Node Start
DefaultDependencies=no
Require=configurechain.service
After=syslog.target network.target
[Service]
User=pi
Type=simple
ExecStart=bash /home/pi/besu/bin/besu --config-file=/home/pi/blockchain/node/node-config
KillMode=process
KillSignal=SINGINT
TimeoutStopSec=90
Restart=on-success
RestartSec=5s
[Install]
WantedBy=multi-user.target
```

Figura 6.21: StartNode.service

6.3.1 Integrazione Notarizzazione con il Firmware LECS

Una volta eseguito i vari test di creazione dei nodi e connessione alla blockchain in modalità plug and play dei dispositivi, è necessario che il sistema di notarizzazione e dunque l'esecuzione delle funzionalità dello smart contract siano integrate con il sistema di rilevamento e classificazione delle minacce implementato nel firmware del dispositivo. Per poter sfruttare la libreria web3.js l'esecuzione della transazione è stata definita in un file JavaScript `notarize.js`. Per poter interagire con lo smart contract nella permissioned blockchain è necessario installare le librerie *ethereumjs-tx* e *ethereumjs-common* tramite npm e successivamente richiamare la funzione *sendSignedTransaction* messa a disposizione da web3. La figura 6.22 mostra un esempio di transazione:

1. Tramite la libreria web3 si definisce un provider e ci si connette al nodo della blockchain.

2. Utilizzando l'indirizzo e l'ABI del contratto si definisce un'istanza del contratto e si richiama la funzione con la quale si vuole interagire e la si salva in una variabile data.

3. Tramite la libreria ethereum-common viene definita la struttura custom della blockchain andando a specificare il network ed il chain ID.

4. La libreria ethereum-tx permette di definire una variabile transazione specificando il nonce, il prezzo ed il limite del gas, l'indirizzo del contratto e la custom chain definita nel passo precedente.

5. Utilizzando la chiave privata dell'account autorizzato la transazione viene firmata e serializzata tramite rispettivamente la funzione *sign* e *serialize*.

6. La transazione una volta firmata e serializzata viene inviata al servizio blockchain tramite la *sendSignedTransaction* per poter essere successivamente validata ed inserita in un nuovo blocco.

```
const Web3 = require('web3');
const Tx= require("ethereumjs-tx").Transaction
const common = require('ethereumjs-common');
const MyContract= require('./build/contracts/Log_Notarization.json')
var privateKey = Buffer.from("<private-key>", 'hex')

function notarize(argument){
    const web3 = new Web3(Web3.givenProvider || "<node-ip>")
    contract_address= MyContract.networks["<nodeid>"].address
    console.log(contract_address)
    chain_id=web3.eth.getChainId().then(console.log)
    const contractInstance = new web3.eth.Contract(MyContract.abi, contract_address);
    web3.eth.getTransactionCount('<user_address>', 'pending',function (err, nonce) {
    var data = contractInstance.methods.contractfunction(argument).encodeABI();
    const chain = common.default.forCustomChain('mainnet', {
       name: 'custom-chain',
       networkId: "<nodeid>",
       chainId: "<nodeid>"
     }, 'petersburg');
    var tx = new Tx({
       nonce: web3.utils.toHex(nonce),
       gasPrice:  0,
       gasLimit:  "0x1fffffffffffff",
       to: contract_address,
       value: web3.utils.toHex(0),
       data: data,
      }, {common: chain} );
    tx.sign(privateKey, 'hex');
    var raw = tx.serialize().toString('hex');
    web3.eth.sendSignedTransaction('0x' + raw)
    })
}
```

Figura 6.22: Esempio di funzione per l'interazione con lo smart contract

Una volta definito il file relativo alla notarizzazione questo viene compilato tramite il comando bytenode; in questo modo il codice salvato all'interno del dispositivo è in bytecode e dunque non visualizzabile. Una volta definita l'iterazione in JavaScript si è passati all'effettiva integrazione con il firmware. All'interno del

firmware si è definita una funzione la quale presi in input tutti campi necessari per la notarizzazione, questi tramite la libreria subprocess sono utilizzati come parametri durante l'esecuzione dello script `notarize.jsc`. La funzione viene richiamata all'interno della funzione di classificazione del firmware la quale è divisa per gravità della minaccia rilevata. Dunque la funzione relativa alle minacce di livello 1 definisce un thread python nel quale eseguire la funzione di notarizzazione.

Capitolo 7

Considerazioni di sicurezza del sistema di notarizzazione

L'uso della blockchain come database distribuito per la notarizzazione dei log offre diversi vantaggi in quanto ci si assicura che i log registrati siano *immutabili* ed *integri*. Lo sviluppo del task di notarizzazione oltre all'immutabilità e integrità dei dati permette di soddisfare altri requisiti:

- **Disponibilità**: I nodi validatori nel server possono essere aumentati in modo tale che anche se uno di essi risulti offline il sistema continui ad eseguire il task di notarizzazione.
- **Anonimato**: Ogni utente è anonimo in quanto ogni nodo all'interno della blockchain è identificato da una chiave pubblica e dagli ultimi 4 caratteri del dispositivo LECS.
- **Autenticità**: Ogni nodo è autenticato grazie all'uso di certificati durante la fase di handshake del protocollo TLS.
- **Autorizzazione**: Una volta autenticato l'utente è autorizzato a notarizzare i log di sicurezza solo nel caso in cui sia a conoscenza sia dell'indirizzo dell' account allocato nel file di genesi, sia dell'indirizzo dello smart contract deployato.
- **Confidenzialità**: La confidenzialità tra i dati scambiati dai nodi è assicurata grazie all'uso di canali cifrati. TLS utilizza una combinazione di crittografia simmetrica e asimmetrica per garantire la privacy dei messaggi. Durante l'handshake i nodi concordano un algoritmo di crittografia e una chiave segreta condivisa da utilizzare durante la sessione. Tutti i messaggi trasmessi dunque sono cifrati, garantendo che il messaggio rimanga privato anche se intercettato.

Inoltre l'uso di una macchina virtuale per la definizione e l'esecuzione dei nodi server permette di applicare la tecnica di isolamento. In questo modo è più complicato interagire con la macchina virtuale in quanto l'hardware è interamente virtualizzato.

7.1 Possibili attacchi

Per quanto riguarda eventuali possibili attacchi si sono considerate diverse possibilità: un eventuale utente malevolo che sia riuscito a collegari alla blockchain potrebbe voler cercare di notarizzare diversi log falsi in maniera continua in modo da rallentare il sistema di notarizzazione ed eventualmente far scartare dal pool di attesa le transazioni relative ai log legittimi. Per mitigare tale attacco ogni nodo presenta l'opzione `tx-pool-retention-hours` che imposta il numero massimo di ore in cui una determinata transazione può rimanere all'interno del pool di attesa prima di essere scartata, e l'opzione `tx-pool-future-max-by-account` con un valore pari a 0.05 il quale assicura che ogni nodo possa inserire al massimo 15 transazioni tra un blocco e un altro. Inoltre la probabilità che un determinato attaccante riesca a collegarsi alla blockchain e riesca a notarizzare un log è molto bassa in quanto deve essere a conoscenza di diverse informazioni:

- Per potersi collegare alla rete è necessario che il nodo malevolo possieda l'indirizzo enode dei nodi del server e successivamente presenti un certificato valido definito all'interno del truststore.
- Per potersi inserire nella blockchain inoltre l'attaccante deve avere a disposizione l'identico file di genesi con lo stesso chain ID.
- Una volta collegato l'attaccante per poter notarizzare log falsi dovrebbe essere a conoscenza sia dell'account autorizzato alla notarizzazione impostato a livello di permissioning della rete ed a livello di modificatore dello smart contract, sia dell'indirizzo e valore di ABI relativo allo smart contract deployato nella blockchain.

Un attaccante per ovviare a tali operazioni dovrebbe fisicamente ottenere un LECS e tentare o di clonare la scheda del dispositivo oppure, essendo il dispositivo una black box chiusa, eseguire reverse engineering per ottenere i dati relativi al nodo. Inoltre nel caso di clonazione della scheda del dispositivo, grazie ai servizi linux sviluppati, il nodo non risulterà clonato in quanto ne verrà creato uno nuovo ogni volta (in questo modo si evitano conflitti nel caso di nodi identici). Inoltre una volta collegato l'attaccante può essere rilevato all'interno della dashboard anche nel caso in cui questo non si colleghi ad essa. La dashboard infatti può essere uno strumento utile non solo per visualizzare lo stato corretto, ma anche per individuare eventuali intrusi in quanto un numero di peers a cui i nodi del server sono connessi risulti maggiore di n+3, dove n è il numero di dispositivi attualmente connessi, indica che un nodo esterno malevolo non connesso alla dashboard si sia collegato alla blockchain. Tale nodo può essere facilmente identificato tramite l'uso delle API le quali permettono per ogni nodo di ottenere la lista dei peer a cui attualmente è connesso. Una volta identificato il nodo malevolo può essere facilmente inserito in una lista di nodi esclusi dalla blockchain.

Capitolo 8

Conclusioni e Sviluppi Futuri

In conclusione il funzionamento dei diversi task svolti durante la tesi è stato testato in maniera corretta. I diversi task di analisi dei dati, in sinergia con il sistema di notarizzazione rappresentano delle funzionalità aggiuntive dell'intera architettura di sistema descritta nel capitolo 4. Alcuni dei diversi task sviluppati sono ancora in fase di test e sviluppo, mentre altri come ad esempio il calcolo dell'entropia e notarizzazione dei log sono entrati rispettivamente nella fase di produzione e pre-produzione. Nei prossimi paragrafi sono descritti dei possibili sviluppi futuri per alcuni dei task sviluppati.

8.1 Sistema di Notifica

Per quanto riguarda il modello di anomalie temporali i test sono eseguiti in modo tale da impostare un livello di threshold relativo all'incremento percentuale il più corretto possibile, cioè in grado di non generare troppe notifiche all'utente, ma allo stesso tempo di individuare le possibili minacce. Inoltre tale livello percentuale deve essere impostato in maniera differente a seconda del valore della media dei giorni precedenti. Infatti una media con valore minore di 1 può portare ad una generazione eccessiva di notifiche anche quando l'incremento effettivo è molto basso, in questi casi è sufficiente aumentare il valore di threshold in modo tale da avere una corretta gestione dell'anomalia. Un altro elemento che può variare il valore di threshold da considerare è il tipo di classificazione la quale può avere un diverso impatto. Ad esempio per quanto riguarda una classificazione del tipo *Corporate Violation Policy* poiché solitamente il valore è sempre pari a 0, è sufficiente che il sistema rilevi uno singolo scostamento per notificare immediatamente l'utente. Dato l'elevato numero di classificazioni è necessario eseguire dei test con dati reali per calibrare in maniera corretta i diversi threshold da gestire. Il sistema di notifica delle sequenze temporali agisce in maniera parallela al sistema di notifica delle anomalie di rete descritte nei paragrafi 5.1 e 5.2 in modo tale che i due sistemi siano visti dall'esterno come un unico sistema di notifica utente. Inoltre tali sistemi di notifica nel caso di attacchi gravi saranno integrati con un chatbot telegram attualmente in sviluppo.

8.2 Sistema di Notarizzazione

Per poter alleggerire il carico di esecuzione dei nodi ed aumentare il livello di isolamento andando a virtualizzare a livello di sistema operativo e dunque a livello di ABI (Application Binary Interface) è possibile utilizzare un approccio orientato all'uso dei container. Un container è un'unità software che racchiude codice e tutte le sue dipendenze, affinché l'applicazione possa essere eseguita nella stessa maniera in ambienti computazionali diversi. Un container dunque è una sorta di piccola macchina virtuale che gira sulla base di un sistema operativo che lo ospita. È un concetto noto da tanti anni ma negli ultimi si è rivelato particolarmente comodo poiché tutte le applicazioni che ospitano micro-servizi sono depoyati come container. Il vantaggio dei container è utilizza solo le librerie strettamente necessarie per far funzionare il micro sistema operativo e di conseguenza il footprint di un'immagine o di un container ha un basso contento di storage (solitamente nell'ordine dei MB). Quando si va a creare un container, lo si crea partendo da un'immagine che altro non è che l'insieme delle dipendenze e del software utili per l'esecuzione della mia applicazione. Facendo un' analogia con la programmazione object-oriented un container è come se fosse un oggetto, viceversa un'immagine è come se fosse la classe, quindi nella classe si ha tutto quello che serve, mentre l'oggetto è semplicemente l'istanza. A differenza delle macchine virtuali, tutti i conteiner di un sistema condividono lo stesso kernel ed eseguono le istruzioni in modo nativo (senza emulazione). Ogni conteiner contiene le librerie e le applicazioni necessarie per l'esecuzione dei programmi in esso contenuti, in questo modo i vari container di sistema sono isolati l'uno dall'altro.

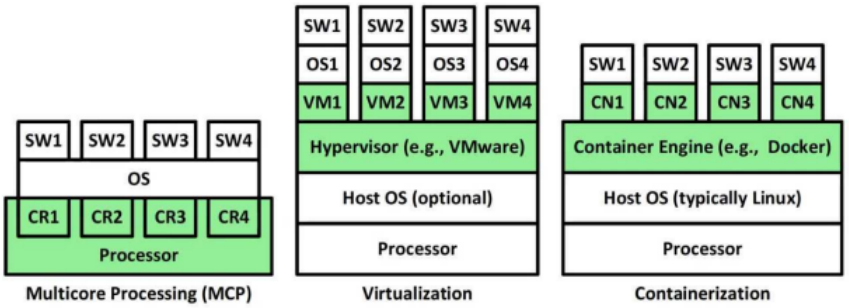

Figura 8.1: Differenza tra macchina virtuale e container

Utilizzando tale approccio è possibile utilizzare Docker, un insieme di prodotti PaaS (Platform as as Service) che permettono lo sviluppo e il delivery di software in container. Per poter utilizzare Besu e tutte le sue dipendenze è sufficiente utilizzare l'immagine messa a disposizione da Docker Hub ed esporre in maniera corretta le porte TCP per la comunicazione e la connessione della rete P2P. Tale approccio inoltre presenta molteplici vantaggi in quanto l'esecuzione di ogni nodo è isolata l'uno dall'altra e il riavvio del server è semplificato in quanto è sufficiente rieseguire i container. In conclusione l'approccio orientato ai container risulta un vantaggio in questo caso, in quanto ogni nodo è esposto come micro-servizio e risulta maggiormente isolato.

Bibliografia

[1] Veronica Balocco. Cybersecurity, in italia attacchi hacker a livelli mai visti. emergenza island hopping. *Corriere Comunicazioni*, 2020.

[2] Gabriele Faggioli. Crescono gli attacchi cyber in italia, ma anche le difese: ecco il quadro. *Agenda Digitale*, 2022.

[3] Microsoft Security Team. What is a siem. https://www.microsoft.com/it-it/security/business/security-101/what-is-siem, 2022.

[4] Cyber Evolution Srl. Last cyber security solution. https://lecs.io/, 2022.

[5] Wikipedia contributors. Entropy (information theory). https://en.wikipedia.org/wiki/Entropy_(information_theory), 2022.

[6] Claude Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 1948.

[7] Introduzione agli smart contracts. https://ethereum.org/it/developers/docs/smart-contracts/, 2022.

[8] Che cos'è uno smart contract? https://www.coinbase.com/it/learn/crypto-basics/what-is-a-smart-contract, 2022.

[9] Wikipedia contributors. Python. https://it.wikipedia.org/wiki/Python, 2022.

[10] Python Community. La crescita di python. https://www.python.it/blog, 2017.

[11] Introduzione ad ethereum. https://ethereum.org/it/developers/docs/intro-to-ethereum/, 2022.

[12] La macchina virtuale ethereum (evm). https://ethereum.org/it/developers/docs/evm/, 2022.

[13] Gas e commissioni. https://ethereum.org/it/developers/docs/gas/, 2022.

[14] Introduzione alle daap. https://ethereum.org/it/developers/docs/dapps/, 2022.

[15] Daniele Tumietto Andrea Giomo. Interoperabilità dati su blockchain: il caso di hyperledger besu. *Agenda Digitale*, 2019.

[16] Besu architecture. https://besu.hyperledger.org/en/stable/private-networks/, 2022.

[17] Solidity. https://soliditylang.org/, 2022.

[18] Structure of a contract. https://docs.soliditylang.org/en/v0.8.17/structure-of-a-contract.html, 2022.

[19] Truffle. https://trufflesuite.com/, 2022.

[20] Ganache. https://trufflesuite.com/ganache/, 2022.

[21] Web3.js library. https://web3js.readthedocs.io, 2022.

[22] Gerhard Münz, Sa Li, and Georg Carle. Traffic anomaly detection using k-means clustering, 2007.

[23] Create a qbft network. https://besu.hyperledger.org/en/stable/private-networks/tutorials/qbft/, 2022.

[24] Node keys and node address. https://besu.hyperledger.org/en/stable/public-networks/concepts/node-keys/, 2022.

[25] Network id and chain id. https://besu.hyperledger.org/en/stable/public-networks/concepts/network-and-chain-id/, 2022.

[26] Configure free gas networks. https://besu.hyperledger.org/en/stable/private-networks/how-to/configure/free-gas/, 2022.

[27] Shobhit Seth. Public, private, permissioned blockchains compared. *Investopedia*, 2022.

[28] Permissioning. https://besu.hyperledger.org/en/stable/private-networks/concepts/permissioning, 2022.

[29] Peer-to-peer tls. https://besu.hyperledger.org/en/stable/private-networks/how-to/configure/tls/p2p/, 2022.

[30] Bhashinee Nirmali. Steps to create keystores and truststores to be used in mutual ssl authentication. https://medium.com/, 2020.

[31] Use the hyperledger besu configuration file. https://besu.hyperledger.org/en/stable/public-networks/how-to/configuration-file/, 2022.

[32] Use truffle. https://besu.hyperledger.org/en/stable/public-networks/how-to/develop/truffle/, 2022.

Elenco delle figure

Elenco delle tabelle