

Tesina di:

Software Security and Blockchain

Blockchain sulla filiera alimentare sostenibile a cura di:

Gabriel Piercecchi
Tosca Pierro
Luca Pigliacampo
Caterina Sabatini



Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione



Tesina di:

Software Security and Blockchain

Esame per il corso tenuto dal Prof. Luca Spalazzi, durante l'anno accademico
2024-2025

9 CFU

Redazione del documento a cura di:

- **Piercecchi Gabriel** (Matr. 1120541) - s1120541@studenti.univpm.it
- **Pierro Tosca** (Matr. 1120542) - s1120542@studenti.univpm.it
- **Pigliacampo Luca** (Matr. 1098288) - s1098288@studenti.univpm.it
- **Sabatini Caterina** (Matr. 1121684) - s1121684@studenti.univpm.it

Anno Accademico 2024-2025

Anno I

Contents

I

Analisi Preliminare

1	Introduzione	6
1.1	Contabilità del carbonio	6
1.2	Compensazione del carbonio	6
1.3	Crediti di carbonio	6
1.3.1	Attori	7
1.4	Riassunto dei capitoli successivi	7
2	Requisiti	8
2.1	Raccolta dei requisiti	8
2.1.1	I requisiti relativi ai vari attori	8

II

Progettazione e Design

3	Progettazione	11
3.1	Modello i*	11
3.2	Identificazione e analisi dei rischi	13
3.2.1	Identificazione degli asset	13
3.2.2	Identificazione delle minacce	14
3.3	Valutazione degli attacchi con gli schemi di Jacobson	18
3.3.1	Use Case	18
3.3.2	Misuse Case	25
3.3.3	Abuse Case	29
3.4	Attack Tree	38
3.4.1	Attack Tree relativo ai dati	38
3.4.2	Attack Tree relativo alla scrittura dei dati	39
3.4.3	Attack Tree relativo alla lettura dei dati	39
3.4.4	Attack Tree relativo all'accesso ai dati in maniera condivisa per gli stakeholder	40
3.4.5	Attack Tree relativo alla misurazione dei parametri	41
3.4.6	Attack Tree relativo alla registrazione delle informazioni sulla blockchain	42
3.4.7	Attack Tree relativo al tracciamento delle azioni	43
3.4.8	Attack Tree relativo all'integrazione con gli smart contract	44
3.4.9	Attack Tree relativo alle politiche di autorizzazione	45
3.4.10	Attack Tree relativo alle politiche di autenticazione	45
3.4.11	Attack Tree relativo al wallet	46

4	Design	47
4.1	Catalogo Saltzer e Schroeder	47
4.2	Catalogo Sommerville	47
4.3	Catalogo OWASP	48
4.4	Esempio di analisi di un componente del sistema mediante Markov Chain	48
4.4.1	Markov Chain	48
4.4.2	Implementazione del Modello in PRISM	49
4.4.3	Verifica delle proprietà di Safety	52
4.4.4	Verifica della proprietà di Response	52
4.5	Esempio di sintesi di un monitor di Runtime Enforcement	52
4.5.1	Esempio di RE di una proprietà di Safety	53
4.5.2	Esempio di RE di una proprietà di Response	54

III**Sviluppo del sistema e Manuale d'uso**

5	Implementazione	56
5.1	Architettura	56
5.1.1	Componenti strutturali	56
5.1.2	L'interazione dei componenti	57
5.2	Parte off-chain	57
5.2.1	Database	57
5.2.2	Frontend	58
5.3	Parte on-chain	59
5.3.1	Blockchain	59
5.3.2	Smart contract	59
6	Manuale d'utilizzo	67
6.1	Manuale d'utilizzo	67
6.1.1	Come scaricare l'applicazione	67
6.2	Livello 0	72
6.3	Login e Registrazione	73
6.4	Livello 1	74
6.4.1	Farmer/Producer	75
6.4.2	Carrier	76
6.4.3	Seller	77
6.5	Livello 2	78



Analisi Preliminare

1	Introduzione	6
1.1	Contabilità del carbonio	
1.2	Compensazione del carbonio	
1.3	Crediti di carbonio	
1.4	Riassunto dei capitoli successivi	
2	Requisiti	8
2.1	Raccolta dei requisiti	

1. Introduzione

Questo documento illustra la progettazione e l'implementazione di un software per il tracciamento delle emissioni di carbonio. L'obiettivo principale è contribuire alla lotta contro il cambiamento climatico attraverso la creazione di un mercato regolato, in cui il numero di permessi per le emissioni è limitato.

1.1 Contabilità del carbonio

La contabilità del carbonio, o contabilità dei gas serra (GHG), comprende i metodi per misurare e monitorare le emissioni di anidride carbonica (CO_2) e di altri gas serra (GHG) generate da un'organizzazione.

L'equivalente di CO_2 è un'unità di misura che quantifica l'impatto sul riscaldamento globale di una determinata quantità di gas serra, esprimendolo in termini di CO_2 . Si misura in chilogrammi (kg) o tonnellate (t) di CO_2eq ed è utilizzato per confrontare e sommare i contributi dei diversi gas serra. Questo permette di stimare l'impronta di carbonio di un'attività umana, fornendo una scala comune per valutare gli effetti climatici di gas differenti.

1.2 Compensazione del carbonio

La compensazione del carbonio è un meccanismo di scambio del carbonio che consente alle entità di risparmiare o compensare le emissioni di gas serra. I risparmi di carbonio rappresentano le emissioni di carbonio evitate attraverso l'adozione di pratiche sostenibili rispetto a uno scenario di riferimento. Quando un'entità investe in un programma di compensazione o risparmio del carbonio, riceve crediti di carbonio o crediti di compensazione. Ogni credito di carbonio corrisponde a una riduzione (o risparmio) o rimozione (o compensazione) di un chilogrammo metrico di CO_2 equivalente (CO_2eq). Dopo la certificazione da parte di un ente governativo o di un organismo di certificazione indipendente, i crediti possono essere scambiati tra entità.

1.3 Crediti di carbonio

Il calcolo dei crediti di carbonio associati a un'attività determina se un attore genera o consuma crediti. Se il bilancio risulta positivo, vengono creati nuovi crediti e assegnati all'attore. Se invece il risultato è negativo, un numero equivalente di crediti viene sottratto dal suo portafoglio.

Nel caso in cui l'attore non disponga di crediti sufficienti, può riceverli da altri membri della catena di approvvigionamento, con un impatto negativo sulla sua reputazione.

L'obiettivo complessivo è garantire che la quantità netta di emissioni di CO_2eq associate a una determinata catena di approvvigionamento, dopo tutte le compensazioni, rimanga inferiore al limite massimo stabilito per quel settore.

1.3.1 Attori

Gli attori coinvolti in questo progetto sono:

- *Oracle*: gestisce le registrazioni e può effettuare trasferimenti eccezionali di coin;
- *Farmer*: produce le materie prime;
- *Carrier*: si occupa del trasporto delle materie prime e dei prodotti tra gli altri attori;
- *Producer*: trasforma le materie prime in prodotti, in base alla tipologia di azienda;
- *Seller*: vende materie prime e/o prodotti finiti;
- *Buyer*: può visualizzare il livello di emissioni di CO₂ e tracciare l'origine di ogni prodotto.

1.4 Riassunto dei capitoli successivi

Il presente documento è organizzato in capitoli, ciascuno dedicato a una fase del processo di progettazione del software:

- **Requisiti**: descrive la fase di raccolta dei requisiti richiesti dagli stakeholder per la realizzazione del software.
- **Progettazione**: si concentra sull'identificazione delle funzionalità e degli asset del software tramite diagrammi i*, sull'analisi dei rischi e delle politiche di sicurezza mediante il modello STRIDE, e sull'esame di Use Case, Misuse Case e Abuse Case tramite Attack Tree e schemi di Jacobson.
- **Design**: approfondisce le scelte architetturali, il design degli asset e la selezione delle tecnologie adottate per lo sviluppo del software.
- **Implementazione**: illustra l'applicazione pratica delle tecnologie scelte e lo sviluppo concreto del software.
- **Manuale d'uso**: descrive i test finali e fornisce le linee guida per l'utilizzo del software.

2. Requisiti

Il presente capitolo descrive in dettaglio il processo di raccolta dei requisiti, identificati dagli stakeholder come fondamentali per lo sviluppo del software. In particolare, vengono analizzati gli obiettivi e le necessità espresse dai vari attori coinvolti, con l'intento di garantire che il prodotto finale risponda adeguatamente alle aspettative e alle esigenze del progetto.

2.1 Raccolta dei requisiti

Il progetto ha come obiettivo la realizzazione di un software suddiviso in due sezioni: una **on-chain** e una **off-chain**. Questa divisione è stata pensata per garantire una gestione sicura dei dati da parte del software. La sezione *off-chain* include tutte le transazioni che avvengono al di fuori della blockchain, mentre la sezione *on-chain* comprende quelle transazioni eseguite direttamente sulla blockchain, dove vengono registrate e validate, assicurando un elevato livello di sicurezza e trasparenza.

Come accennato nel Capitolo 1, il software verrà sviluppato per consentire ai *Buyer* di visualizzare l'intera catena di produzione dei singoli prodotti, dal *Farmer* al *Seller*, compresi tutti i passaggi intermedi. Infatti, nel sistema verranno registrati i consumi di CO₂ relativi a ciascuna fase di produzione. In questo modo, il *Buyer* sarà in grado di consultare le emissioni di CO₂ rilasciate durante la lavorazione dei singoli prodotti.

2.1.1 I requisiti relativi ai vari attori

Requisiti per il **Buyer**

Il software deve consentire al *Buyer* di visualizzare l'intera filiera di elaborazione dei prodotti, evidenziando il relativo consumo di CO₂ in ciascuna fase della produzione.

Requisiti per il **Farmer**, il **Producer** e il **Seller**

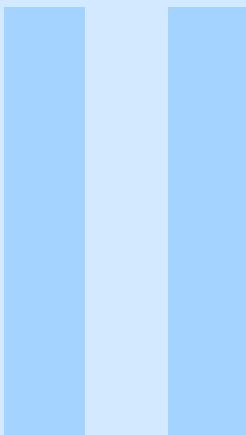
Il software deve permettere la registrazione delle diverse aziende e dei rispettivi dipendenti operanti al loro interno. Inoltre, deve facilitare l'inserimento dei prodotti, associandoli al relativo consumo di CO₂. È fondamentale che le aziende possano anche effettuare richieste per l'acquisto dei prodotti, in modo da avviarli alla lavorazione o alla vendita.

Requisiti per il **Carrier**

Per le aziende registrate come *Carrier*, incaricate della consegna dei prodotti, il software dovrà consentire la visualizzazione delle richieste di trasporto provenienti dalle altre aziende. Il *Carrier* dovrà inviare i dati relativi alla *Delivery* dei prodotti, indicando il consumo di CO₂ associato, insieme alle informazioni sull'azienda che ha effettuato la richiesta e sull'azienda da cui i prodotti verranno prelevati.

Requisiti relativi ai Coin

La caratteristica distintiva di questo progetto è la possibilità di scambiare i *Coin* tra le aziende, operazione che avverrà nella parte on-chain per garantire la sicurezza e la protezione dei dati aziendali. I *Coin* verranno accumulati o decurtati in base al consumo di CO₂ relativo alla quantità di prodotto trattato. Questo meccanismo viene descritto in dettaglio nel Capitolo 1.



Progettazione e Design

3	Progettazione	11
3.1	Modello i*	
3.2	Identificazione e analisi dei rischi	
3.3	Valutazione degli attacchi con gli schemi di Jacobson	
3.4	Attack Tree	
4	Design	47
4.1	Catalogo Saltzer e Schroeder	
4.2	Catalogo Sommerville	
4.3	Catalogo OWASP	
4.4	Esempio di analisi di un componente del sistema mediante Markov Chain	
4.5	Esempio di sintesi di un monitor di Runtime Enforcement	

3. Progettazione

Il seguente capitolo è dedicato all'analisi approfondita delle funzionalità e degli asset del software, con l'uso dei diagrammi i* per mappare gli elementi coinvolti. Inoltre, si procederà con l'esame della gestione dei rischi e delle politiche di sicurezza, adottando il modello STRIDE. Infine, verranno esaminati gli Use case, i Misuse case e gli Abuse case, utilizzando la metodologia degli Attack Tree e gli schemi proposti da Jacobson.

3.1 Modello i*

Nel Diagramma i* (Figura 3.1) è rappresentato l'intero processo di tracciamento delle emissioni di carbonio, con i vari attori coinvolti, senza l'inclusione del sistema.

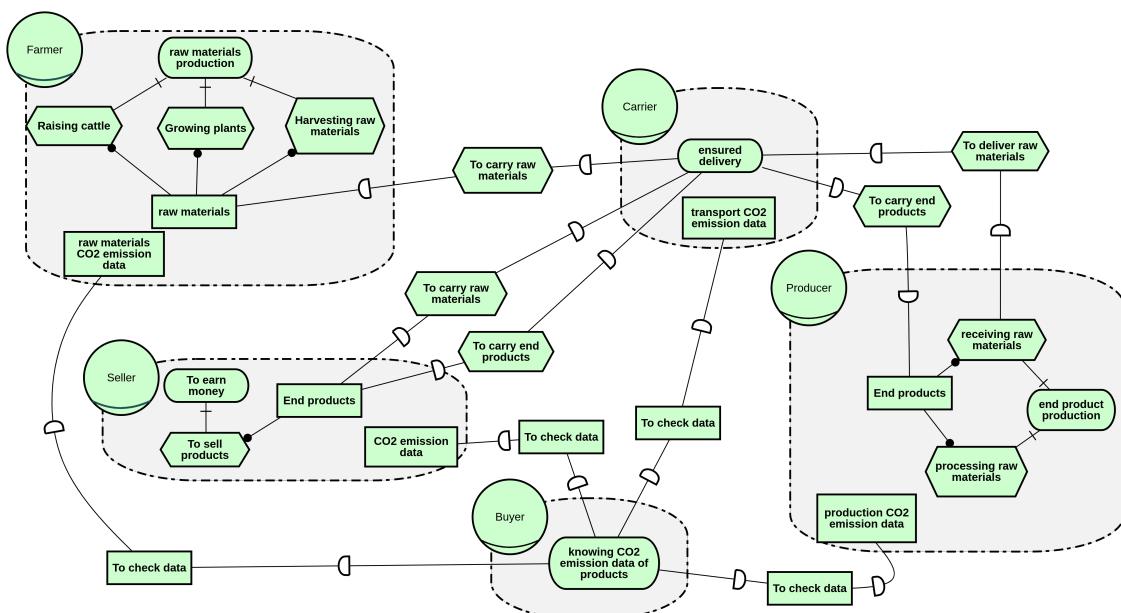


Figure 3.1: Modello i* SENZA sistema

Il processo inizia con il *Farmer*, responsabile della produzione delle materie prime, che comprende attività come la coltivazione e la raccolta delle colture, nonché l'allevamento degli animali. Le risorse ottenute vengono poi trasportate dal *Carrier*, il quale si occupa di garantire il buon esito di ogni fase di consegna.

Le materie prime arrivano al *Producer*¹, che è incaricato di trasformare le risorse ricevute in prodotti finiti. Successivamente, i prodotti finiti vengono nuovamente trasportati dal *Carrier* al *Seller*, il quale è responsabile della vendita degli articoli.

¹I prodotti finiti possono essere successivamente inviati ad altri Producer per ulteriori trasformazioni.

Infine, il *Buyer* ha la possibilità di consultare i dati relativi alle emissioni di CO₂ associati ai singoli prodotti, completando così il ciclo di tracciamento delle emissioni lungo tutta la filiera.

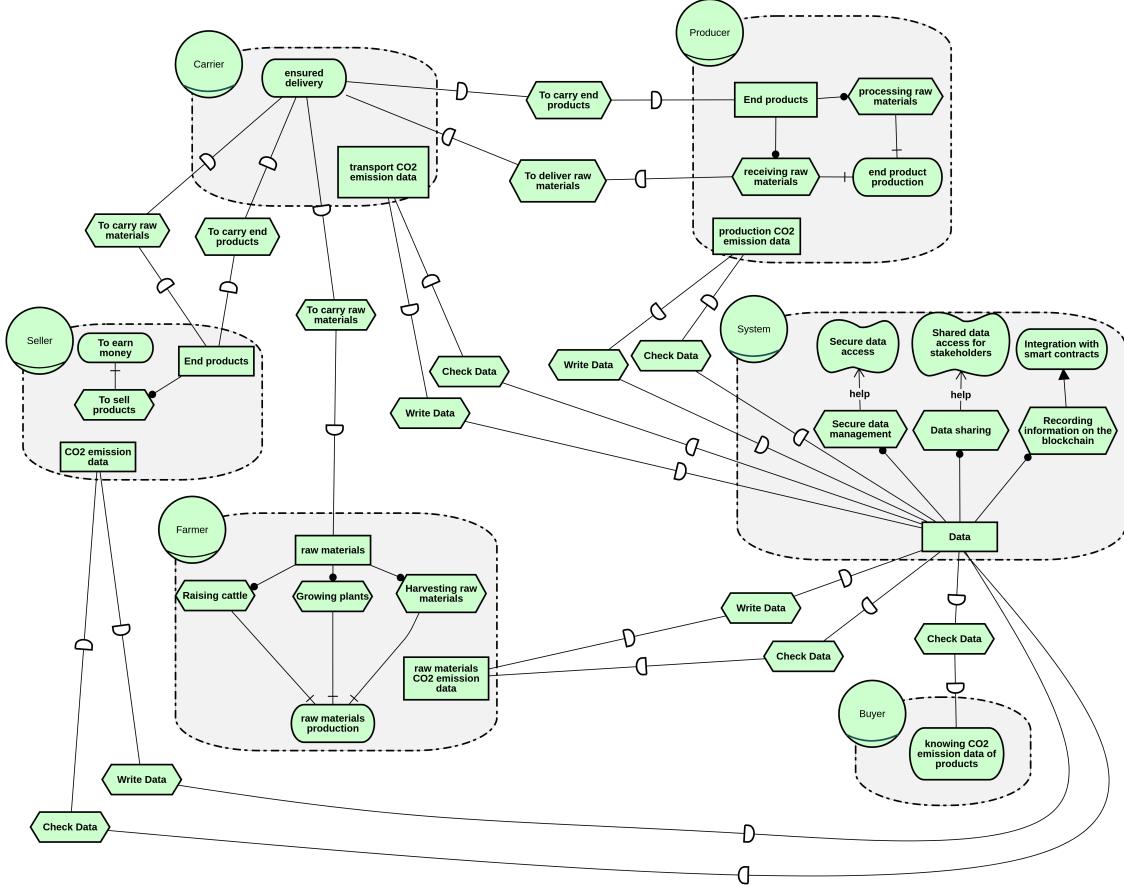


Figure 3.2: Modello i* CON sistema

Come illustrato in Figura 3.2, gli attori coinvolti nella produzione, tra cui il *Farmer*, il *Carrier*, il *Producer* e il *Seller* (con l'eccezione del *Buyer*, che non interviene direttamente nel processo di produzione), raccoglieranno, nel corso delle loro attività, i dati relativi alle emissioni di CO₂. Queste informazioni vengono successivamente elaborate dal sistema, rappresentato dal software *CO₂ Application*.

Il software garantisce un accesso sicuro e controllato ai dati, facilitando una condivisione regolamentata tra i diversi stakeholder. Ogni utente ha il permesso di visualizzare solo le informazioni a lui necessarie, assicurando così elevati livelli di sicurezza e protezione dei dati da accessi non autorizzati.

Il sistema adotta la tecnologia blockchain attraverso l'uso di smart contract, rafforzando ulteriormente la sicurezza e la tracciabilità delle informazioni. Ogni transazione e modifica viene registrata in modo immutabile e trasparente, fornendo una documentazione sicura e facilmente verificabile. Le operazioni di scrittura e verifica dei dati sono progettate per garantire che tutte le interazioni siano sicure, tracciabili e conformi agli standard di protezione, preservando l'integrità e la riservatezza delle informazioni archiviate.

3.2 Identificazione e analisi dei rischi

Nel contesto dello sviluppo software, e in particolare quando la tutela dei dati riveste un'importanza cruciale, la valutazione preliminare del rischio assume un ruolo fondamentale. Questo processo inizia con un'analisi accurata degli asset principali del sistema, finalizzata all'identificazione delle potenziali minacce. Successivamente, viene condotta un'analisi approfondita della probabilità che tali minacce si concretizzino, nonché delle relative conseguenze sul sistema. Sulla base dei dati raccolti, è possibile definire e implementare le strategie di mitigazione più appropriate, al fine di garantire la sicurezza complessiva e ridurre al minimo i rischi associati.

3.2.1 Identificazione degli asset

Gli asset considerati fondamentali per il processo di protezione della sicurezza del sistema sono i seguenti:

- **Dati:** rappresentano tutte le informazioni rilevanti per il sistema, come, ad esempio, i dati sulle emissioni di CO₂.
- **Scrittura dei dati:** comprende tutte le operazioni di scrittura e inserimento dei dati nel sistema.
- **Lettura dei dati:** riguarda tutte le operazioni di lettura dei dati da parte degli utenti autorizzati.
- **Accesso ai dati in maniera condivisa per gli stakeholder:** l'accesso ai dati deve essere sicuro e regolamentato, consentendo agli stakeholder coinvolti di operare in modo protetto e controllato.
- **Misurazione dei parametri:** definisce il processo di protezione dell'integrità e della correttezza dei dati, garantendo che le informazioni siano accurate e affidabili.
- **Registrazione delle informazioni su blockchain:** riguarda tutte le procedure per la registrazione sicura delle informazioni sulla blockchain.
- **Tracciamento delle azioni:** include il tracciamento dei log di sistema per monitorare tutte le operazioni e garantire la trasparenza delle attività.
- **Integrazione con gli smart contract:** riguarda l'uso della tecnologia blockchain, integrata con smart contract, per automatizzare i processi di convalida e esecuzione delle operazioni relative ai dati.
- **Politiche di autorizzazione:** definiscono i diritti degli utenti, regolando l'accesso alle risorse e le operazioni consentite all'interno del sistema.
- **Politiche di autenticazione:** consentono agli utenti di verificare la propria identità prima dell'accesso, tramite credenziali, autenticazione a più fattori o certificati digitali.
- **Wallet:** identifica il portafoglio elettronico e le chiavi associate che ne garantiscono la sicurezza.

In Figura 3.3 è mostrata una tabella che consente di identificare gli asset, associandoli al loro valore e agli obiettivi ad essi correlati.

Asset	Value	Objective
Dati	Risorsa cardine del software di cui si deve garantire la sicurezza e la disponibilità	Integrity, Availability, Resilience
Scrittura dei dati	L'integrità e l'affidabilità e precisione dei dati delle emissioni di CO2 misurate immesse nel sistema	Integrity, Non-repudiation, Authentication, Authorization
Lettura dei dati	L'attendibilità del processo di verifica per garantire l'integrità, la qualità e la disponibilità dei dati	Integrity, Availability
Gestione sicura dei dati	Le operazioni di CRUD sui dati devono essere opportunamente protette	Confidentiality, Authentication, Integrity
Accesso ai dati in maniera condivisa per gli stakeholder	Gli stakeholder autorizzati devono poter accedere ai dati quando desiderano	Integrity, Non-repudiation
Misurazione dei parametri	L'attività di rilevazione e registrazione dei parametri deve essere continuamente protetta e monitorata poiché fondamentale per il calcolo dei Carbon credits	Integrity, Confidentiality, Reliability, Non-repudiation
Registrazione delle informazioni su blockchain	È necessario porre il caricamento delle informazioni sulla blockchain in quanto quest'ultima monitorerà l'accesso ai dati in sicurezza	Confidentiality, Integrity, Availability, Authentication, Authorization, Non-repudiation
Tracciamento delle azioni	Prassi per il monitoraggio dei log prodotti dal software	Integrity, Confidentiality, Non-repudiation
Integrazione con gli smart contract	Azione volta all'integrazione dell'applicativo off-chain con la sua controparte on-chain	Authentication, Availability, Non-repudiation
Politiche di autorizzazione	Policy e criteri per la gestione delle autorizzazioni di sistema	Authorization, Non-repudiation
Politiche di autenticazione	Policy e criteri per la gestione delle autenticazione di sistema	Authorization, Non-repudiation
Wallet	Portafoglio elettronico contenente le chiavi che permettono l'autenticazione sulla blockchain	Integrity, Confidentiality, Non-repudiation

Figure 3.3: Asset

3.2.2 Identificazione delle minacce

Una volta individuati gli asset, si procede con l'analisi delle minacce, al fine di determinare le misure di mitigazione più appropriate. Questo processo viene condotto applicando il modello DUAL-STRIDE, che verrà descritto nel seguito.

Asset	Value [€]	Spawning	Environment	Repudiation	Information disclosure	DOS	levation of privilege	Denial	Unreliability	Access and Resilience	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C	Overall Cost
Dati	250.000-450.000		x					x			200.000 - 250.000	CAPEC-93: Log injection-Temp ering Forging	25.00 %	50.000 - 62.500	Controllare l'accesso ai file di log, validare i dati tramite liste di autorizzazioni, implementare meccanismi di processi, usare analisi statica per individuare vulnerabilità e evitare struttura che interrompa i canali di controllo nei log.	6.000 - 7.000	Fattibilità: Elevata. L'impostazione di controlli di validazione e la semplificazione dei log sono essenziali per prevenire la manipolazione dei log.	7%	200.000 - 250.000	14.000	5,0	20.000
				x			x				225.000 - 275.000	CAPEC-119: Collect and Analyze Information	37.50 %	84.375 - 103.125	Limitare l'accesso, critografare le informazioni, monitorare le attività, effettuare autenticazione forte, recuperare solo i dati necessari e garantire la conformità alle normative.	8.000 - 8.500	Fattibilità: Moderata. Garantisce la protezione delle informazioni raccolte e analizzate sulle quali. Un approccio strategico è l'implementazione di misure di sicurezza strutturate. È cruciale impostare rigorosi controlli di accesso e sistemi di monitoraggio per ridurre il rischio di esposizione delle informazioni sensibili.	9%	225.000 - 275.000	20.250	7,0	28.250
		x	x	x	x			x			170.000 - 180.000	CAPEC-66: SQL injection	25.00 %	42.500 - 45.000	La validazione dell'input, l'uso di query parametrizzate e l'impiego di query di inserire o modificare dati sono fondamentali per prevenire attacchi di iniezione SQL e la divulgazione di informazioni sensibili.	7.800 - 8.600	Fattibilità: Elevata. L'utilizzo di richieste parametrizzate o sistemi che è necessario e non presenti conflitti di contraddiczioni.	3%	170.000 - 180.000	5.100	3,8	12.900

Figure 3.4: Threat Model - Dati

Asset	Value (€)	Profiling	Tampering	Repudiation	Information disclosure	DOS	levation of privilege	Denial	Unreliability	Awareness of Resilience	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C.	Overall Cost
Scrittura dei dati	150.000-250.000		x	x			x				140.000 - 220.000	CAPEC-116: Communication channel Manipulation	37.50 %	52.500 - 82.500	Utilizzare canali di comunicazione sicuri, ad esempio via rete privata, per l'integrità e autenticità dei messaggi tramite sistemi critografici	5.500 - 6.700	Fattibilità: Elevata. Proteggere le risposte HTTP e prevenire l'HTTP response Splitting richiede una corretta validazione e controllo dei dati in ingresso, l'encoding dei caratteri speciali e il controllo rigoroso degli header HTTP. È importante tenere conto di implementare misure di sicurezza integrate nel ciclo di sviluppo per ridurre i rischi associati a tale attacco.	8%	140.000 - 220.000	11.200	6,5	16.700
			x				x				140.000 - 220.000	CAPEC-153: Input Data Manipulation	9.00 %	12.600 - 19.800	È necessario controllare la validità dei dati forniti dall'utente ed assicurarsi che non possono essere interpretati in maniera non voluta	1.500 - 1.600	Fattibilità: Elevata. Utilizzare i privilegi di utente con la limitazione degli stessi rappresentano misure di sicurezza fondamentali e ampiamente riconosciute nel settore informatico.	3%	140.000 - 220.000	4.200	5,5	5.500

Figure 3.5: Threat Model - Scrittura dei Dati

Asset	Value (€)	Profiling	Tampering	Repudiation	Information disclosure	DOS	levation of privilege	Denial	Unreliability	Awareness of Resilience	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C.	Overall Cost
Lettura dei dati	100.000-200.000	x	x	x				x			70.000 - 90.000	CAPEC-94: Adversary in the Middle	37.50 %	26.250 - 33.750	Garantire la sicurezza delle connessioni utilizzando critografia, autenticazione reciproca e canali sicuri per lo scambio delle chiavi.	5.000 - 5.500	Fattibilità: Media. La configurazione sicura di trasmissioni sicuri, come l'implementazione di TLS (Transport Layer Security), è una pratica comune e ben supportata dalle principali piattaforme di sviluppo e infrastrutture di rete. Tuttavia, per una installazione corretta, è necessario un livello avanzato di competenze tecniche, al fine di garantire la massima sicurezza e prevenire potenziali vulnerabilità.	8%	70.000 - 90.000	5.600	3,1	10.600

Figure 3.6: Threat Model - Lettura dei Dati

Asset	Value (€)	Profiling	Tampering	Repudiation	Information disclosure	DOS	levation of privilege	Denial	Unreliability	Awareness of Resilience	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C.	Overall Cost
Accesso ai dati in maniera condivisa per gli stakeholder	350.000-500.000	x	x								330.000 - 450.000	CAPEC-115: Authentication Bypass	25.00 %	82.500 - 107.500	L'uso di metodi di autenticazione deboli, come la validatione rigorosa dei input, la protezione contro le vulnerabilità di accesso diretto ai dati e il monitoraggio delle attività sospette.	7.500 - 8.700	Fattibilità: Media. Assicurare la funzionalità dei meccanismi di autenticazione deve essere attenta alla implementazione dei metodi di accesso, ma è assolutamente necessario per proteggere la sicurezza del sistema.	4%	330.000 - 450.000	13.200	8,2	
			x		x			x			150.000 - 250.000	CAPEC-63: Cross-Site Scripting	37.50 %	56.250 - 93.750	Progettare e implementare misure per prevenire attacchi XSS, convalidando i dati e mantenendo il software aggiornato.	3.500 - 4.500	Fattibilità: Elevata. Prevenire vulnerabilità XSS richiede un'attenzione particolare all'implementazione di meccanismi di sanitizzazione e validazione degli input, nonché l'uso corretto di tecniche di encoding nei contesti appropriati. Sono anche possibili richieste uno sforzo significativo, sono indispensabili per garantire la sicurezza dell'applicazione e proteggere gli utenti da attacchi mirati.	10%	150.000 - 250.000	15.000	10,8	18.500

Figure 3.7: Threat Model - Accesso ai dati in maniera condivisa per gli stakeholder

Asset	Value (€)	Profiling	Tampering	Repudiation	Information disclosure	DOS	levation of privilege	Denial	Unreliability	Awareness of Resilience	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C.	Overall Cost
Misurazione dei parametri	110.000-260.000	x		x			x				110.000 - 260.000	CAPEC-123: Buffer Manipulation	37.50 %	41.250 - 97.500	Utilizzare linguaggi sicuri, funzioni sicure e strumenti di protezione a livello di compilatore e sistema operativo.	5.000 - 6.800	Fattibilità: Elevata. La presenza della vulnerabilità legata alla manipolazione dei buffer richiede un'attenza particolare alla sicurezza del codice, con l'uso di funzioni sicure per la manipolazione di stringhe e array, controlli rigorosi dei limiti e l'adozione di linguaggi che offrono protezioni automatiche contro buffer overflow. Sebbene queste misure richiedano un significativo sforzo di sviluppo e implementazione, sono fondamentali per garantire la sicurezza e l'affidabilità del sistema.	8%	110.000 - 260.000	8.800	5,5	13.800

Figure 3.8: Threat Model - Misurazione dei parametri

Asset	Value (€)	Sniffing	Tampering	Repudiation	Information disclosure	DoS	Revelation of privilege	Denial of Service	Unreliability	Insufficient Resource	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C	Overall Cost
Registrazione delle informazioni su blockchain	110.000-220.000			x	x			x	x		90.000 - 110.000	CAPEC-23: File Content Injection	25.00 %	22.500 - 27.500	Aplicare rigorosi controlli sui dati, sanificare e validare l'input, e mantenere il software aggiornato.	4.000 - 4.500	Fattibilità: Elevata. Prevenire vulnerabilità di File Content Injection richiede un'attenta validazione degli input utilizzati per modificare o inserire contenuti dei file, oltre a restrizioni sui percorsi e formati consentiti. È necessario implementare controlli rigorosi per verificare l'autenticità e l'integrità dei dati forniti dagli utenti. Sebbene queste misure possano essere complesse da implementare, sono essenziali per proteggere l'applicazione da manipolazioni dannose.	8%	90.000 - 110.000	7.200	2,8	11.200

Figure 3.9: Threat Model - Registrazione delle informazioni su blockchain

Asset	Value (€)	Sniffing	Tampering	Repudiation	Information disclosure	DoS	Revelation of privilege	Denial of Service	Unreliability	Insufficient Resource	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C	Overall Cost
Tracciamento delle azioni	100.000-110.000		x				x				90.000 - 100.000	CAPEC-268: Audit Log Manipulation	25.00 %	22.500 - 25.000	Controlli rigorosi degli accessi, la protezione tramite crittografia, il versionamento dei log e l'uso di avvisi per attività sospette.	5.500 - 6.000	Fattibilità: Media. Prevenire manipolazione dei log di audit richiede l'adozione di meccanismi di protezione avanzati, come la scrittura di log in diversi luoghi diversi, l'uso di firme digitali per garantire l'integrità e l'archiviazione sicure nei sistemi dedicati. Sebbene queste misure possano comportare un aumento della complessità architettonale, sono indispensabili per assicurare la tracciabilità e la sicurezza degli eventi registrati.	8%	90.000 - 100.000	7.200	1,8	12.700

Figure 3.10: Threat Model - Tracciamento delle azioni

Asset	Value (€)	Sniffing	Tampering	Repudiation	Information disclosure	DoS	Revelation of privilege	Denial of Service	Unreliability	Insufficient Resource	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C	Overall Cost
				x	x			x	x		120.000 - 150.000	CAPEC-23: File Content Injection	25.00 %	30.000 - 37.500		4.700 - 5.200	Fattibilità: Elevata. Prevenire vulnerabilità di File Content Injection richiede un'attenta validazione degli input utilizzati per modificare o inserire contenuti dei file, oltre a restrizioni sui percorsi e formati consentiti. È necessario implementare controlli rigorosi per verificare l'autenticità e l'integrità dei dati forniti dagli utenti. Sebbene queste misure possano essere complesse da implementare, sono essenziali per proteggere l'applicazione da manipolazioni dannose.	8%	120.000 - 150.000	9.600	3,3	14.300
Integrazione con gli smart contract	150.000-170.000		x				x	x			105.000 - 140.000	CAPEC-130: Excessive Allocation	15.00 %	15.750 - 21.000	Limitare le risorse per gli utenti non privilegiati e valutare accuratamente ogni input per prevenire abusi.	3.700 - 4.000	Fattibilità: Media. Prevenire vulnerabilità legate all'allocatione eccessiva di risorse richiede di ridurre la gestione della memoria e un monitoraggio costante delle risorse allocate. L'implementazione di limiti di allocatione, insieme all'utilizzo di tecniche di controllo dinamico delle risorse e di validazione dell'input, può ridurre significativamente il rischio. Sebbene queste misure possano richiedere un certo impegno nel lo sviluppo e nel test, sono fondamentali per garantire la stabilità e la sicurezza dell'applicazione.	2%	105.000 - 140.000	2.100	2,7	5.800

Figure 3.11: Threat Model - Integrazione con gli smart contract

Asset	Value (€)	Spooling	Impersonation	Information disclosure	DOS	levation of privilege	Denial of service	Unreliability	Abuse of Resources and Resiliency	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C.	Overall Cost
Politiche di autorizzazione	70.000-90.000		x	x	x	x	x	x	x	60.000 - 80.000	CAPEC-39: Manipulating Opaque Client-based Data Tokens	15.00 %	9.000 - 12.000	Proteggere i dati critici e l'utile di autenticazione con tecniche come HMAC e criptografia, isolare alla validazione server-side, per garantire la confidenzialità, integrità e coerenza.	12.500 - 13.500	Fattibilità: Media. Prevenire la manipolazione dei token richiede l'adozione di tecniche di protezione avanzate, come l'uso di critografia robusta per garantire l'integrità e la riservatezza dei dati, nonché l'implementazione di meccanismi di validazione e verifica sul lato server.	2%	60.000 - 80.000	1.200	-0,4	13.700
				x	x					35.000 - 65.000	CAPEC-180: Exploiting Incorrectly Configured Access Control Security levels	15.00 %	5.250 - 9.750	Configurare correttamente il controllo accessi.	2.300 - 2.900	Fattibilità: Elevata. Prevenire questa tipologia di attacco richiede una corretta configurazione, sia essa una regolazione rigorosa dei permessi, l'adozione del principio di sicurezza minimale o il uso dei strumenti automatizzati per il controllo e la revisione delle configurazioni di accesso sono essenziali.	2%	35.000 - 65.000	700	1,0	3.000
			x	x	x					35.000 - 65.000	CAPEC-69: Target Programs with Elevated Privileges	37.5%	13.125 - 24.575	Applica il principio del minimo privilegio, valida i dati, monitora e controlla i aggiornamenti, limita l'esposizione dei servizi, monitora le risorse e proteggi i dati, garantendo la sicurezza e la resilienza del sistema.	4.500 - 5.500	Fattibilità: Elevata. La protezione contro questa tipologia di attacco è una misura di sicurezza comune, disponibile da molte soluzioni facilmente integrabili nella maggior parte dei sistemi. L'adozione di tecniche, come il controllo dei privilegi, permettono di limitare efficacemente i rischi derivanti dall'accesso non autorizzato a processi privilegiati.	8%	35.000 - 65.000	2.800	1,3	7.300

Figure 3.12: Threat Model - Politiche di autorizzazione

Asset	Value (€)	Spooling	Impersonation	Information disclosure	DOS	levation of privilege	Denial of service	Unreliability	Abuse of Resources and Resiliency	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C.	Overall Cost
Politiche di autenticazione	70.000-90.000		x	x	x					30.000 - 60.000	CAPEC-114: Authentication Abuse	9.00 %	2.700 - 5.400	L'uso di autenticazione riduttiva, come la connessione di forza bruta, il monitoraggio delle attività sospette e la validazione continua delle credenziali.	6.500 - 7.500	Fattibilità: Elevata. Mentre le limitazioni del tentativo di accesso e l'autenticazione multifattore (MFA) sono ampiamente supportate dalle moderne e rispondenti soluzioni efficaci per prevenire abusi nell'autenticazione. L'implementazione di tali misure riduce significativamente i rischi di accesso non autorizzato.	2%	30.000 - 60.000	600	-0,7	7.100
			x	x	x					30.000 - 60.000	CAPEC-115: Authentication Bypass	15.00 %	4.500 - 9.000	L'uso di metodi di autenticazione robusti, la validazione rigorosa degli input, la limitazione delle vulnerabilità di accesso diretto ai dati e il monitoraggio delle attività sospette.	5.200 - 6.200	Fattibilità: Elevata. L'implementazione di HTTPS e la validazione dell'input sono misure di sicurezza standard e molto semplici da integrare, con HTTPS che garantisce le ritrattorie delle comunicazioni e una protezione contro le intercettazioni e le manipolazioni. La gestione sicura delle sessioni (session management) richiede un approccio più complesso.	3%	30.000 - 60.000	900	-0,3	6.100
			x	x	x					30.000 - 60.000	CAPEC-560: Use of Known Domain Credentials	25.00 %	7.500 - 15.000	Implementare l'autenticazione a più fattori, politiche di password forti, limitare i privilegi degli account e monitorare gli accessi anomali.	7.500 - 8.500	Fattibilità: Elevata, utilizzo di credenziali di dominio consente di aumentare la sicurezza, ma le misure di sicurezza per prevenirne sono ben noto. Eseguire diverse soluzioni di controllo integrabili nei sistemi di autenticazione esistenti, come l'adozione dell'autenticazione multi-fattore (MFA), la gestione sicura delle credenziali e il monitoraggio delle attività sospette.	5%	30.000 - 60.000	1.500	-0,2	9.000

Figure 3.13: Threat Model - Politiche di autenticazione

Asset	Value (€)	Profiling	Tampering	Spreading	Information disclosure	DDoS	Exposure of private info	User	Vulnerability	Advanced Persistent Threat	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C	Overall Cost
Wallet	70.000-90.000		x				x				60.000 - 70.000	CAPEC-115: Authentication Bypass	25.00 %	15.000 - 17.500	L'uso di metodi di autenticazione robusti, la validazione rigorosa degli input, la protezione contro le vulnerabilità di accesso diretto ai dati e il monitoraggio delle attività sospette.	6.300 - 7.000	Potibilità: Elevata. L'autenticazione basata su chiavi e la validazione dell'input sono misure di sicurezza standard e relativamente facili da integrare, con HTTPS che garantisce la confidenzialità delle comunicazioni e una protezione robusta contro l'intercettazione e manipolazioni. Tuttavia, la protezione sicura delle sessioni degli utenti è meno intuitiva e il processo di validazione delle informazioni deve essere utilizzato nel rispetto della privacy.	3%	60.000 - 70.000	1.800	1,1	8.100
		x	x								60.000 - 70.000	CAPEC-22: Exploiting Trust in Client	25.00 %	15.000 - 17.500	Autenticare i client, usare firme digitali, autenticazione a due fattori e consolidare l'input remoto per migliorare la sicurezza.	6.300 - 7.000	Potibilità: Bassa. Un attaccante che ottiene accesso a informazioni relative a vulnerabilità nel processo di comunicazione HTTPS può forzare che un client malintenzionato interagisca con il server in modi non previsti dal server legittimo, il quale non è controllato dall'utente. Tuttavia, la protezione contro questo tipo di attacco è relativamente semplice da implementare attraverso l'adozione di tecniche come la validazione approfondita dei dati in ingresso, l'uso di certificati SSL/TLS sicuri (es. HTTPS) e la crittografia dei dati scambiati, che riducono significativamente il rischio di manipolazioni.	3%	60.000 - 70.000	1.800	1,1	8.100

Figure 3.14: Threat Model - Wallet

3.3 Valutazione degli attacchi con gli schemi di Jacobson

Questa fase prevede una suddivisione dettagliata di ogni rischio, con l'obiettivo di condurre un'analisi approfondita e sistematica. Tale approccio consente di individuare le misure di mitigazione più appropriate, garantendo una risposta adeguata e mirata per ciascuno degli scenari identificati.

Per effettuare tale analisi, sono stati adottati gli schemi di Jacobson, che vengono presentati successivamente in relazione ai casi più significativi e rilevanti. Per una maggiore chiarezza, gli schemi sono stati organizzati secondo le tre categorie principali.

3.3.1 Use Case

Gli Use Case descrivono le funzionalità e i comportamenti attesi del software, delineando gli scenari in cui il sistema opera correttamente, in conformità con le esigenze previste.

Di seguito sono riportati i casi più rilevanti individuati.

Case Type	Use Case	Case ID	UC-1
Case Name	Inserimento Dati		
Actors	Oracle, Farmer, Carrier, Producer, Seller, CO2 Application		
Description	Funzione che permette ad ognuno degli attori coinvolti di registrare dei nuovi dati relativi alle informazioni personali e/o relativi alla produzione di CO2 delle materie e prodotti distribuiti nella supply-chain nel software. Inoltre, permette la modifica di dati persistenti.		
Data	Tutti i dati relativi alle informazioni personali e alla produzione di CO2 degli articoli distribuiti nella supply-chain.		
Stimulus and preconditions	Gli attori coinvolti devono prima autenticarsi sul sistema per accedere al servizio di scrittura.		
Basic Flow	1) Autenticazione 2) Inserimento dati 3) Registrazione Dati all'interno del sistema 4) Registrazione Dati all'interno dei log 5) Uscita dell'utente dal sistema		
Alternative Flow	1) Autenticazione 2) Inserimento dati 3) Registrazione Dati MODIFICATI all'interno del sistema 4) Registrazione Dati MODIFICATI all'interno dei log 5) Uscita dell'utente dal sistema		
Exception Flow	Presenza di possibili errori generati da un incorretto inserimento dei dati o da una loro formattazione non consentita.		
Response and Postconditions	Verrà mostrato un messaggio all'inserimento dei dati nel sistema.		
Non Functional Requirements	Integrità, Affidabilità		
Comments	Per la modifica di dati sulle emissioni di CO2 già presenti nel sistema è necessario che l'utente si identifichi come Oracle.		

Figure 3.15: Use Case - Inserimento Dati

Case Type	Use Case	Case ID	UC-2
Case Name	Lettura Dati		
Actors	Oracle, Farmer, Carrier, Producer, Seller, Buyer, CO2 Application		
Description	Funzione che permette ad ognuno degli attori coinvolti di visualizzare i dati personali e/o le emissioni di CO2 dei prodotti nella supply-chain.		
Data	Tutti i dati relativi alle informazioni personali e alla produzione di CO2 degli articoli distribuiti nella supply-chain.		
Stimulus and preconditions	Gli attori coinvolti, che non sono Buyer, devono prima autenticarsi sul sistema per visualizzare i dati personali.		
Basic Flow	<ol style="list-style-type: none"> 1) Autenticazione 2) Lettura Dati Personalni 3) Registrazione dell'attività all'interno dei log 4) Uscita dell'utente dal sistema 		
Alternative Flow	<ol style="list-style-type: none"> 1) Ricerca Dati CO2 dei prodotti nella supply-chain 2) Lettura Dati CO2 dei prodotti nella supply-chain 3) Registrazione dell'attività all'interno dei log 		
Exception Flow	Presenza di possibili errori generati da un incorretto inserimento dei dati, da una loro formattazione non consentita o nel caso in cui i dati non siano disponibili.		
Response and Postconditions	Verrà mostrato un messaggio all'inserimento dei dati nel sistema durante la ricerca.		
Non Functional Requirements	Integrità, Affidabilità		
Comments	-		

Figure 3.16: Use Case - Lettura Dati

Case Type	Use Case	Case ID	UC-3
Case Name	Autorizzazione basata sui ruoli		
Actors	Oracle, Farmer, Carrier, Producer, Seller, Buyer, CO2 Application		
Description	Funzione che assegna e verifica i permessi di accesso alle funzionalità e ai dati in base al ruolo specifico di ciascun attore nella supply chain.		
Data	Tutti i dati relativi alle informazioni personali, ai prodotti e alle emissioni di CO2.		
Stimulus and preconditions	Gli attori coinvolti, che non sono Buyer, devono autenticarsi sul sistema per ottenere i permessi legati al loro ruolo. I ruoli sono predefiniti con accessi e autorizzazioni specifici. Il Buyer ha permessi minimi e non richiede autenticazione.		
Basic Flow	<ol style="list-style-type: none"> 1) L'utente tenta di accedere a una funzione o a un dato specifico. 2) Il sistema verifica il ruolo e le autorizzazioni assegnate. 3) Se necessario il sistema richiede l'autenticazione da parte dell'utente 4) L'utente effettua la procedura di login 5) Il sistema verifica le credenziali 6) Una volta autenticato, il sistema recupera il ruolo associato all'utente. 7) Il sistema determina le autorizzazioni e le funzionalità accessibili in base al ruolo 8) L'utente esegue l'azione prefissata 9) Registrazione attività all'interno dei log 		
Alternative Flow	<p>AF-1: Login fallito</p> <ol style="list-style-type: none"> 1) L'utente tenta di accedere a una funzione o a un dato specifico. 2) Il sistema verifica il ruolo e le autorizzazioni assegnate. 3) Se necessario il sistema richiede l'autenticazione da parte dell'utente 4) L'utente effettua la procedura di login 5) Il sistema verifica le credenziali 6) Le credenziali non sono valide 7) Il sistema mostra un messaggio di errore e richiede di reinserire le credenziali. <p>AF-2: Non serve il login</p> <ol style="list-style-type: none"> 1) Il Buyer accede al sistema senza autenticazione. 2) Il sistema assegna all'utente Buyer autorizzazioni e funzionalità limitate: Ricerca dei prodotti e Lettura delle informazioni pubbliche sui prodotti e sulle emissioni di CO2 3) Registrazione attività all'interno dei log. 		
Exception Flow	Il sistema non è in grado di verificare le credenziali a causa di un errore tecnico. Viene mostrato un messaggio di errore personalizzato.		
Response and Postconditions	L'utente ha accesso ai dati e alle funzionalità in base ai ruoli con cui sono autorizzati		
Non Functional Requirements	Integrità, Confidenzialità, Affidabilità		

Figure 3.17: Use Case - Autorizzazione basata sui ruoli

Case Type	Use Case	Case ID	UC-4
Case Name	Tracciamento tramite Log		
Actors	CO2 Application		
Description	Funzione del sistema che registra e traccia le attività svolte dagli utenti all'interno del programma. Queste attività vengono archiviate in file di log, contenenti informazioni dettagliate sulle azioni eseguite da ciascun utente.		
Data	Informazioni sull'uso del software.		
Stimulus and preconditions	Un utente deve eseguire una qualsiasi operazione all'interno del programma		
Basic Flow	1) Esecuzione di una qualsiasi azione all'interno del programma 2) Conferma da parte del programma dell'avvenuta esecuzione dell'azione 3) Memorizzazione dell'attività appena svolta e di chi l'ha effettuata all'interno dei file di log.		
Alternative Flow	-		
Exception Flow	Il sistema tenta di memorizzare un'attività, ma si verifica un errore legato alla memoria rimanente del disco in cui si trova il sistema o/e ai permessi di lettura e scrittura legati alla cartella in cui sono posizionati i file di log. Il sistema genera un messaggio di errore interno.		
Response and Postconditions	Memorizzazione dei dati all'interno dei file di log avvenuta con successo		
Non Functional Requirements	Affidabilità, Non-Ripudiabilità		
Comments	-		

Figure 3.18: Use Case - Tracciamento tramite Log

Case Type	Use Case	Case ID	UC-5
Case Name	Caricamento dati sulla Blockchain		
Actors	CO2 Application		
Description			Funzione che carica all'interno della blockchain le emissioni di CO2 dei prodotti presenti nella supply-chain per garantirne la sicurezza e l'integrità
Data			Tutti i dati relativi ai prodotti e le loro emissioni di CO2.
Stimulus and preconditions			Un altro attore autenticato ed autorizzato inserisce dei nuovi dati legati alla CO2 dei prodotti in modo da poter essere salvati sulla blockchain
Basic Flow			1) Inserimento e memorizzazione dei dati all'interno del sistema 2) Registrazione dei dati sulla Blockchain attraverso lo smartContract
Alternative Flow			-
Exception Flow			Possibili eccezioni generate da degli errori di connessione durante la registrazione dei dati sulla Blockchain. Il sistema tenta di conseguenza di riconnettersi in maniera automatica con intervalli regolari e memorizzando i dati in una coda temporanea. Si possono generare ulteriori eccezioni nel caso in cui vi sia la mancanza di gas e/o durante l'esecuzione degli smartContract
Response and Postconditions			L'utente ha accesso ai dati e alle funzionalità in base ai ruoli con cui sono autorizzati
Non Functional Requirements			Integrità, Confidentialità, Affidabilità
Comments			-

Figure 3.19: Use Case - Caricamento dati sulla Blockchain

Case Type	Use Case	Case ID	UC-6
Case Name	Registrazione Attori		
Actors	Oracle, CO2 Application		
Description		Funzione che permette l'iscrizione degli attori Farmer, Producer, Carrier e Seller e dei loro impiegati.	
Data		Tutti i dati personali degli attori e dei loro impiegati	
Stimulus and preconditions		L'attore Oracle deve inserire i dati personali forniti tramite delle form di richiesta da parte degli attori Farmer, Producer, Carrier e Seller e dei loro impiegati. Per registrare gli impiegati è necessario registrare prima L'attore (Azienda) a cui appartengono.	
Basic Flow		1) L'utente effettua la procedura di login come Oracle 2) Il sistema verifica le credenziali 3) Una volta autenticato, il sistema recupera il ruolo associato all'utente. 4) Il sistema determina le autorizzazioni e le funzionalità accessibili in base al ruolo 5) L'utente esegue l'azione prefissata di registrazione degli Attori ed/o Impiegati 6) Registrazione attività all'interno dei log 7) Uscita dell'utente dal sistema	
Alternative Flow		1) L'utente effettua la procedura di login come Oracle 2) Il sistema verifica le credenziali 3) Le credenziali non sono valide 4) Il sistema mostra un messaggio di errore e richiede di reinserire le credenziali.	
Exception Flow		Il sistema non è in grado di verificare le credenziali a causa di un errore tecnico. Viene mostrato un messaggio di errore personalizzato.	
Response and Postconditions		L'utente con il ruolo di Oracle registra con successo gli Attori e/o i loro impiegati.	
Non Functional Requirements		Integrità, Confidentialità, Affidabilità	

Figure 3.20: Use Case - Registrazione Attori

3.3.2 Misuse Case

I Misuse Case descrivono le funzionalità e i comportamenti attesi del software, delineando gli scenari in cui il sistema opera correttamente in conformità alle esigenze previste.

In seguito sono riportati i casi più rilevanti individuati.

Case Type	Misuse Case	Case ID	MC-01
Case Name	Authentication Bypass		
Actors	Attaccanti, Utenti, CO2 Application		
Description	Un utente ottiene l'accesso a dati o funzionalità riservate senza completare il processo di autenticazione, eludendo le procedure previste.		
Data	Tutti i dati riservati, non accessibili a utenti non autenticati.		
Stimulus and preconditions	L'utente non ha eseguito la procedura di autenticazione.		
Attack Flow 1	Nel tentativo di effettuare la procedura di login, l'utente ottiene l'accesso a dati riservati, senza completare correttamente il processo di autenticazione previsto.		
Attack Flow 2	Un attaccante, mediante l'ottenimento di un token o altre manipolazioni del software, induce il sistema a riconoscere erroneamente il completamento di una procedura di autorizzazione, senza averla realmente eseguita.		
Response and Postconditions	Accesso a dati privati da parte di utenti non autorizzati. Impossibilità di accesso a dati significativi per utenti con privilegi superiori.		
Non Functional Requirements	Affidabilità, Autenticazione, Confidentialità, Non-ripudiazione		
Mitigations	Il software deve integrare meccanismi di autorizzazione robusti e prevedere un fail state che impedisca a un utente l'accesso a dati ai quali non dovrebbe poter accedere.		
Comments	-		

Figure 3.21: Misuse Case - Authentication Bypass

Case Type	Misuse Case	Case ID	MC-02
Case Name	Input Data Manipulation		
Actors	Attaccanti, Utenti, CO2 Application		
Description	Un utente inserisce dati malformati nel sistema interferendo con la fornitura di servizi e potenzialmente compromettendo l'integrità dei dati		
Data	Tutti i dati a cui l'utente ha accesso.		
Stimulus and preconditions	L'utente necessita di inserire dati nel sistema .		
Attack Flow 1	In un campo di input l'utente fornisce dati contenenti caratteri non previsti, che causano errori o risultati errati nella lettura o scrittura di tali dati.		
Response and Postconditions	Sistema instabile, vengono restituiti dati non validi in alcune operazioni.		
Non Functional Requirements	Affidabilità, Integrità, Resilienza		
Mitigations	Controlli di sanità dei dati in ingresso, ad esempio utilizzando una whitelist di caratteri ammessi.		
Comments	-		

Figure 3.22: Misuse Case - Input Data Manipulation

Case Type	Misuse Case	Case ID	MC-03
Case Name	Exploiting Incorrectly Configured Access Control Security Levels		
Actors	Attaccanti, Utenti, CO2 Application		
Description	Un utente, a seguito di una procedura di login errata o sfruttando vulnerabilità nel processo di autenticazione, potrebbe compromettere il sistema di autorizzazione, attribuendo permessi inadeguati agli utenti e compromettendo il normale funzionamento del sistema.		
Data	Tutti i dati riservati, non accessibili a utenti non autenticati.		
Stimulus and preconditions	La presenza di vulnerabilità nel processo di autenticazione.		
Attack Flow 1	L'utente, inserendo dati sbagliati durante il processo di login, compromette il corretto funzionamento del sistema di autorizzazione.		
Attack Flow 2	Un attaccante riesce ad accedere a dati riservati sfruttando una qualsiasi vulnerabilità e compromettendo il corretto funzionamento del sistema di autorizzazione.		
Response and Postconditions	Accesso a dati privati da parte di utenti non autorizzati. Funzionamento anomalo e non gestibile del sistema. Fallimento durante il procedimento di modifica dei dati.		
Non Functional Requirements	Affidabilità, Autorizzazione, Confidentialità, Non-ripudiazione		
Mitigations	Il software deve integrare meccanismi di autorizzazione robusti e prevedere un fail state che impedisca a un utente l'accesso a dati ai quali non dovrebbe poter accedere.		
Comments	-		

Figure 3.23: Misuse Case - Exploiting Incorrectly Configured Access Control Security Levels

Case Type	Misuse Case	Case ID	MC-04
Case Name	Excessive Allocation		
Actors	Attaccanti, Utenti, CO2 Application		
Description	Un utente, intenzionalmente o meno, inserisce dati di input non consentiti o sovraccarica il sistema con richieste inappropriate, determinando l'esaurimento delle risorse e compromettendo il corretto funzionamento del sistema.		
Data	Tutti i dati che possono essere inseriti nel sistema, inclusi input non validi o di dimensione eccessiva		
Stimulus and preconditions	La presenza di vulnerabilità nel sistema che consente l'inserimento di input di dimensione eccessiva o non validi.		
Attack Flow 1	L'utente inserisce dati sbagliati o di dimensione eccessiva, causando il malfunzionamento del sistema di autorizzazione a causa dell'esaurimento delle risorse.		
Attack Flow 2	Un attaccante sfrutta una vulnerabilità nel sistema per inviare richieste in eccesso o dati non validi, causando l'esaurimento delle risorse del sistema e il fallimento del processo di autorizzazione.		
Response and Postconditions	Funzionamento anomalo e non gestibile del sistema a causa dell'esaurimento delle risorse. Fallimento durante il processo di modifica dei dati a causa della mancanza di risorse necessarie.		
Non Functional Requirements	Affidabilità, Resilienza		
Mitigations	Il software deve integrare meccanismi di limitazioni di input e controlli sulle risorse, per evitare il loro esaurimento e, quindi, malfunzionamenti da parte del sistema		
Comments	-		

Figure 3.24: Misuse Case - Excessive Allocation

3.3.3 Abuse Case

Gli Abuse Case delineano comportamenti dannosi, messi in atto deliberatamente da attori malintenzionati, con l'intento di compromettere il sistema.

Di seguito sono riportati i casi più rilevanti individuati.

Case Type	Abuse Case	Case ID	AC-01
Case Name	Log Injection-Tempering Forging		
Actors	Attaccanti, CO2 Application		
Description	È un attacco informatico in cui un malintenzionato manipola i log di sistema per alterare o nascondere informazioni cruciali. Nel contesto di un sistema di forgia (forging), dove la sicurezza dei dati e la tracciabilità delle operazioni sono fondamentali, un abuso di questo tipo può compromettere seriamente l'integrità del sistema e la qualità del prodotto finale.		
Data	I file di log del sistema		
Stimulus and preconditions	L'attaccante deve avere accesso al sistema di logging tramite: - Credenziali compromesse - Vulnerabilità del software - Mancanza di crittografia dei log - Assenza di controlli di accesso.		
Attack Flow 1	Accesso ai log: L'attaccante ottiene l'accesso non autorizzato al sistema di log.		
Attack Flow 2	Iniezione o manomissione: L'attaccante inserisce dati falsi o modifica i log esistenti utilizzando API, script o strumenti di hacking.		
Attack Flow 3	Cancellazione dei log: L'attaccante elimina alcune voci per mascherare la propria attività.		
Response and Postconditions	Rilevazione: Il sistema dovrebbe rilevare modifiche non autorizzate nei log. Notifica: Inviare un avviso agli amministratori di sistema. Recupero: Ripristinare i log compromessi dai backup.		
Non Functional Requirements	Confidenzialità, Integrità, Autorizzazione		
Mitigations	Autenticazione a più fattori (MFA). Crittografia avanzata dei log. Monitoraggio continuo dei log per identificare anomalie. Notifiche in tempo reale per modifiche sospette.		
Comments	-		

Figure 3.25: Abuse Case - Log Injection-Tempering Forging

Case Type	Abuse Case	Case ID	AC-02
Case Name	SQL Injection		
Actors	Attaccanti, CO2 Application		
Description	È un attacco in cui un malintenzionato sfrutta vulnerabilità nei campi di input di un'applicazione per iniettare comandi SQL malevoli. Questo può consentire all'attaccante di accedere, manipolare o cancellare i dati all'interno di un database, compromettendo l'integrità, la disponibilità e la riservatezza delle informazioni.		
Data	Dati critici memorizzati nel database		
Stimulus and preconditions	Il sistema accetta input dagli utenti senza una validazione o sanitizzazione adeguata. L'attaccante ha una conoscenza di base della struttura del database. La gestione degli errori dell'applicazione espone dettagli del database.		
Attack Flow 1	Iniezione di input malevolo: L'attaccante inserisce codice SQL nei campi di input (es. moduli web o URL).		
Attack Flow 2	Esecuzione del comando SQL: Il sistema interpreta l'input come un comando SQL legittimo e lo esegue.		
Attack Flow 3	Accesso ai dati: L'attaccante ottiene dati sensibili, modifica informazioni critiche o distrugge il contenuto del database.		
Response and Postconditions	<p>Rilevazione: Identificare e registrare i tentativi di iniezione SQL nel sistema di monitoraggio.</p> <p>Mitigazione: Applicare patch di sicurezza e aggiornamenti al sistema per correggere la vulnerabilità.</p> <p>Ripristino: Ripristinare il database da un backup integro.</p>		
Non Functional Requirements	Resilienza, Integrità, Disponibilità		
Mitigations	Utilizzare query parametrizzate o prepared statements per evitare che i comandi SQL vengano interpretati come dati		
Comments			

Figure 3.26: Abuse Case - SQL Injection

Case Type	Abuse Case	Case ID	AC-03
Case Name	Collect and Analyze Information		
Actors	Attaccanti, CO2 Application		
Description	Un attacco volto a raccogliere e analizzare informazioni sensibili o critiche dal sistema per sfruttarle a scopo malevolo.		
Data	Dati critici memorizzati nel database		
Stimulus and preconditions	Il sistema espone informazioni non protette o ha vulnerabilità che consentono l'accesso non autorizzato. L'attaccante ha strumenti per raccogliere e analizzare grandi volumi di dati.		
Attack Flow 1	L'attaccante ottiene accesso al sistema tramite vulnerabilità o credenziali compromesse.		
Attack Flow 2	L'attaccante raccoglie informazioni sensibili sfruttando API, query al database o strumenti di scraping.		
Attack Flow 3	Le informazioni raccolte vengono analizzate e utilizzate per scopi malevoli, come attacchi mirati o divulgazione.		
Response and Postconditions	Monitoraggio attivo per identificare accessi anomali. Bloccare l'attaccante e rafforzare le vulnerabilità sfruttate. Verifica e ripristino dei dati compromessi		
Non Functional Requirements	Confidenzialità, Integrità, Autorizzazione, Affidabilità		
Mitigations	Implementare meccanismi di autenticazione e autorizzazione robusti. Monitoraggio attivo e rilevamento di anomalie nel sistema.		
Comments	-		

Figure 3.27: Abuse Case - Collect and Analyze Information

Case Type	Abuse Case	Case ID	AC-04
Case Name	Communication channel Manipulation		
Actors	Attaccanti, CO2 Application		
Description	Questo attacco mira a manipolare i canali di comunicazione tra i componenti del sistema, intercettando, alterando o bloccando messaggi cruciali.		
Data	Dati trasmessi nei canali di comunicazione		
Stimulus and preconditions	Il canale di comunicazione non utilizza protocolli sicuri (es. mancanza di crittografia). Assenza di verifiche di integrità sui messaggi trasmessi. Accesso dell'attaccante al canale di comunicazione.		
Attack Flow 1	L'attaccante accede al canale di comunicazione e monitora i messaggi scambiati.		
Attack Flow 2	L'attaccante modifica i messaggi trasmessi per manipolare il comportamento del sistema o inviare informazioni errate		
Attack Flow 3	L'attaccante interrompe il flusso di comunicazione, causando malfunzionamenti o interruzioni del servizio.		
Response and Postconditions	Identificare e registrare anomalie nei canali di comunicazione. Reindirizzare il traffico su canali sicuri e isolare il canale compromesso. Garantire l'integrità e la continuità del servizio attraverso protocolli sicuri.		
Non Functional Requirements	Confidenzialità, Integrità, Affidabilità		
Mitigations	Monitorare i canali per rilevare attività anomale o non autorizzate.		
Comments	-		

Figure 3.28: Abuse Case - Communication channel Manipulation

Case Type	Abuse Case	Case ID	AC-05
Case Name	Adversary in the Middle		
Actors	Attaccanti, CO2 Application		
Description	Un attacco in cui un malintenzionato si inserisce tra due parti in comunicazione, intercettando, alterando o bloccando i messaggi. Questo tipo di attacco compromette gravemente la confidenzialità, l'integrità e l'affidabilità dei dati trasmessi.		
Data	Credenziali di accesso		
Stimulus and preconditions	Il sistema utilizza protocolli non sicuri o vulnerabili. L'attaccante ha accesso alla rete o al canale di comunicazione. Assenza di certificati o autenticazioni mutue per le comunicazioni.		
Attack Flow 1	L'attaccante si pone come intermediario e cattura i messaggi scambiati tra due entità.		
Attack Flow 2	L'attaccante modifica i messaggi trasmessi per influenzare le operazioni o ottenere vantaggi illeciti.		
Attack Flow 3	L'attaccante reinvia messaggi alterati o blocca del tutto la comunicazione per causare disservizi.		
Response and Postconditions	Registrare e analizzare anomalie nel traffico di rete. Bloccare l'accesso non autorizzato e ripristinare la comunicazione sicura. Implementare misure per garantire la continuità operativa senza compromissioni future.		
Non Functional Requirements	Confidenzialità, Integrità, Affidabilità, Autorizzazione		
Mitigations	Applicazione di un canale sicuro per le informazioni		
Comments	-		

Figure 3.29: Abuse Case - Adversary in the Middle

Case Type	Abuse Case	Case ID	AC-06
Case Name	Cross-Site Scripting		
Actors	Attaccanti, CO2 Application		
Description	Un attacco di tipo Cross-Site Scripting (XSS) si verifica quando un attaccante inietta script malevoli in una pagina web visualizzata da altri utenti. Questo tipo di attacco può compromettere la sicurezza dell'applicazione web, permettendo il furto di informazioni sensibili, la manipolazione del contenuto della pagina o l'esecuzione di azioni non autorizzate a nome dell'utente.		
Data	Credenziali degli utenti, session cookies, dati sensibili visualizzati nel browser		
Stimulus and preconditions	Il sistema consente l'inserimento di input utente non validati o non sanitizzati. Il browser dell'utente interpreta e esegue script malevoli presenti nella pagina web.		
Attack Flow 1	L'attaccante inserisce codice malevolo in campi di input o URL.		
Attack Flow 2	L'utente visualizza una pagina vulnerabile che esegue il codice iniettato.		
Attack Flow 3	Il codice eseguito può rubare dati, reindirizzare l'utente a siti fraudolenti, o manipolare i contenuti della pagina.		
Response and Postconditions	Registrare e notificare ogni tentativo di iniezione sospetto. Rimuovere il codice malevolo e notificare agli utenti potenzialmente colpiti. Applicare patch per eliminare la vulnerabilità.		
Non Functional Requirements	Resilienza, Integrità, Disponibilità, Confidentialità		
Mitigations	Validare e sanificare tutti gli input utente. Utilizzare librerie di escaping per impedire l'inserimento di codice in HTML, JavaScript, o altri contesti.		
Comments	-		

Figure 3.30: Abuse Case - Cross-Site Scripting

Case Type	Abuse Case	Case ID	AC-07
Case Name	File Content Injection		
Actors	Attaccanti, CO2 Application		
Description	Un attacco in cui un malintenzionato inserisce o modifica il contenuto di un file di sistema o di applicazione per compromettere il funzionamento del sistema, eseguire codice dannoso o manipolare i dati.		
Data	File eseguibili o script		
Stimulus and preconditions	Il sistema non valida o protegge adeguatamente i file caricati o gestiti dall'utente		
Attack Flow 1	L'attaccante carica un file contenente codice dannoso o manipolato attraverso una vulnerabilità del sistema.		
Attack Flow 2	L'attaccante modifica un file esistente per alterarne il contenuto e ottenere un comportamento malevolo.		
Attack Flow 3	Il sistema esegue il file compromesso, esponendo l'ambiente a vulnerabilità come l'esecuzione di codice arbitrario o la compromissione dei dati.		
Response and Postconditions	Monitorare e loggare attività sospette relative ai file caricati o modificati. Impedire l'esecuzione di file non autorizzati e rimuovere i file compromessi. Ripristinare i file originali e rafforzare i controlli di sicurezza sui file.		
Non Functional Requirements	Riservatezza, Affidabilità, Disponibilità, Resilienza		
Mitigations	Validare e sanificare i file caricati, impedendo l'esecuzione di contenuti non sicuri.		
Comments	-		

Figure 3.31: Abuse Case - File Content Injection

Case Type	Abuse Case	Case ID	AC-08
Case Name	Authentication Abuse		
Actors	Attaccanti, CO2 Application		
Description	Un attacco in cui un malintenzionato sfrutta vulnerabilità nel processo di autenticazione per ottenere accesso non autorizzato al sistema o alle risorse protette.		
Data	Credenziali di autenticazione		
Stimulus and preconditions	Le credenziali di accesso sono deboli o facilmente indovinabili (password deboli o predittive).		
Attack Flow 1	L'attaccante tenta di accedere al sistema utilizzando un nome utente e una password rubata o indovinata.		
Attack Flow 2	L'attaccante prova una serie di credenziali (password comuni o precedentemente esposte) per forzare l'accesso all'account.		
Attack Flow 3	-		
Response and Postconditions	Monitorare tentativi di login sospetti e anomali per identificare attacchi di tipo brute-force o credential stuffing. Avvisare l'utente di accessi non autorizzati e richiedere il cambio della password.		
Non Functional Requirements	Confidenzialità, Affidabilità, Resilienza, Disponibilità		
Mitigations	Forzare l'uso di password forti e la loro rotazione periodica.		
Comments	-		

Figure 3.32: Abuse Case - Authentication Abuse

Case Type	Abuse Case	Case ID	AC-09
Case Name	Exploiting Trust in Client		
Actors	Attaccanti, CO2 Application		
Description	L'attacco sfrutta la fiducia che un'applicazione cliente ha nei confronti di un server o di un altro componente del sistema.		
Data	Dati contenuti nell'applicazione e nella blockchain.		
Stimulus and preconditions	Il client si fida implicitamente delle risposte del server o delle altre entità con cui comunica.		
Attack Flow 1	L'attaccante invia dati falsificati al client, facendo credere al sistema che provengano da una fonte affidabile.		
Attack Flow 2	L'attaccante sfrutta una vulnerabilità nel client per eseguire codice arbitrario.		
Attack Flow 3	-		
Response and Postconditions	Compromissione della privacy dei dati. Esecuzione di attività non previste. Log di attività non accurati.		
Non Functional Requirements	Riservatezza, Autenticazione		
Mitigations	Implementare meccanismi sul server per controllare la validità del client e delle azioni da non eseguire.		
Comments	-		

Figure 3.33: Abuse Case - Exploiting Trust in Client

3.4 Attack Tree

Gli Attack Tree offrono una rappresentazione visiva delle possibili vulnerabilità di un sistema, esplorando in modo sistematico i percorsi che un attaccante potrebbe seguire per raggiungere il suo obiettivo. A partire da un nodo centrale, che rappresenta l'intento dell'intruso, i rami si diramano in diverse azioni o fasi che potrebbero essere intraprese per compromettere la sicurezza. Questa tecnica consente di identificare in dettaglio le varie strategie di attacco, offrendo una panoramica completa delle potenziali minacce. Grazie a questa struttura, è possibile valutare la probabilità di successo di ciascun attacco e determinare il possibile impatto sul sistema, facilitando così l'individuazione delle aree più vulnerabili e l'implementazione di misure di difesa mirate.

Nei paragrafi successivi sono riportati gli Attack Tree per ciascun asset di dati individuato precedentemente².

3.4.1 Attack Tree relativo ai dati

L'Attack Tree mostrato in Figura 3.34 illustra le modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe utilizzare per compromettere dati sensibili. Vengono analizzate tre tipologie di attacco: *Log Injection-Tampering Forging*, *Collect and Analyze Information* e *SQL Injection*, descritte brevemente tramite la struttura ad albero.

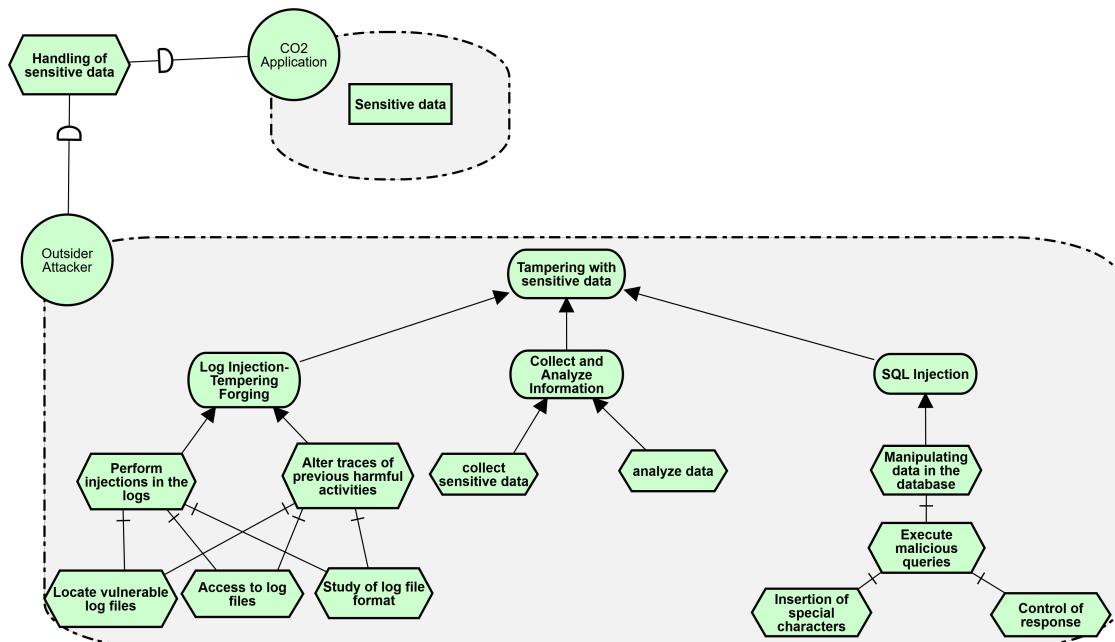


Figure 3.34: Attack Tree relativo ai dati

²Il nodo radice non è stato direttamente connesso alla risorsa del sistema a causa di una limitazione tecnica del software di editing utilizzato, nonostante il linguaggio I* supporti tale collegamento secondo le sue specifiche.

3.4.2 Attack Tree relativo alla scrittura dei dati

L'Attack Tree mostrato in Figura 3.35 illustra le modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe utilizzare per manomettere i dati di input. Vengono analizzate due tipologie di attacco: *Communication Channel Manipulation* e *Input Data Manipulation*. Queste vengono illustrate in maniera sintetica attraverso la struttura ad albero, evidenziando le specifiche tecniche e le implicazioni di ciascun approccio.

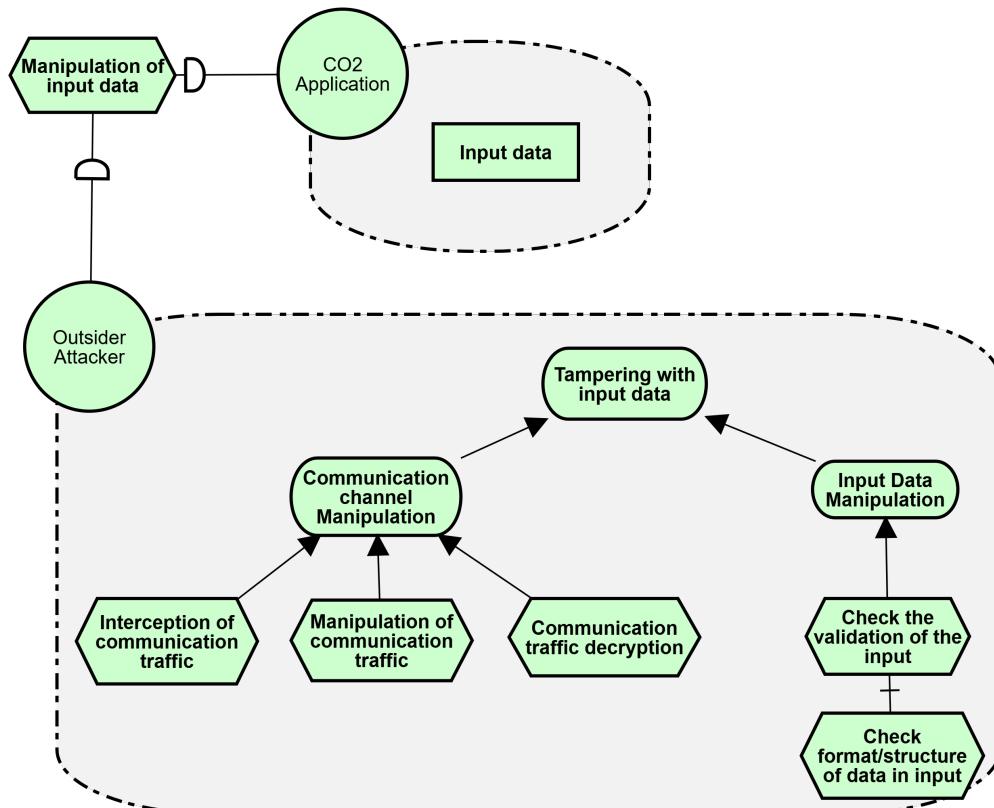


Figure 3.35: Attack Tree relativo alla scrittura dei dati

3.4.3 Attack Tree relativo alla lettura dei dati

L'Attack Tree riportato in Figura 3.36 illustra le possibili modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe adottare per interferire con la lettura dei dati. L'analisi si concentra su una tecnica specifica, *Adversary in the Middle*, descritta sinteticamente attraverso la struttura ad albero.

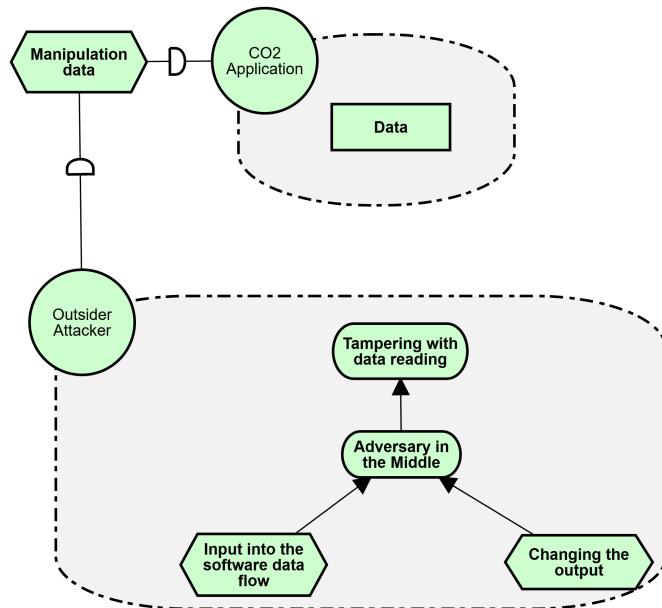


Figure 3.36: Attack Tree relativo alla lettura dei dati

3.4.4 Attack Tree relativo all'accesso ai dati in maniera condivisa per gli stakeholder

L'Attack Tree mostrato in Figura 3.37 illustra le modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe utilizzare per compromettere l'accesso ai dati condivisi tra i possibili stakeholder. Vengono analizzate due tipologie di attacco: *Authentication Bypass* e *Cross-Site Scripting*, descritte brevemente tramite la struttura ad albero.

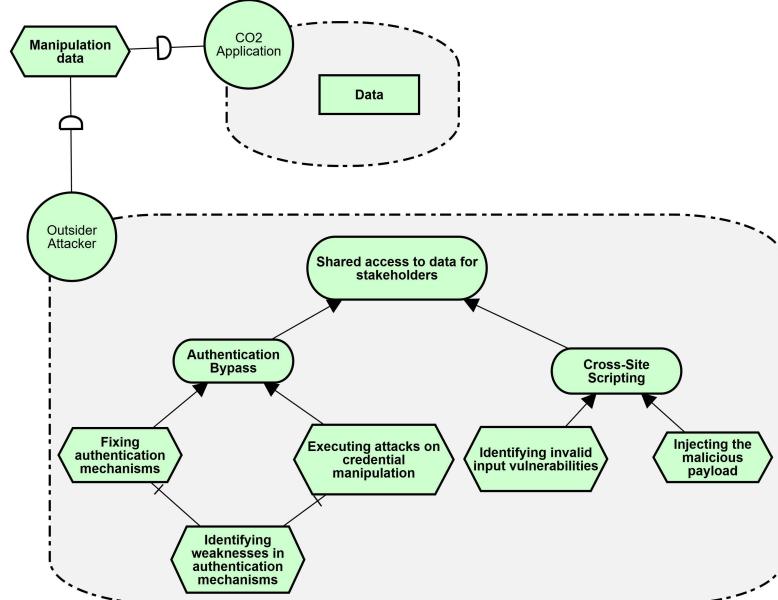


Figure 3.37: Attack Tree relativo accesso ai dati in maniera condivisa per gli stakeholder

3.4.5 Attack Tree relativo alla misurazione dei parametri

L'Attack Tree riportato in Figura 3.38 illustra le possibili modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe adottare per interferire sulla misurazione di determinati parametri. L'analisi si concentra su una tecnica specifica, *Buffer Manipulation*, descritta sinteticamente attraverso la struttura ad albero.

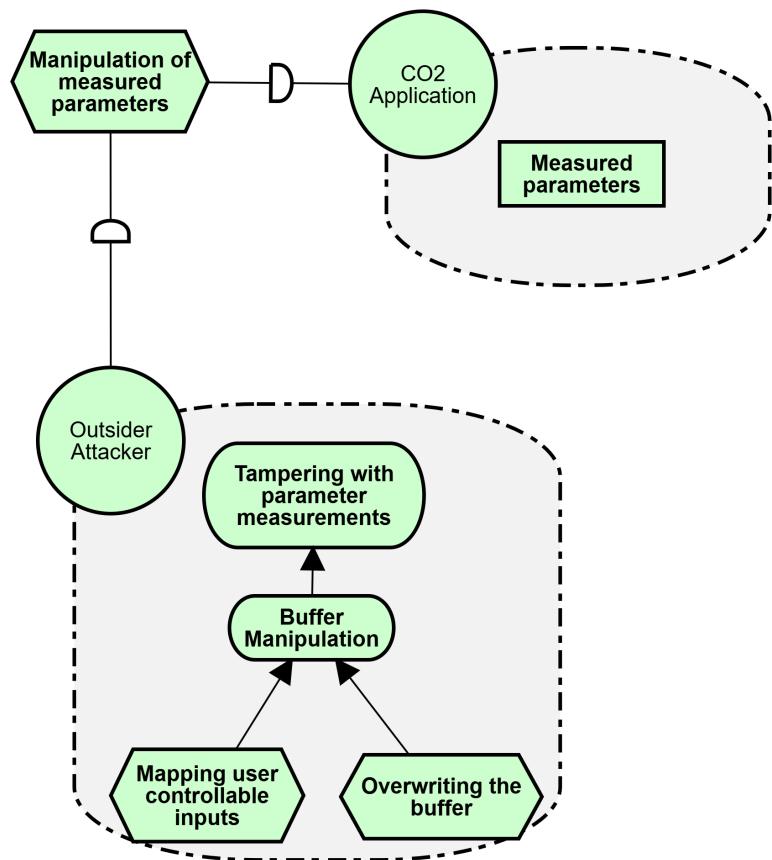


Figure 3.38: Attack Tree relativo alla misurazione dei parametri

3.4.6 Attack Tree relativo alla registrazione delle informazioni sulla blockchain

L'Attack Tree mostrato in Figura 3.39 illustra le modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe utilizzare per compromettere la registrazione delle informazioni sulla blockchain. Viene analizzata una tipologia di attacco, *File Content Injection*, illustrata in maniera sintetica attraverso la struttura ad albero.

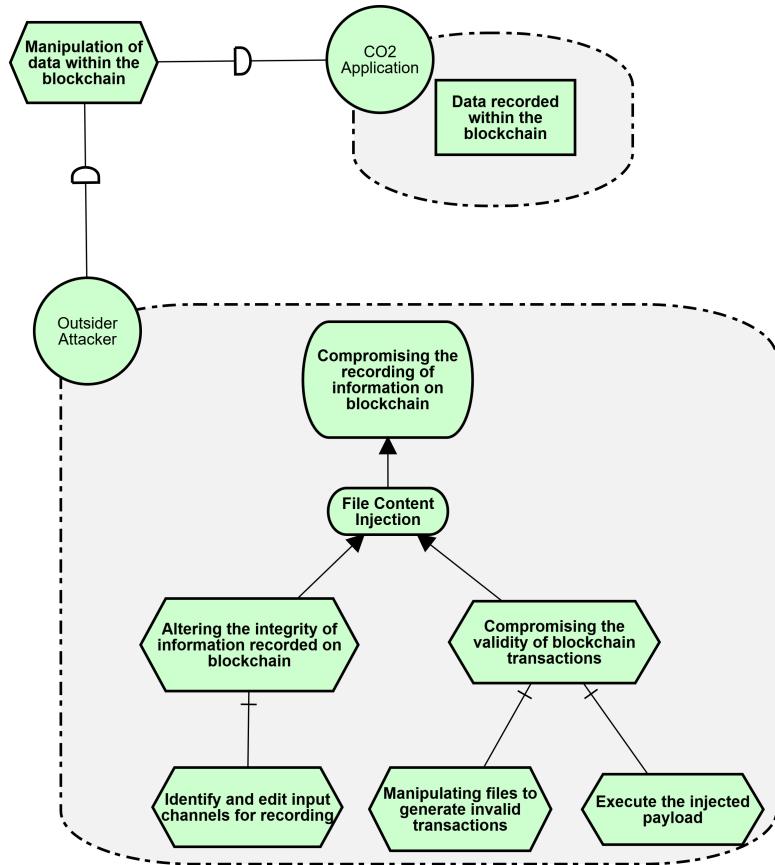


Figure 3.39: Attack Tree relativo alla registrazione delle informazioni sulla blockchain

3.4.7 Attack Tree relativo al tracciamento delle azioni

L'Attack Tree mostrato in Figura 3.40 illustra le modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe utilizzare per compromettere il tracciamento delle azioni. Viene analizzata una specifica tecnica di attacco, *Audit Log Manipulation*, descritta brevemente tramite la struttura ad albero.

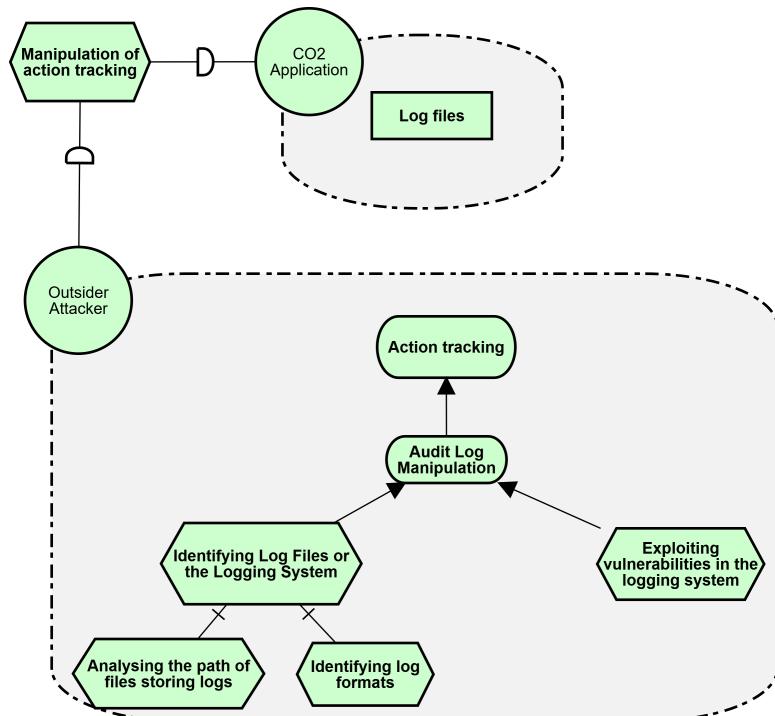


Figure 3.40: Attack Tree relativo al tracciamento delle azioni

3.4.8 Attack Tree relativo all'integrazione con gli smart contract

L'Attack Tree mostrato in Figura 3.41 illustra le modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe utilizzare per manomettere l'integrità e l'esecuzione degli smart contract. Vengono analizzate due tipologie di attacco: *File Content Injection* e *Excessive Allocation*. Queste vengono illustrate in maniera sintetica attraverso la struttura ad albero, evidenziando le specifiche tecniche e le implicazioni di ciascun approccio.

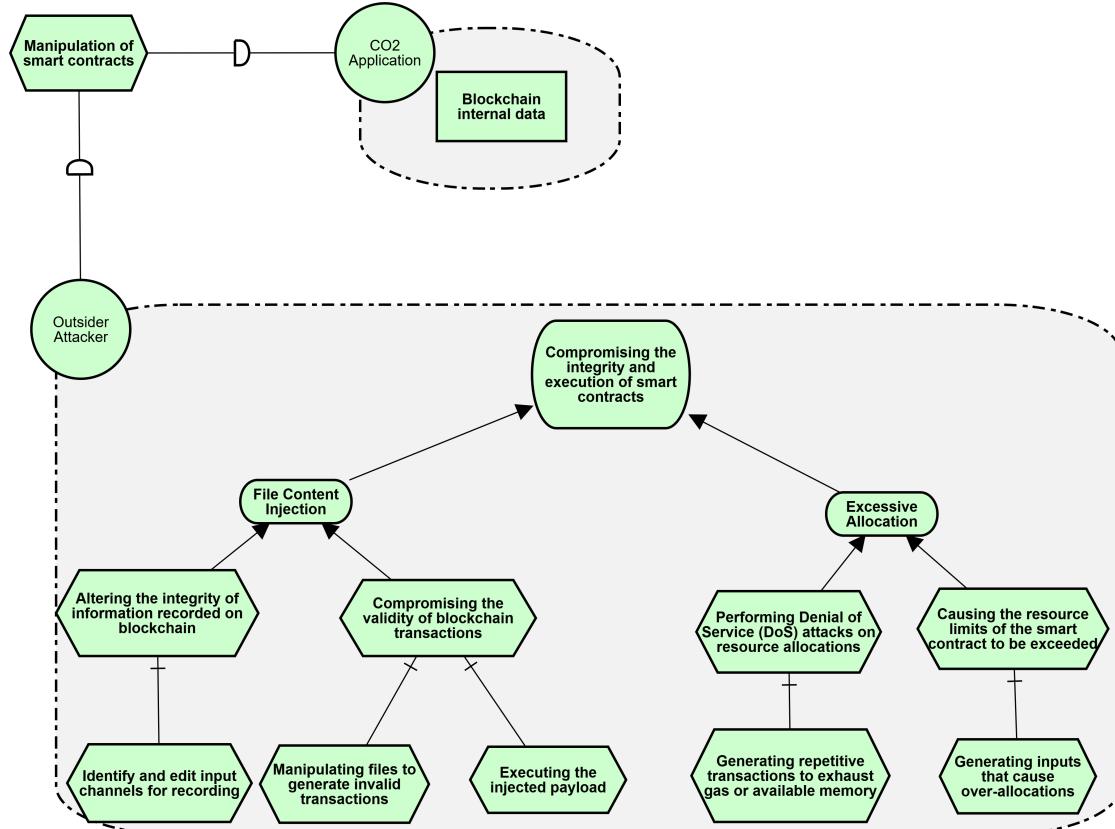


Figure 3.41: Attack Tree relativo all'integrazione con gli smart contract

3.4.9 Attack Tree relativo alle politiche di autorizzazione

L'Attack Tree mostrato in Figura 3.42 illustra le modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe utilizzare per compromettere le politiche di autorizzazione. Vengono analizzate tre tipologie di attacco: *Manipulating Opaque Client-based Data Tokens*, *Exploiting Incorrectly Configured Access Control Security Levels* e *Target Programs with Elevated Privileges*, descritte brevemente tramite la struttura ad albero.

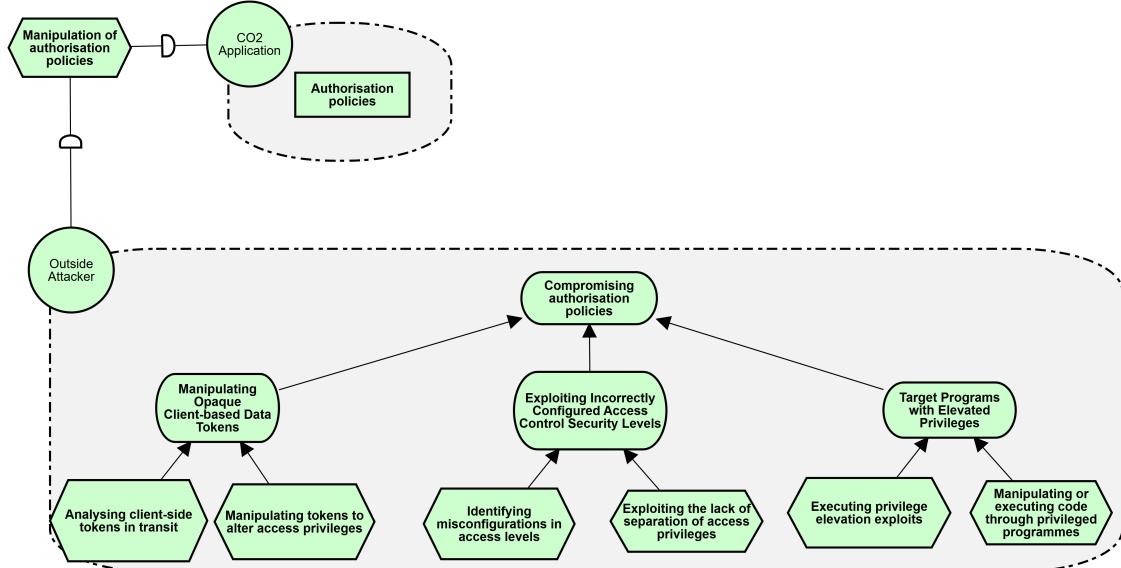


Figure 3.42: Attack Tree relativo alle politiche di autorizzazione

3.4.10 Attack Tree relativo alle politiche di autenticazione

L'Attack Tree mostrato in Figura 3.43 illustra le modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe utilizzare per manomettere le politiche di autenticazione. Vengono analizzate tre tipologie di attacco: *Authentication Abuse*, *Authentication Bypass* e *Use of Known Domain Credentials*. Queste vengono illustrate in maniera sintetica attraverso la struttura ad albero, evidenziando le specifiche tecniche e le implicazioni di ciascun approccio.

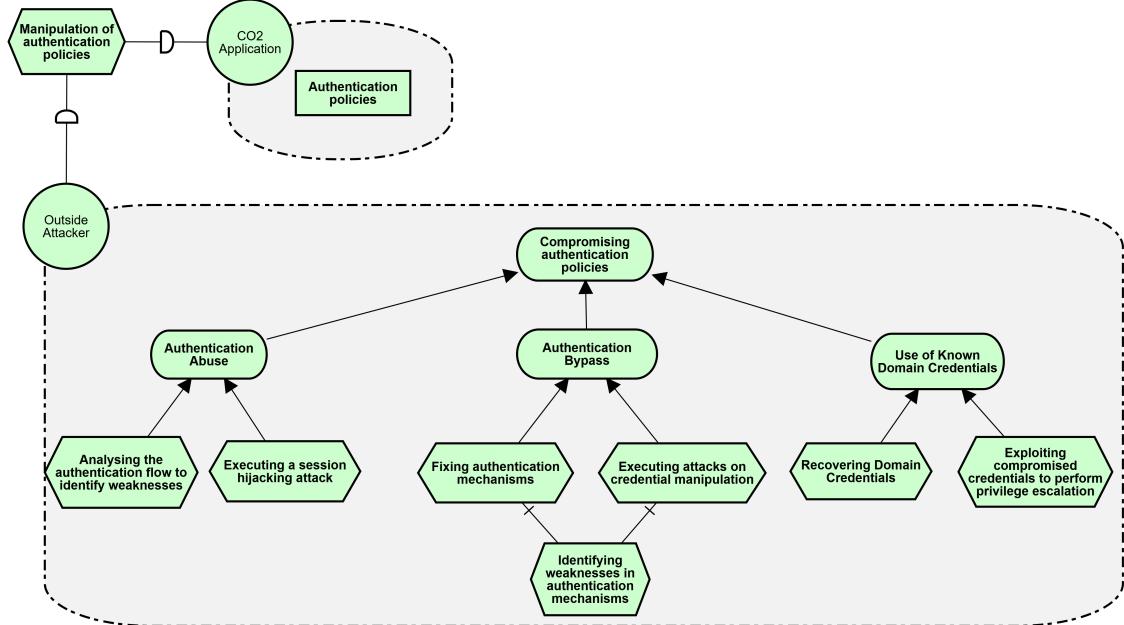


Figure 3.43: Attack Tree relativo alle politiche di autenticazione

3.4.11 Attack Tree relativo al wallet

L'Attack Tree mostrato in Figura 3.44 illustra le modalità di attacco che un attore malevolo (Outsider Attacker) potrebbe utilizzare per compromettere il wallet. Vengono analizzate due tipologie di attacco: *Authentication Bypass* e *Exploiting Trust in Client*. Queste vengono illustrate in maniera sintetica attraverso la struttura ad albero.

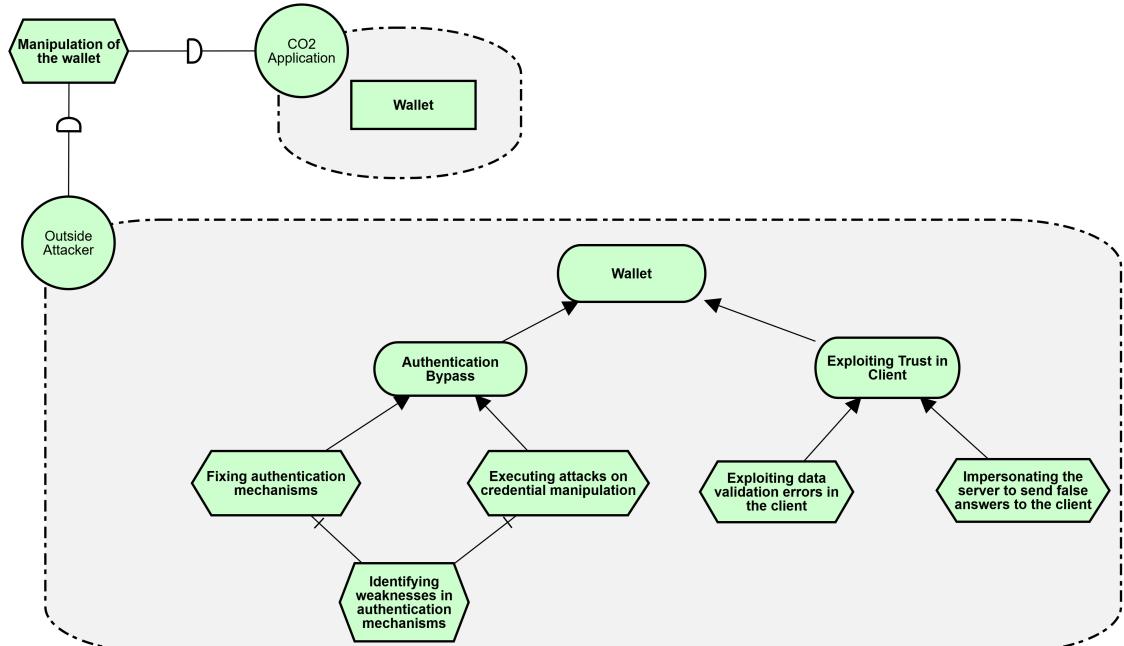


Figure 3.44: Attack Tree relativo al wallet

4. Design

Il presente capitolo è dedicato alla progettazione sicura del software oggetto di analisi, con particolare attenzione a una selezione di principi di software design ritenuti essenziali per garantire un livello adeguato di sicurezza. Verranno, inoltre, presentati esempi di Markov Chain, relativi a un'unità del sistema, e uno riguardante un monitor di Runtime Enforcement.

4.1 Catalogo Saltzer e Schroeder

Il primo catalogo consultato è quello di *Saltzer e Schroeder (1975)*, dal quale sono stati selezionati i seguenti principi:

- **Psychological acceptability:** il software è progettato per essere accessibile anche a utenti con competenze tecnologiche limitate. Per questo motivo, l'interfaccia del programma deve garantire un'esperienza *user-friendly*, ovvero facile da comprendere e utilizzare anche per chi non ha conoscenze avanzate.
- **KISS principle ("Keep It Simple, Stupid"):** ogni componente del software deve offrire unicamente le funzionalità per cui è stato progettato. L'implementazione deve essere la più semplice possibile, per ridurre la possibilità di introdurre vulnerabilità non necessarie.
- **Defense in depth:** il software implementerà un sistema di autenticazione robusto, al fine di garantire la protezione del sistema e l'integrità dei dati degli utenti, fornendo più livelli di sicurezza.
- **Open design:** la sicurezza del software non deve dipendere dal segreto del codice sorgente. Un meccanismo di protezione non deve basarsi sull'ignoranza dell'attaccante riguardo al funzionamento del sistema.

4.2 Catalogo Sommerville

Il secondo catalogo consultato è quello di *Sommerville (2017)*. I principi selezionati da questo catalogo sono i seguenti:

- **Log user actions:** per prevenire possibili tentativi di violazione da parte di attori malintenzionati, ogni azione compiuta dagli utenti sul sistema dovrà essere registrata. Questo garantirà una tracciabilità completa delle operazioni, permettendo di rilevare attività sospette o non autorizzate.
- **Avoid a single point of failure:** il sistema dovrà essere progettato in modo tale da evitare il rischio di un singolo punto di fallimento, minimizzando il rischio che una vulnerabilità isolata possa compromettere l'intero sistema.
- **Least common mechanism:** le interazioni tra i vari asset, che operano in modo indipendente, devono avvenire solo quando strettamente necessario. Questo principio aiuta a ridurre la superficie di attacco e a limitare la possibilità di esposizione del sistema a rischi non necessari.

4.3 Catalogo OWASP

L'ultimo catalogo consultato è quello di *OWASP (2016)*, dal quale sono stati estratti i seguenti principi:

- **Don't trust services:** è fondamentale assicurarsi che ogni servizio esterno, con cui il software interagisce, sia sicuro e non compromesso. È necessario valutare continuamente i servizi per prevenire vulnerabilità che potrebbero minare la sicurezza complessiva del sistema.
- **Least privilege:** ogni utente deve avere un livello di accesso ridotto ai minimi privilegi necessari per svolgere le proprie attività. Questo principio previene movimenti orizzontali¹ e minimizza i rischi di abuso dei privilegi.
- **Fail securely:** quando il sistema fallisce, deve farlo in modo sicuro, garantendo che eventuali errori non vengano sfruttati dagli attaccanti per compromettere la sicurezza o l'integrità del software.

4.4 Esempio di analisi di un componente del sistema mediante Markov Chain

In questa sezione vengono presentati i risultati dell'analisi di un'unità del sistema utilizzando le catene di Markov. La modellazione è stata eseguita per rappresentare il comportamento del sistema, in modo da studiarne le probabilità di transizione tra diversi stati e di prevedere il comportamento futuro sulla base di scenari specifici.

In particolare, l'analisi si concentrerà su due aspetti fondamentali:

- **Safety:** La proprietà di sicurezza riguarda la capacità del sistema di evitare situazioni indesiderate o pericolose. In questo contesto, si esamina la probabilità che il sistema rimanga in uno stato sicuro nel tempo, evitando transizioni che possano portare a un fallimento o a una compromissione del sistema.
- **Response:** La proprietà di risposta è un'analisi che esamina la capacità del sistema di rispondere tempestivamente a determinati eventi o azioni. Questa proprietà si concentra sulla velocità con cui il sistema raggiunge uno stato desiderato o risponde a stimoli esterni, garantendo un comportamento efficiente.

L'analisi è stata realizzata utilizzando il linguaggio formale *PRISM*, uno strumento per la modellazione e l'analisi delle probabilità in sistemi stocastici, che consente di verificare le proprietà di safety e response tramite tecniche probabilistiche avanzate. Questo approccio permette di ottenere risultati concreti per migliorare la progettazione e la sicurezza del sistema.

4.4.1 Markov Chain

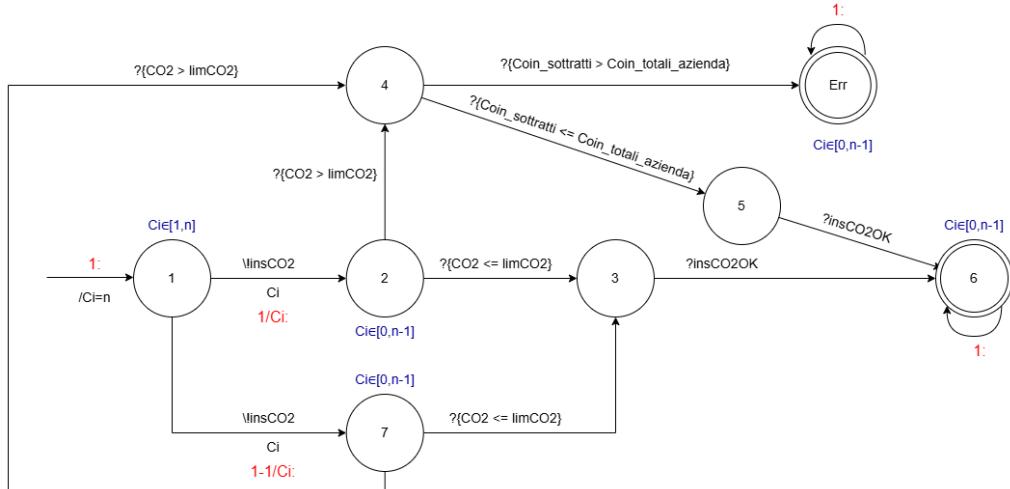
La Markov Chain è un modello matematico che descrive il comportamento di un sistema attraverso una sequenza di stati. Questo approccio consente di analizzare i percorsi di transizione tra gli stati e di calcolare le probabilità di passaggio da uno stato all'altro.

Per le analisi successive, è stata presa in considerazione una particolare unità operativa del software progettato, relativa al processo di monitoraggio e gestione delle emissioni di CO₂. Il diagramma sottostante rappresenta una simulazione del processo di inserimento di CO₂, illustrando le transizioni tra i vari stati in base all'andamento delle emissioni².

Nella Figura 4.1, viene mostrato il processo di inserimento di CO₂ da parte dell'utente nel sistema. Inizialmente, lo stato 1 rappresenta lo stato di partenza, da cui l'utente può avviare

¹Per "movimenti orizzontali" si intende l'uso dei permessi di un utente per compiere azioni non autorizzate che esulano dalle proprie funzioni.

²Per facilitare la lettura del grafico, gli elementi in nero rappresentano le etichette associate a ciascuna transazione, le probabilità di transizione sono indicate in rosso, mentre in blu sono evidenziati gli eventuali tentativi di inserimento.


 Figure 4.1: Markov Chain relativa all'inserimento di CO₂

il processo di registrazione delle emissioni. Il sistema è progettato per monitorare le emissioni di CO₂, assicurandosi che rimangano sotto un determinato standard. Se la condizione di emissione viene rispettata, l'utente ottiene un Bonus, cioè un numero di coin proporzionale alla quantità di CO₂ risparmiata, rappresentato nel diagramma dal passaggio dallo stato 2 allo stato 3.

Nel caso in cui il requisito di emissione non venga rispettato, l'utente incorrerà in un Malus, che comporta la sottrazione di coin. Se il numero di coin sottratti è inferiore a quello attualmente posseduto, il sistema raggiunge uno stato finale (**stato 6**). In caso contrario, l'utente finirà nello stato di errore.

Un aspetto importante del modello è che l'utente può effettuare diversi tentativi di inserimento, ciascuno dei quali può portare a esiti differenti a seconda della situazione corrente dell'organizzazione. Questo è rappresentato dagli stati 2 e stato 7, che evidenziano le varie casistiche possibili. Infatti, le probabilità di transizione verso lo stato 2 o lo stato 7 sono complementari, riflettendo diverse probabilità di successo o fallimento nel processo di inserimento dei dati.

4.4.2 Implementazione del Modello in PRISM

PRISM è uno strumento open-source per la modellazione, la verifica e l'analisi di sistemi stocastici, particolarmente utile per studiare sistemi complessi che presentano incertezze, come quelli utilizzati nella sicurezza informatica. Questo strumento consente di eseguire analisi formali e calcoli probabilistici, che sono fondamentali per comprendere il comportamento di sistemi che dipendono da eventi casuali o variabili stocastiche.

Nella sezione seguente, viene fornito il codice PRISM³ che rappresenta la Markov Chain descritta nel paragrafo 4.4.1. Questo codice permette di eseguire simulazioni e analisi probabilistiche per valutare le transizioni tra gli stati e le probabilità di ciascun evento all'interno del sistema.

```

1 dtmc
2
3 const int n = 10; // Numero di tentativi
    
```

³Per quanto riguarda le variabili globali e locali, sono stati inseriti dei valori fittizi, utilizzati esclusivamente per testare il processo di modellazione.

```

4 const int limCO2 = 70;           // Limite massimo di CO2
5
6
7 module ins_co2
8
9     stato : [1..10] init 1;      // Stato iniziale
10    ci : [0..n] init n;         // Numero di tentativi iniziali
11    CO2 : [1..limCO2+100] init 80; // Variabile CO2
12    Coin_sottratti : [0..30] init 30; //Coin sottratti in caso di Malus
13    Coin_azienda : [0..20] init 20; // Coin totali dell'azienda
14
15    // Stato iniziale: tentativo di inserire CO2
16    [insCO2] (stato=1) & (ci>0) -> (1/ci) : (stato'=2) & (ci'=ci-1) + (1-1/
17    ci) : (stato'=7) & (ci'=ci-1);
18
19    // Bonus: Inserimento CO2 OK
20    [bonus] (stato=2) & (CO2<=limCO2) -> (stato'=3); // Successo se CO2 <
21    = limCO2
22    [bonus] (stato=7) & (CO2<=limCO2) -> (stato'=3); // Successo se CO2 <
23    = limCO2
24
25    // Malus: Inserimento CO2 KO
26    [malus] (stato=2) & (CO2>limCO2) -> (stato'=4); // Malus se CO2 >
27    limCO2
28    [malus] (stato=7) & (CO2>limCO2) -> (stato'=4); // Malus CO2 > limCO2
29
30    // Transizione a stato di successivo tentativo
31    [OK] (stato=3) -> (stato'=6); //Raggiunto stato finale
32    [OK] (stato=4) & (Coin_sottratti <= Coin_azienda)-> (stato'=5); //
33    Raggiunto stato finale
34    [OK] (stato=5) -> (stato'=6); // Raggiunto stato finale
35
36    // Errore
37    [Err] (stato=4) & (Coin_sottratti > Coin_azienda) -> (stato'=8); //
38    Transizione a stato 8 per errore
39
40 endmodule
41
42 // Etichette per stato finale
43 label "bonus" = (stato=3); // Etichetta bonus: stato 3
44 label "malus" = (stato=4); // Etichetta malus: stato 4
45 label "errore" = (stato=8); // Etichetta stato di errore: stato Err
46 label "finale" = (stato=6); // Etichetta stato finale di successo

```

Utilizzando la sezione *Simulator* di PRISM, è stato possibile osservare il comportamento dinamico del sistema. Nel primo scenario, illustrato nella Figura 4.2, il livello di CO2 è stato impostato a un valore di 10, inferiore al limite limCO2 fissato a 70. Come evidenziato nell'immagine, il sistema riesce a raggiungere senza difficoltà lo stato di *Bonus*, consentendo all'utente di ottenere i coin previsti, in conformità con le condizioni stabilite.

Il caso successivo, mostrato nella Figura 4.3, descrive una situazione in cui il valore di CO2, fissato a 80, supera il limite limCO2 stabilito a 70. In questo scenario, il sistema transita nello stato di *Malus*. Di conseguenza, il sistema detrae un numero specifico di coin dall'azienda, la quale dispone di una quantità patrimoniale sufficiente per coprire tale sottrazione.

4.4 Esempio di analisi di un componente del sistema mediante Markov Chain 51

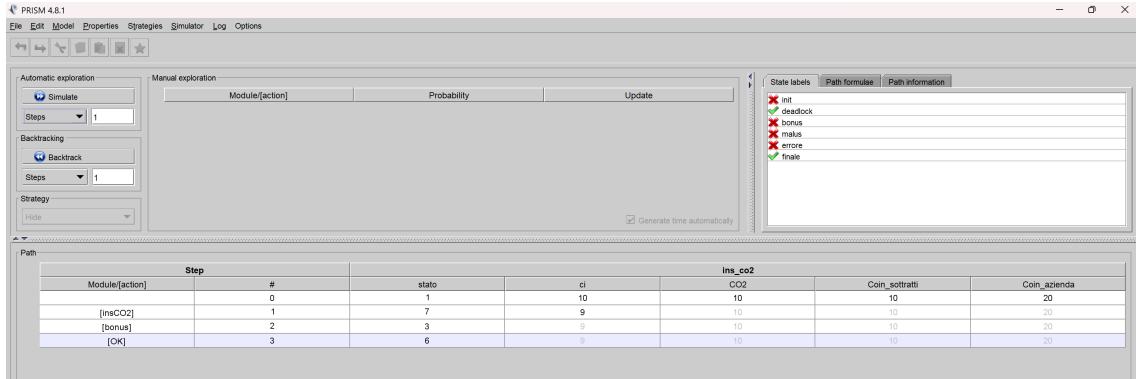


Figure 4.2: Simulazione del sistema nel caso in cui il livello di CO₂ sia minore o uguale del limite di CO₂

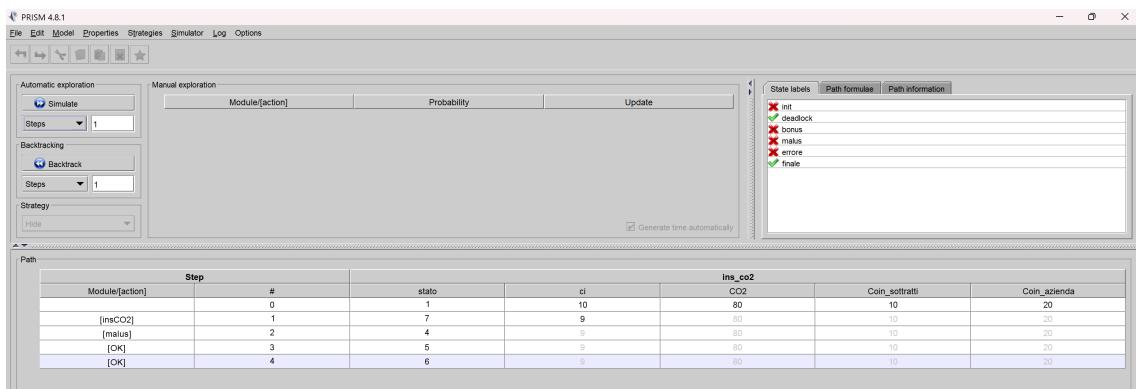


Figure 4.3: Simulazione del sistema nel caso in cui il livello di CO₂ sia maggiore del limite di CO₂

Nel caso illustrato nella Figura 4.4, che riguarda ancora una situazione in cui il valore di CO₂ supera limCO₂, si verifica una condizione diversa: i coin posseduti dall'azienda non sono sufficienti a coprire l'importo sottratto dal sistema. In questo scenario, l'utente si ritrova nello stato di errore in cui è necessario, nell'applicazione, inviare una richiesta per ottenere coin aggiuntivi da una seconda organizzazione.

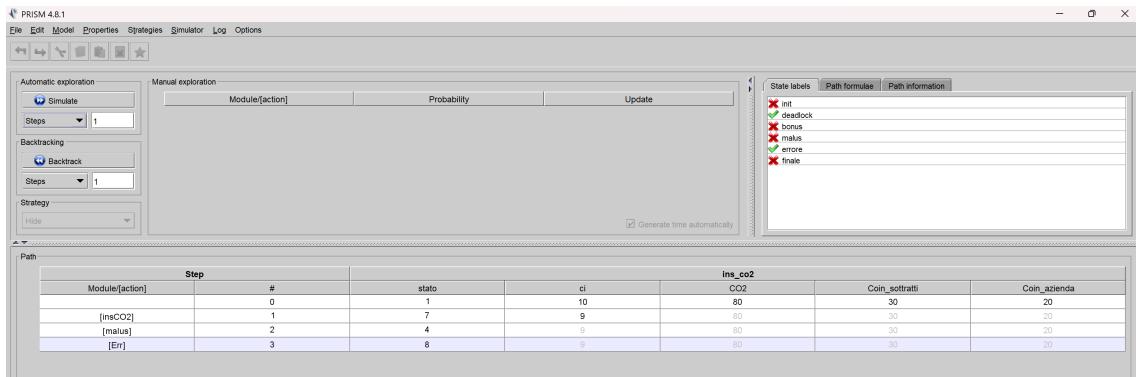


Figure 4.4: Simulazione del sistema nel caso in cui il livello di CO₂ sia maggiore del limite di CO₂ (Stato di errore)

A seguito dell'implementazione del modello PRISM, nei paragrafi successivi sono state verificate due proprietà: una relativa alla **Safety** e l'altra alla **Response**.

4.4.3 Verifica delle proprietà di Safety

Per quanto riguarda la proprietà di **Safety**, è stato analizzato il seguente aspetto:

"Tutti i dati devono essere inseriti correttamente nel sistema".

Qualora i dati, in particolare quelli relativi alla CO₂, siano inseriti correttamente, il sistema sarà in grado di assegnare un *Bonus* o un *Malus* all'utente. In tale scenario, il sistema potrà raggiungere, a seconda dei risultati, lo **stato 3**, corrispondente all'ottenimento del *Bonus*, o lo **stato 4**, associato al *Malus*.

La stessa proprietà viene successivamente espressa nel linguaggio PRISM nella seguente formula (Formula 4.1).

$$P = ?[F(stato = 3 \mid stato = 4)] \quad (4.1)$$

Tramite lo strumento di verifica, è stato possibile visualizzare la probabilità con cui questa proprietà viene rispettata. La probabilità risulta essere pari a **1.0**, indicativo del fatto che l'inserimento di CO₂ nel sistema avviene con successo in tutti i casi analizzati.

4.4.4 Verifica della proprietà di Response

Per quanto riguarda la proprietà di **Response**, è stato preso in considerazione il seguente aspetto:

"Ogni volta che viene risparmiata CO₂, l'organizzazione ottiene un coin."

Questa proprietà è tradotta in PRISM attraverso la Formula 4.2. Come risulta evidente da questa formulazione, ogni volta che il livello di CO₂ è minore o uguale allo standard fissato, il sistema raggiungerà uno stato di *Bonus*, per poi proseguire verso lo **stato 9**, che rappresenta la conclusione del processo.

$$P = ?[G(F(CO2 \leq limCO2 \& stato = 9))] \quad (4.2)$$

Per quanto riguarda la verifica di questa proprietà, è necessario distinguere due casi:

1. *CO2 ≤ limCO2*: In questo caso, la probabilità di raggiungere lo stato di *Bonus* è pari a **1.0**, poiché il sistema soddisfa la condizione e assegna i coin all'utente.
2. *CO2 > limCO2*: In questo scenario, la proprietà di *Response* non viene verificata, poiché il sistema transita nello stato di *Malus*⁴.

4.5 Esempio di sintesi di un monitor di Runtime Enforcement

I monitor di *Runtime Enforcement* (*RE*) sono strumenti utilizzati per monitorare l'esecuzione di un sistema in tempo reale, assicurando che vengano rispettate le specifiche di sicurezza o correttezza. Nel contesto del *RE*, se il monitor rileva un comportamento non conforme, può intervenire per fermare il sistema e forzare il rispetto della policy di sicurezza definita.

⁴Anche se in questo caso non si ottengono coin direttamente, va sottolineato che l'utente può comunque riceverli tramite una richiesta a una seconda organizzazione, se i coin sottratti sono superiori alla disponibilità dell'azienda.

4.5.1 Esempio di RE di una proprietà di Safety

Per il monitor di RE relativo alla proprietà di Safety, ci si è basati sulla proprietà analizzata nella Sezione 4.4.3. In questo caso, il sistema Σ riceve in input due proposizioni, denotate come a e b , che vengono rappresentate attraverso la *Legenda* riportata in Figura 4.5. La proprietà, espressa in linguaggio naturale, può essere formalizzata come $P = a^{\omega}$.

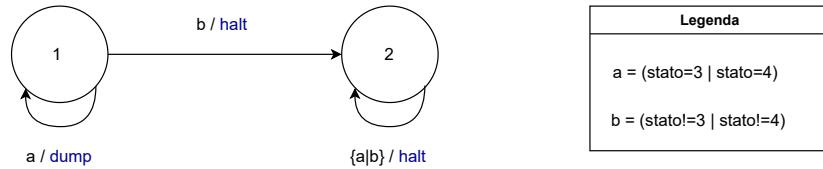


Figure 4.5: Monitor di RE relativo alla proprietà di Safety

Fino a quando la proprietà viene rispettata, il sistema continua a effettuare il **dump** dei dati, rimanendo nello **stato 1**. Tuttavia, non appena la policy di sicurezza viene violata, il sistema passa allo **stato 2**, fermando l'esecuzione del processo. In questo stato, il software non memorizza nuovi eventi né produce output.

4.5.2 Esempio di RE di una proprietà di Response

Per quanto riguarda la proprietà di Response, si è deciso di fare riferimento alla stessa proprietà esaminata nella Sezione 4.4.4. Il sistema Σ riceve in input due proposizioni, a e b , come spiegato nella *Legenda* riportata in Figura 4.6. La proprietà, espressa in linguaggio naturale, può essere formalizzata come $P = (a \cdot b \cdot b \cdot a)^\omega$.

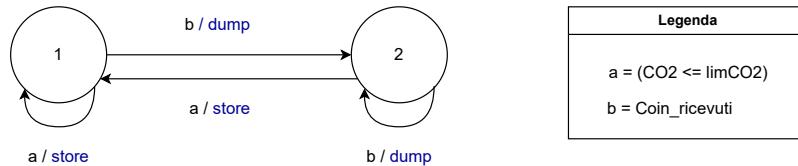
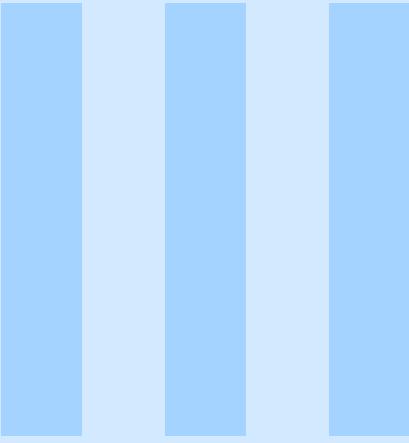


Figure 4.6: Monitor di RE relativo alla proprietà di Safety

Come si può osservare dal diagramma in Figura 4.6, una volta che la condizione a è rispettata, il sistema risponde eseguendo un *dump* dei coin verso l'esterno. Quando si verifica un nuovo inserimento di CO₂ e la condizione della proprietà è nuovamente soddisfatta, il sistema ritorna allo **stato 1**, consentendo al software di replicare il processo.



Sviluppo del sistema e Manuale d'uso

5	Implementazione	56
5.1	Architettura	
5.2	Parte off-chain	
5.3	Parte on-chain	
6	Manuale d'utilizzo	67
6.1	Manuale d'utilizzo	
6.2	Livello 0	
6.3	Login e Registrazione	
6.4	Livello 1	
6.5	Livello 2	

5. Implementazione

5.1 Architettura

In questo capitolo, viene illustrata l'architettura dell'applicazione sviluppata, con un focus sulle scelte progettuali adottate per garantire sicurezza, modularità ed efficienza.

L'architettura si basa su **Flask** per gestire l'interfaccia API e la comunicazione tra i nodi, assicurando un'interazione fluida ed efficiente con la blockchain. Per lo sviluppo e la gestione degli smart contract, è stato scelto **Hyperledger Besu**, mentre l'interfacciamento con il registro distribuito è stato reso possibile tramite **Web3**.

L'applicazione adotta un'architettura ibrida, come descritto nel Capitolo 2, Sezione 2.1, suddivisa in due componenti principali: una on-chain e una off-chain.

Nel processo di deployment sono stati utilizzati **Nginx** e **Gunicorn** per ottimizzare la gestione del traffico web e le performance complessive del sistema. **Gunicorn** è un server **WSGI**^a che esegue l'applicazione Flask e gestisce le richieste HTTP, indirizzandole correttamente all'applicazione. Sebbene Gunicorn sia altamente efficiente nel gestire il carico delle richieste, da solo potrebbe non essere sufficiente a gestire volumi elevati di traffico. Per questo motivo, è stato introdotto **Nginx**, che funge da reverse proxy: riceve le richieste in ingresso e le instrada verso Gunicorn. Inoltre, Nginx è particolarmente indicato per la gestione di contenuti statici, come immagini e file CSS/JS, e per ottimizzare le performance tecniche come la compressione dei dati. In questo modo, Gunicorn si concentra sull'esecuzione dell'applicazione Flask, mentre Nginx si occupa della gestione del traffico e dell'ottimizzazione generale del sistema.

^aWSGI, che sta per Web Server Gateway Interface, è uno standard di interfaccia tra i server web e le applicazioni Python

5.1.1 Componenti strutturali

Flask

applicazioni web in modo semplice e veloce. Utilizzando Flask, è possibile configurare un server web in grado di caricare template HTML e renderizzarli dinamicamente utilizzando la sintassi di template **Jinja2**. Questo approccio consente di separare la logica di presentazione dalla logica dell'applicazione, migliorando la modularità e la manutenibilità del codice.

Hyperledger Besu

Hyperledger Besu è una piattaforma blockchain open source, progettata per offrire un ambiente sicuro, robusto e ad alte prestazioni per la creazione di applicazioni decentralizzate (*dApp*) e smart contract. Besu supporta sia reti pubbliche che private, ed è completamente compatibile con l'ecosistema di **Ethereum**, inclusi la *Ethereum Virtual Machine* (EVM) e il linguaggio di programmazione **Solidity**. Grazie a queste caratteristiche, Besu consente agli

sviluppatori di sfruttare appieno le potenzialità delle blockchain pubbliche, pur mantenendo la possibilità di costruire soluzioni private per ambienti aziendali e casi d'uso specifici.

Docker

Docker è una piattaforma open source per la containerizzazione che consente di impacchettare un'applicazione insieme a tutte le sue dipendenze in un'unità standardizzata chiamata container. I container sono leggeri e isolati dall'infrastruttura sottostante, nonché da altri container, il che facilita la loro portabilità e l'esecuzione su diverse piattaforme. Le immagini Docker possono essere eseguite come container su qualsiasi macchina che abbia Docker installato, assicurando la stessa performance e compatibilità, indipendentemente dal sistema operativo.

Docker Compose è uno strumento che consente di gestire applicazioni multi-contenitore, utilizzando un file di configurazione in formato YAML. Il file YAML descrive tutte le configurazioni necessarie per distribuire e orchestrare i contenitori, ciascuno dei quali è progettato per essere eseguito su un singolo host.

5.1.2 L'interazione dei componenti

Nell'applicazione è stato utilizzato **Docker** per eseguire **Flask** e **Besu** come container separati ma interconnessi, garantendo la coerenza tra la parte *off-chain* e *on-chain* del sistema. Flask gestisce la logica applicativa e la comunicazione con il database **PostgreSQL**, mentre Besu, eseguito in un container separato, si occupa delle transazioni sulla blockchain. La rete interna Docker consente ai due container di comunicare in modo sicuro, permettendo a Flask di inviare e ricevere transazioni dalla blockchain Besu e di mantenere la sincronizzazione dei dati tra il sistema *off-chain* e *on-chain*.

5.2 Parte off-chain

La parte *off-chain* del sistema è stata containerizzata utilizzando il framework web **Flask** e il database relazionale **PostgreSQL**. PostgreSQL è stato scelto per la sua robustezza, scalabilità e capacità di gestire grandi volumi di dati in modo efficiente, offrendo una solida base per la gestione delle informazioni e delle transazioni non legate direttamente alla blockchain.

5.2.1 Database

Per stabilire la connessione tra l'applicazione **Flask** e il database **PostgreSQL**, è stato utilizzato SQLAlchemy come ORM, semplificando la definizione delle tabelle e la gestione delle relazioni tra esse. È stato adottato il pattern **Singleton** per garantire che esista un'unica istanza della connessione al database in tutta l'applicazione. Inoltre, le credenziali e le configurazioni del database vengono gestite tramite variabili d'ambiente, caricate utilizzando `dotenv`, per migliorare la sicurezza e la configurabilità del sistema.

Nella Figura 5.1 è riportata la struttura del database utilizzato per il software.

La base di dati include diverse tabelle interconnesse che gestiscono le informazioni principali del sistema. La tabella **Oracle** gestisce gli utenti amministratori, le cui credenziali di accesso sono protette tramite hashing delle password. La tabella **Employer** rappresenta i dipendenti delle organizzazioni, identificati da username ed email univoci. La tabella **Organization** definisce le organizzazioni, contenendo informazioni come nome, ragione sociale, partita IVA, indirizzo e contatti. Ogni organizzazione ha un tipo specifico (agricoltore, venditore, produttore o trasportatore) e uno stato (attivo o inattivo). Inoltre, include un sistema di crediti chiamato **coin**, utilizzato per le transazioni.

La tabella **Product** gestisce i prodotti, con informazioni su nome, tipologia (materia prima o prodotto finito), quantità disponibile e il livello di emissioni di CO₂ associato alla

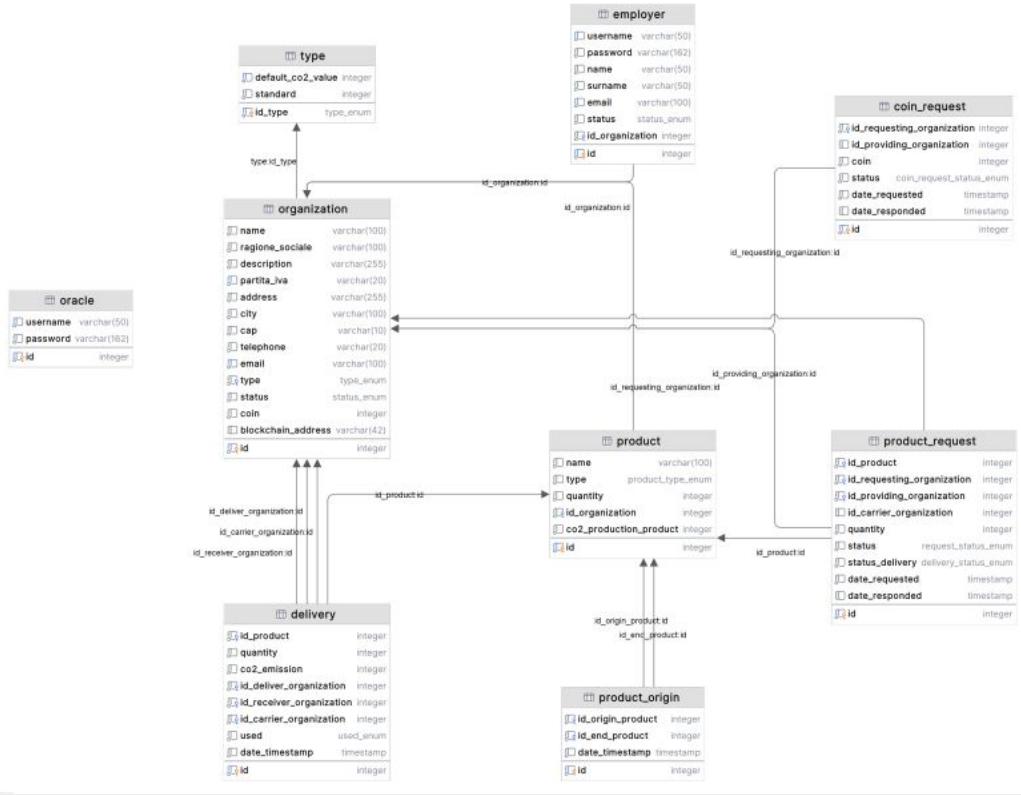


Figure 5.1: Database

loro produzione. La tabella `ProductRequest` registra le richieste di prodotti, indicando il richiedente, il fornitore, l'eventuale trasportatore e lo stato della richiesta e della consegna.

La tabella `CoinRequest` traccia le transazioni di `coin` tra le organizzazioni, gestendo le richieste di crediti. La tabella `Delivery` registra le consegne effettuate, specificando prodotto, quantità trasportata, emissioni di CO₂ generate e le organizzazioni coinvolte nel processo logistico. Infine, la tabella `ProductOrigin` traccia la provenienza dei prodotti, registrando le relazioni tra prodotti di origine e prodotti finiti.

5.2.2 Frontend

Nel progetto, per la parte di frontend, sono stati utilizzati **HTML** e **CSS**, che sono stati integrati con **Flask**. Un aspetto fondamentale nella gestione dei form è stato l'uso delle **FlaskForm**. Questa funzionalità è stata utilizzata frequentemente per facilitare la creazione e la validazione dei `form`, semplificando così la gestione dell'interazione con l'utente. **FlaskForm** gestisce automaticamente i token `CSRF`. Quando si utilizza una `FlaskForm`, ogni modulo **HTML** generato includerà un campo nascosto con un token `CSRF`¹. Questo token viene convalidato al momento della ricezione della richiesta `POST`, impedendo attacchi `CSRF`.

¹Il `CSRF` (Cross-Site Request Forgery) è un tipo di attacco informatico in cui un malintenzionato sfrutta l'autenticazione di un utente su un sito web per inviare richieste non autorizzate a un altro sito web in cui l'utente è autenticato. In altre parole, l'attaccante induce l'utente a eseguire azioni indesiderate su un'applicazione web in cui è già loggato, senza il suo consenso.

5.3 Parte on-chain

Per la parte on-chain, è stata integrata la blockchain **Hyperledger Besu** che offre un'implementazione conforme agli standard **Ethereum**. Inoltre, è stato utilizzato **Solidity** per sviluppare gli smart contract, consentendo l'automazione e l'esecuzione sicura delle transazioni sulla blockchain. Per l'interazione con la blockchain, gli utenti possono utilizzare **MetaMask**, un portafoglio digitale che facilita la gestione e l'autenticazione delle transazioni in modo sicuro e user-friendly.

5.3.1 Blockchain

Per collegare Flask a Hyperledger Besu utilizzando **Web3**, si utilizza un'architettura client-server in cui Flask funge da interfaccia API per interagire con la blockchain.

Hyperledger Besu espone un'interfaccia **RPC** (Remote Procedure Call) che consente la comunicazione con i nodi della rete tramite **JSON-RPC**. Flask, attraverso Web3, invia richieste a questa interfaccia per eseguire operazioni sulla blockchain, come la lettura dello stato della rete, l'invio di transazioni e l'interazione con gli smart contract. Nella configurazione utilizzata per questo progetto, sono presenti quattro nodi validatori (coloro che hanno il permesso di mettere i blocchi sulla blockchain) e tre nodi membri.

Il server Flask riceve le richieste HTTP dai client, le elabora e utilizza Web3 per connettersi all'endpoint RPC di Besu. Dopo aver ottenuto la risposta dal nodo blockchain, Flask la restituisce al client in formato JSON, consentendo così un'integrazione trasparente tra l'applicazione e la rete decentralizzata.

Tessera è un transaction manager open-source sviluppato per gestire transazioni private in blockchain permissioned. Si occupa di cifrare e distribuire in modo sicuro i dati delle transazioni, garantendo che solo i nodi autorizzati possano accedervi.

In un'architettura basata su Docker, ogni nodo Besu è affiancato da un nodo Tessera, e i due comunicano per gestire la privacy delle transazioni. I nodi Tessera si scambiano informazioni cifrate su una rete interna, mantenendo la riservatezza delle operazioni senza esporle all'intera blockchain.

Nella Figura 5.2 viene mostrato **Quorum Explorer**, un'interfaccia web per monitorare e analizzare lo stato di una blockchain, utilizzato solo a scopi dimostrativi durante lo sviluppo.

Quorum Explorer fornisce una panoramica dell'intera rete, come le informazioni sui blocchi, votazioni o rimozione dei validatori dalla rete. Inoltre, dimostra l'utilizzo dello **SimpleStorage** smart contract con privacy abilitata e l'invio di transazioni tra **wallet** in un'unica interfaccia. La pagina **Nodes** (Figura 5.3) fornisce una panoramica dei nodi sulla rete.

5.3.2 Smart contract

Nella directory del progetto si può notare la cartella **contract** in cui all'interno vi è tutta la parte relativa alla blockchain. Al suo interno è presente il file **CoinContract.sol** in cui c'è lo smart contract sviluppato in **Solidity**. Il file **deploy.py** sviluppato in Python, avvia il contratto prima che si avvii la parte off-chain, così da assicurare il funzionamento corretto dell'applicazione.

Lo smart contract **CoinContract** 5.1 include funzioni per la gestione dei bilanci e delle transazioni, come l'aggiornamento delle monete (*coin*) in base alle emissioni di CO₂ e la registrazione delle origini dei prodotti. Permette inoltre di trasferire monete tra organizzazioni e di registrare transazioni relative all'origine dei prodotti. Oltre a ciò, registra anche le transazioni *rifiutate*, specificando le motivazioni per cui non sono andate a buon fine. Ad esempio, una transazione può essere rifiutata a causa dell'insufficienza di *coin* necessari per accettare un livello di emissioni di CO₂ superiore alla norma durante la registrazione di un prodotto.

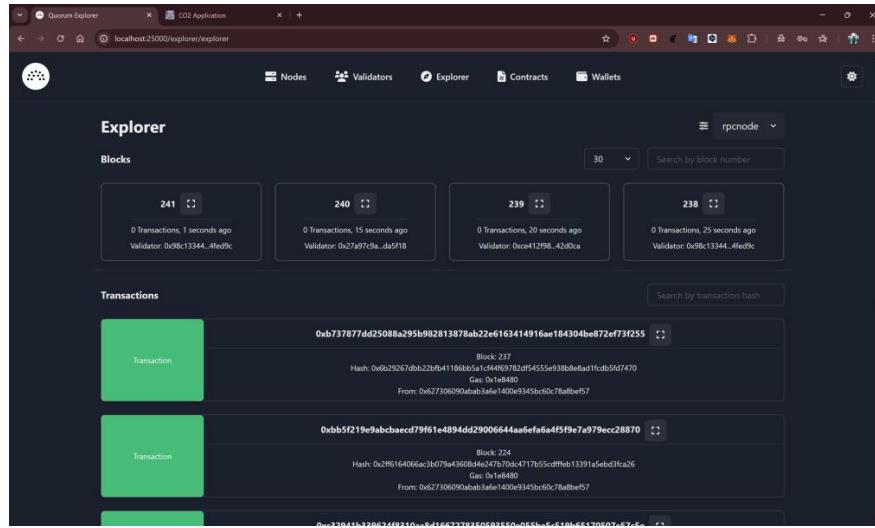


Figure 5.2: Quorum Explorer

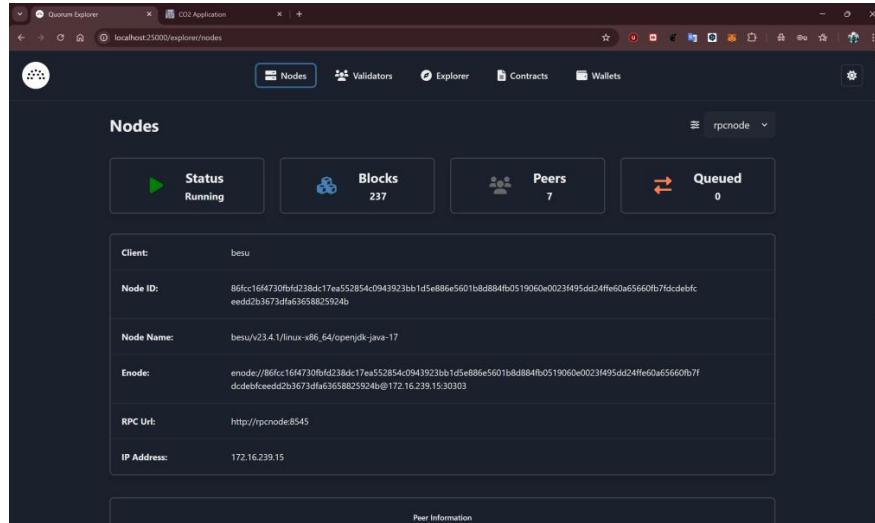


Figure 5.3: Quorum Explorer Nodes

Scendendo nel dettaglio, le transazioni vengono memorizzate e possono essere consultate, mentre gli eventi vengono emessi per notificare aggiornamenti o registrazioni.

```
1 // SPDX-License-Identifier: UNLICENSED
2 pragma solidity ^0.8.0;
3
4 /// @title CoinContract
5 /// @notice This contract manages the balances of organizations and allows
6 ///         updating their coin balances.
7 /// @dev Follows the EthTrust Security Levels and Solidity Style Guide.
8
9 contract CoinContract {
10     /// @notice Mapping to store the balances of organizations.
11     mapping(address => uint256) public balances;
12
13     /// @notice Struct to store transaction details.
14     struct Transaction {
15         address organization;
16         int256 amount;
17         uint256 timestamp;
18         bytes32 txHash;
19     }
20
21     /// @notice Struct to store product origin transaction details.
22     struct ProductOriginTransaction {
23         uint256 originProduct;
24         uint256 endProduct;
25         uint256 timestamp;
26     }
27
28     /// @notice Struct to store rejected transaction details.
29     struct RejectedTransaction {
30         address organization;
31         int256 amount;
32         uint256 timestamp;
33         string reason;
34         string productName;
35         uint256 productQuantity;
36         uint256 co2Emission;
37         bytes32 txHash;
38     }
39
40     /// @notice Array to store all transactions.
41     Transaction[] public transactions;
42
43     /// @notice Array to store all product origin transactions.
44     ProductOriginTransaction[] public productOriginTransactions;
45
46     /// @notice Array to store all rejected transactions.
47     RejectedTransaction[] public rejectedTransactions;
48
49     /// @notice Event emitted when the coin balance of an organization is
50     ///         updated.
51     /// @param organization The address of the organization.
52     /// @param newBalance The new balance of the organization.
53     event CoinsUpdated(address indexed organization, uint256 newBalance);
54
55     /// @notice Event emitted when a product origin is registered.
56     /// @param originProduct The ID of the origin product.
57     /// @param endProduct The ID of the end product.
58     event ProductOriginRegistered(uint256 indexed originProduct, uint256
59         indexed endProduct);
```



```

111     uint256 bonusCoin = co2Limit - co2Emission;
112     newBalance = balances[organization] + bonusCoin;
113     amount = int256(bonusCoin);
114 }
115 balances[organization] = newBalance;
116
117 // Memorizza la transazione
118 bytes32 txHash = keccak256(abi.encodePacked(block.number,
119 organization, amount, block.timestamp));
120 transactions.push(Transaction({
121     organization: organization,
122     amount: amount,
123     timestamp: block.timestamp,
124     txHash: txHash
125 }));
126
127     emit CoinsUpdated(organization, newBalance);
128 }
129
130 /// @notice Transfers coins from one organization to another.
131 /// @param from The address of the organization sending the coins.
132 /// @param to The address of the organization receiving the coins.
133 /// @param amount The amount of coins to transfer.
134 /// @dev The transaction hash is generated using the block number,
135 organization address, amount, and timestamp.
136 function transferCoins(address from, address to, uint256 amount) public
137 {
138     require(from != address(0), "Invalid sender address");
139     require(to != address(0), "Invalid recipient address");
140     require(balances[from] >= amount, "Insufficient balance");
141
142     balances[from] -= amount;
143     balances[to] += amount;
144
145     // Memorizza la transazione per il mittente
146     bytes32 txHashFrom = keccak256(abi.encodePacked(block.number, from,
147 -int256(amount), block.timestamp));
148     transactions.push(Transaction({
149         organization: from,
150         amount: -int256(amount),
151         timestamp: block.timestamp,
152         txHash: txHashFrom
153 }));
154
155     // Memorizza la transazione per il destinatario
156     bytes32 txHashTo = keccak256(abi.encodePacked(block.number, to,
157 int256(amount), block.timestamp));
158     transactions.push(Transaction({
159         organization: to,
160         amount: int256(amount),
161         timestamp: block.timestamp,
162         txHash: txHashTo
163 }));
164
165     emit CoinsUpdated(from, balances[from]);
166     emit CoinsUpdated(to, balances[to]);
167 }
168
169 /// @notice Registers a product origin.
170 /// @param originProduct The ID of the origin product.
171 /// @param endProduct The ID of the end product.

```

```

167     function registerProductOrigin(uint256 originProduct, uint256
168 endProduct) public {
169     require(originProduct != 0, "Invalid origin product ID");
170     require(endProduct != 0, "Invalid end product ID");
171
172     // Memorizza la transazione di origine del prodotto
173     productOriginTransactions.push(ProductOriginTransaction({
174         originProduct: originProduct,
175         endProduct: endProduct,
176         timestamp: block.timestamp
177     }));
178
179     // Emit the event
180     emit ProductOriginRegistered(originProduct, endProduct);
181 }
182
183     /// @notice Registers a rejected transaction.
184     /// @param organization The address of the organization.
185     /// @param amount The amount of the rejected transaction.
186     /// @param reason The reason for the rejection.
187     /// @param productName The name of the rejected product.
188     /// @param productQuantity The quantity of the rejected product.
189     /// @param co2Emission The CO2 emission of the rejected product.
190     /// @dev The transaction hash is generated using the block number,
191     /// organization address, amount, and timestamp.
192     function registerRejectedTransaction(address organization, int256
193 amount, string memory reason, string memory productName, uint256
194 productQuantity, uint256 co2Emission) public {
195     require(organization != address(0), "Invalid address");
196
197     // Genera il txHash
198     bytes32 txHash = keccak256(abi.encodePacked(block.number,
199 organization, amount, block.timestamp));
200
201     // Memorizza la transazione rifiutata
202     rejectedTransactions.push(RejectedTransaction({
203         organization: organization,
204         amount: amount,
205         timestamp: block.timestamp,
206         reason: reason,
207         productName: productName,
208         productQuantity: productQuantity,
209         co2Emission: co2Emission,
210         txHash: txHash
211     }));
212
213     emit TransactionRejected(organization, amount, reason, productName,
214 productQuantity, co2Emission, txHash);
215 }
216
217     /// @notice Returns the coin balance of an organization.
218     /// @param organization The address of the organization.
219     /// @return The balance of the organization.
220     function getBalance(address organization) public view returns (uint256)
221     {
222         return balances[organization];
223     }
224
225     /// @notice Returns the transactions of an organization.
226     /// @param organization The address of the organization.
227     /// @return Arrays of transaction details.

```

```
221     function getTransactions(address organization) public view returns (
222         address[] memory, int256[] memory, uint256[] memory, bytes32[] memory)
223     {
224         uint256 count = 0;
225         for (uint256 i = 0; i < transactions.length; i++) {
226             if (transactions[i].organization == organization) {
227                 count++;
228             }
229         }
230
231         address[] memory orgs = new address[](count);
232         int256[] memory amounts = new int256[](count);
233         uint256[] memory timestamps = new uint256[](count);
234         bytes32[] memory txHashes = new bytes32[](count);
235         uint256 index = 0;
236         for (uint256 i = 0; i < transactions.length; i++) {
237             if (transactions[i].organization == organization) {
238                 orgs[index] = transactions[i].organization;
239                 amounts[index] = transactions[i].amount;
240                 timestamps[index] = transactions[i].timestamp;
241                 txHashes[index] = transactions[i].txHash;
242                 index++;
243             }
244         }
245     }
246
247     /// @notice Returns the product origin transactions.
248     /// @return Arrays of product origin transaction details.
249     function getProductOriginTransactions() public view returns (uint256[]
250         memory, uint256[] memory, uint256[] memory) {
251         uint256 count = productOriginTransactions.length;
252
253         uint256[] memory originProducts = new uint256[](count);
254         uint256[] memory endProducts = new uint256[](count);
255         uint256[] memory timestamps = new uint256[](count);
256
257         for (uint256 i = 0; i < count; i++) {
258             originProducts[i] = productOriginTransactions[i].originProduct;
259             endProducts[i] = productOriginTransactions[i].endProduct;
260             timestamps[i] = productOriginTransactions[i].timestamp;
261         }
262
263     return (originProducts, endProducts, timestamps);
264 }
265
266     /// @notice Returns the rejected transactions of an organization.
267     /// @param organization The address of the organization.
268     /// @return Arrays of rejected transaction details.
269     function getRejectedTransactionDetails1(address organization) public
270         view returns (address[] memory, int256[] memory, uint256[] memory,
271         string[] memory) {
272         uint256 count = 0;
273         for (uint256 i = 0; i < rejectedTransactions.length; i++) {
274             if (rejectedTransactions[i].organization == organization) {
275                 count++;
276             }
277         }
278
279         address[] memory orgs = new address[](count);
```

```

277     int256[] memory amounts = new int256[](count);
278     uint256[] memory timestamps = new uint256[](count);
279     string[] memory reasons = new string[](count);
280     uint256 index = 0;
281     for (uint256 i = 0; i < rejectedTransactions.length; i++) {
282         if (rejectedTransactions[i].organization == organization) {
283             orgs[index] = rejectedTransactions[i].organization;
284             amounts[index] = rejectedTransactions[i].amount;
285             timestamps[index] = rejectedTransactions[i].timestamp;
286             reasons[index] = rejectedTransactions[i].reason;
287             index++;
288         }
289     }
290     return (orgs, amounts, timestamps, reasons);
291 }
292
293 /**
294  * @notice Returns the rejected transactions of an organization.
295  * @param organization The address of the organization.
296  * @return Arrays of rejected transaction details.
297 */
298 function getRejectedTransactionDetails2(address organization) public
299 view returns (string[] memory, uint256[] memory, uint256[] memory,
300 bytes32[] memory) {
301     uint256 count = 0;
302     for (uint256 i = 0; i < rejectedTransactions.length; i++) {
303         if (rejectedTransactions[i].organization == organization) {
304             count++;
305         }
306     }
307
308     string[] memory productNames = new string[](count);
309     uint256[] memory productQuantities = new uint256[](count);
310     uint256[] memory co2Emissions = new uint256[](count);
311     bytes32[] memory txHashes = new bytes32[](count);
312     uint256 index = 0;
313     for (uint256 i = 0; i < rejectedTransactions.length; i++) {
314         if (rejectedTransactions[i].organization == organization) {
315             productNames[index] = rejectedTransactions[i].productName;
316             productQuantities[index] = rejectedTransactions[i].
317             productQuantity;
318             co2Emissions[index] = rejectedTransactions[i].co2Emission;
319             txHashes[index] = rejectedTransactions[i].txHash;
320             index++;
321         }
322     }
323     return (productNames, productQuantities, co2Emissions, txHashes);
324 }
325 }
```

Listato 5.1: Contratto Solidity Esempio

6. Manuale d'utilizzo

6.1 Manuale d'utilizzo

Per utilizzare l'applicazione che è stata descritta finora, è necessario per prima cosa avere **Docker** e **Docker Compose** installati sul proprio dispositivo.

Secondo l'articolo ufficiale di *Quorum Dev*, si raccomanda di limitare l'uso della memoria di Docker a 6 GB durante il lavoro con la blockchain. Il procedimento per impostare questa limitazione dipende dal sistema operativo in uso.

6.1.1 Come scaricare l'applicazione

Per scaricare l'applicazione sul proprio dispositivo è necessario clonare la repository al seguente *link* tramite il comando da terminale

```
1 git clone https://github.com/GabrielPiercecchi/SoftwareSecurity -  
BlockchainProject.git
```

ed accedervi con

```
1 cd SoftwareSecurity -BlockchainProject
```

Una volta scaricata l'applicazione si passa a creare le variabili d'ambiente del file `.env`:

Configurazione del servizio e database:

- SERVICE_HOST → Definisce la porta su cui il servizio è in esecuzione;
- SERVICE_PORT → Specifica la porta del servizio, tipicamente utilizzata per accedere all'applicazione;
- DATABASE_USER → Nome utente per accedere al database PostgreSQL;
- DATABASE_PASSWORD → Password per l'accesso al database;
- DATABASE_NAME → Nome del database utilizzato dall'applicazione;
- DATABASE_PORT → Porta su cui è in ascolto il database PostgreSQL;
- DATABASE_HOST → Indirizzo del server database (in questo caso, la macchina locale);
- DATABASE_PORT_FASK_APP → Porta del database PostgreSQL specifica per l'applicazione Flask.

Chiavi di amministrazione e connessione blockchain:

- ADMIN_PRIVATE_KEY → Chiave privata dell'amministratore della blockchain. Corrisponde alla chiave relativa al profilo utente su *Metamask*;
- ADMIN_ADDRESS → Indirizzo pubblico dell'amministratore sulla blockchain. Corrisponde anche qui all'indirizzo pubblico relativo al profilo utente su *Metamask*;

■ **Nota Bene** 6.1 Per ottenere questi dati su *Metamask* è sufficiente installare la sua estensione all'interno del proprio Browser e creare un Account:

- Per *Google Chrome* al seguente *link*
- Per *Firefox* al seguente *link*

- SECRET_KEY → Chiave segreta utilizzata dall'applicazione per autenticazione e sicurezza. Può essere creata tramite il file *algorithms/secret_key_generator.py*;
- BLOCKCHAIN_URL = *http://rpcnode:8545* → URL del nodo RPC della blockchain per le interazioni con gli smart contract

Credenziali per Oracle:

- ORACLE_USERNAME → Nome utente per il database Oracle;
- ORACLE_PASSWORD → Password per il database Oracle.

Configurazione PostgreSQL per Docker:

- POSTGRES_USER → Nome utente per PostgreSQL utilizzato nei container Docker;
- POSTGRES_PASSWORD → Password per PostgreSQL nei container Docker;
- POSTGRES_HOST → Nome host del database PostgreSQL nei container Docker;
- POSTGRES_DB → Nome del database PostgreSQL nei container Docker.

Configurazione Blockscout:

- BLOCKSCOUT_HOST → Host del database PostgreSQL utilizzato da Blockscout per l'esplorazione della blockchain.

Configurazione di rete dell'applicazione:

- HOST → Indica che l'applicazione deve essere accessibile da qualsiasi indirizzo IP;
- PORT → Definisce la porta su cui l'applicazione Flask è in ascolto;
- LOCALHOST → Alias per riferirsi alla macchina locale.

Versioni dei componenti Quorum e Besu:

- BESU_VERSION = 23.4.1 → Versione del client Ethereum Besu;
- QUORUM_VERSION = 23.4.0 → Versione della piattaforma blockchain Quorum;
- TESSERA_VERSION = 23.4.0 → Versione del gestore delle transazioni private Tessera;
- ETHSIGNER_VERSION = 22.1.3 → Versione di EthSigner, il servizio per la gestione delle firme digitali;
- QUORUM_EXPLORER_VERSION = 4f60191 → Versione dell'esploratore blockchain per Quorum.

File di blocco per Quickstart:

- LOCK_FILE = .quorumDevQuickstart.lock → File utilizzato per evitare configurazioni duplicate durante l'avvio del network di Quorum in modalità Quickstart.

Credenziali degli impiegati delle organizzazioni:

- USER1_USERNAME → username impiegato della prima organizzazione.
- USER1_PASSWORD → password impiegato della prima organizzazione.
- USER2_USERNAME → username impiegato della seconda organizzazione.
- USER2_PASSWORD → password impiegato della seconda organizzazione.
- USER3_USERNAME → username impiegato della terza organizzazione.
- USER3_PASSWORD → password impiegato della terza organizzazione.
- USER4_USERNAME → username impiegato della quarta organizzazione.
- USER4_PASSWORD → password impiegato della quarta organizzazione.

Algoritmi di consenso per Quorum e Besu:

- GOQUORUM_CONS_ALGO = qbft → Algoritmo di consenso utilizzato per Quorum GoQuorum (QBFT);
- BESU_CONS_ALGO = QBFT → Algoritmo di consenso utilizzato per Besu (IBFT 2.0, noto anche come QBFT).

Configurazione del logging:

- LOG4J_CONFIGURATION_FILE = /quorum-test-network/config/log-config.xml → Percorso del file di configurazione Log4J per la gestione dei log del network Quorum.

Ricreazione rete di quorum

Per creare la rete che ospiterà la *blockchain* è necessario seguire i seguenti passaggi:

1. Rimuovere `quorum-test-network/` dalla cartella root del progetto:

```
rm -rf quorum-test-network
```

2. Creare rete di test con il comando da terminale:

```
npx quorum-dev-quickstart
```

■ **Nota Bene** 6.2 Se si utilizza Windows, è necessario eseguire alcune fasi aggiuntive prima di eseguire questo comando:

- (a) Installare **WSL** (Windows Subsystem for Linux) su Windows utilizzando *Powershell* con i permessi di amministratore, eseguendo il seguente comando:

```
wsl --install
```

Dopo l'installazione, è consigliabile riavviare il PC per assicurarsi che tutte le modifiche vengano applicate correttamente.

- (b) Scaricare una distribuzione Linux tramite WSL:

```
wsl --install -d <Distribuzione Linux>
```

Per verificare le distribuzioni disponibili, eseguire:

```
wsl --list --online
```

Riavviare il PC dopo aver completato l'installazione.

- (c) Avviare Docker Desktop e abilitare l'integrazione con WSL e la distribuzione Linux scaricata. La configurazione corretta per Ubuntu è mostrata nella Figura 6.1.

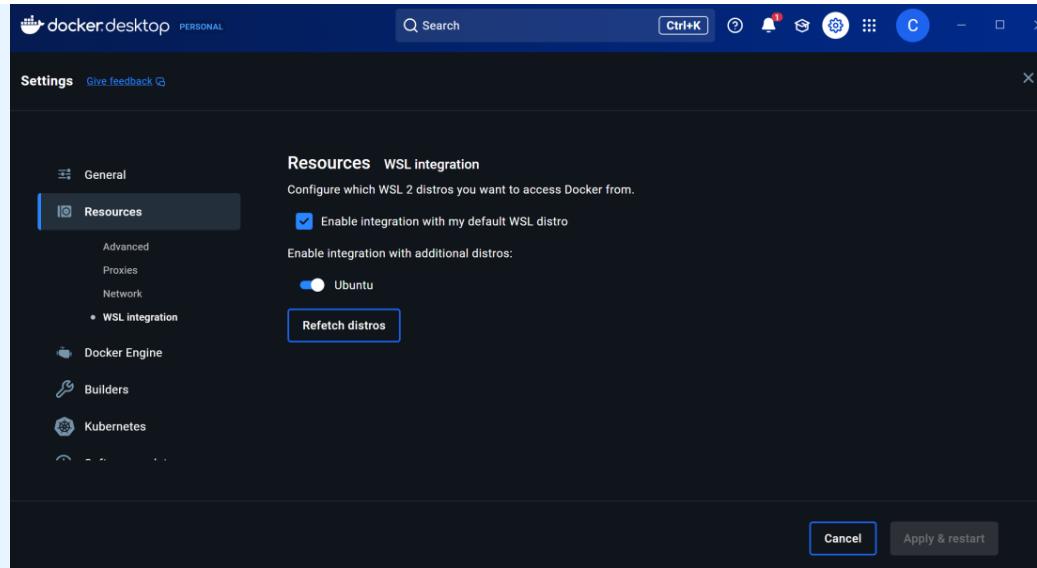


Figure 6.1: Integrazione di WSL con Docker

- (d) Aprire un terminale nella cartella di root del progetto ed eseguire il seguente comando per accedere alla distribuzione Linux:

```
wsl -d <Distibuzione Linux>
```

Questo ti permetterà di interagire con il sistema Linux direttamente dal terminale di Windows.

- (e) Per eseguire `npx` correttamente, è necessario installare i pacchetti `nodejs` e `npm`. Esegui i seguenti comandi:

```
sudo apt update && sudo apt upgrade -y
sudo apt install nodejs npm -y
```

Questi pacchetti sono necessari per eseguire comandi JavaScript da riga di comando e per gestire le dipendenze del progetto.

Una volta completati questi passaggi, sarà possibile eseguire

```
npx quorum-dev-quickstart
```

senza problemi.

- (a) scegliere *Hyperledger Besu* [1]
- (b) scegliere **Yes** alle transazioni private
- (c) scegliere *Loki* [1]
- (d) scegliere **No** a *Chainlens*
- (e) scegliere **No** a *Blockscout*
- (f) scegliere come cartella `./quorum-test-network` [*default*]

3. Scaricare **OpenSSI**, la cui documentazione è disponibile al seguente *link*, tramite il comando

```
sudo apt install openssl
```

per Linux (Ubuntu) e Windows (WSL), o tramite l'*App Store* per macOS al seguente link.

4. Posizionarsi nella cartella *nginx/certs* tramite il comando

```
cd nginx/certs
```

Nel caso in cui non esistesse sarà necessario crearla in questo modo:

```
mkdir -p nginx/certs
```

Successivamente, eseguire il seguente comando per generare i certificati SSL necessari a *nginx*:

```
1      openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes  
2
```

Questo comando crea una chiave privata (*key.pem*) e un certificato pubblico (*cert.pem*) valido per 365 giorni. L'opzione **-nodes** disabilita la protezione della chiave privata con una passphrase, il che è utile per automatizzare il processo di avvio del server web.

Dopo aver eseguito il comando, avrai i due file *key.pem* e *cert.pem*, che sono necessari per configurare *nginx* con HTTPS.

Rimozione eventuali residui di esecuzioni precedenti

Nel caso in cui il progetto fosse stato eseguito in precedenza, potrebbero essere rimasti dei file residui che potrebbero compromettere lo stato del progetto durante la successiva esecuzione. È quindi importante rimuovere questi file per evitare conflitti o errori. I file che potrebbero essere residui includono:

- *algorithms/nonce.txt* → File contenente un nonce che potrebbe essere stato utilizzato in esecuzioni precedenti.
- *contract/compiled_code.json* → File contenente il codice compilato del contratto smart, che potrebbe non essere aggiornato.
- *contract/contract_address.txt* → File contenente l'indirizzo del contratto smart precedentemente distribuito sulla blockchain.

Per rimuovere questi file, eseguire i seguenti comandi:

```
1      rm algorithms/nonce.txt  
2      rm contract/compiled_code.json  
3      rm contract/contract_address.txt
```

Una volta che questi file sono stati rimossi, è possibile procedere con la creazione dell'immagine Docker e l'avvio del container utilizzando il comando:

```
1      docker-compose up -d --build
```

Questo comando costruirà l'immagine Docker e avvierà il container in modalità distacco (-d), in modo che il servizio continui a funzionare in background. Attendere qualche secondo affinché l'immagine venga costruita e il container avviato correttamente. Una volta completato, l'applicazione sarà disponibile all'indirizzo <http://localhost:5000>.

Poiché l'applicazione in Flask attende che la blockchain venga avviata prima di iniziare il proprio funzionamento, sarà necessario attendere alcuni secondi prima di accedere al sito web, che sarà disponibile all'indirizzo <http://localhost:5000>.

Come già spiegato precedentemente, è possibile controllare lo stato della blockchain tramite l'esploratore all'indirizzo: <http://localhost:25000/explorer/explorer>.

6.2 Livello 0

Il Livello 0 è associato al ruolo di *Buyer*, che include tutti gli utenti con privilegi fortemente limitati. Questi utenti possono visualizzare solo i contenuti pubblici del software. Il Buyer avrà accesso a queste informazioni senza la necessità di registrarsi.

In Figura 6.2, è mostrata la *Home Page* del sito, dalla quale il Buyer potrà esplorare informazioni sui *Products* e sulle *Organizations* che li producono.

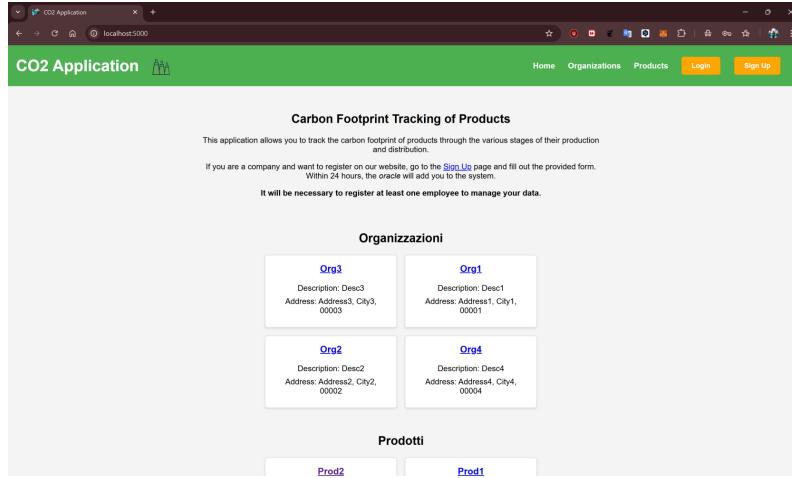


Figure 6.2: Home Page

Dall'header della *Home Page*, il Buyer avrà la possibilità di accedere sia alla pagina che visualizza tutte le *Organizations*, sia a quella che mostra tutti i *Products*, come evidenziato nelle Figure 6.3 e 6.4.

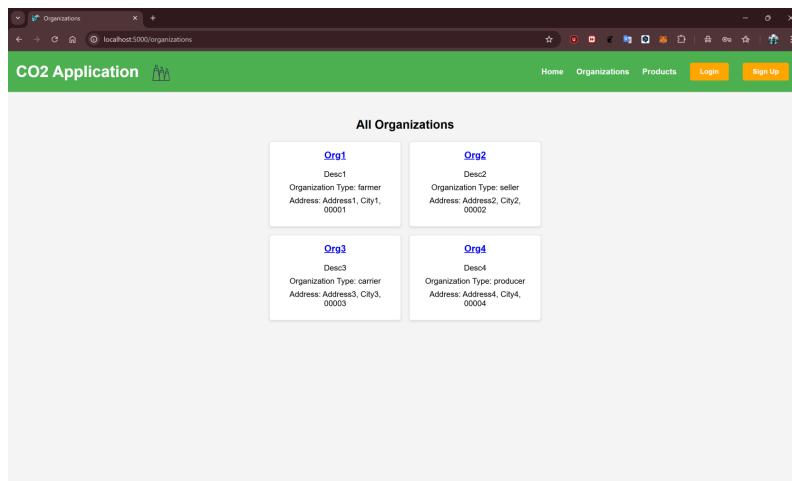


Figure 6.3: Pagina di visualizzazione di tutte le *Organization*

Successivamente, facendo clic sui nomi delle singole *Organizations*, il Buyer potrà accedere alle informazioni dettagliate relative a ciascuna azienda, come mostrato in Figura 6.5. Lo stesso procedimento può essere seguito per i *Products*: basta cliccare sui singoli nomi presenti nell'header per visualizzare la pagina corrispondente, come indicato nella Figura.

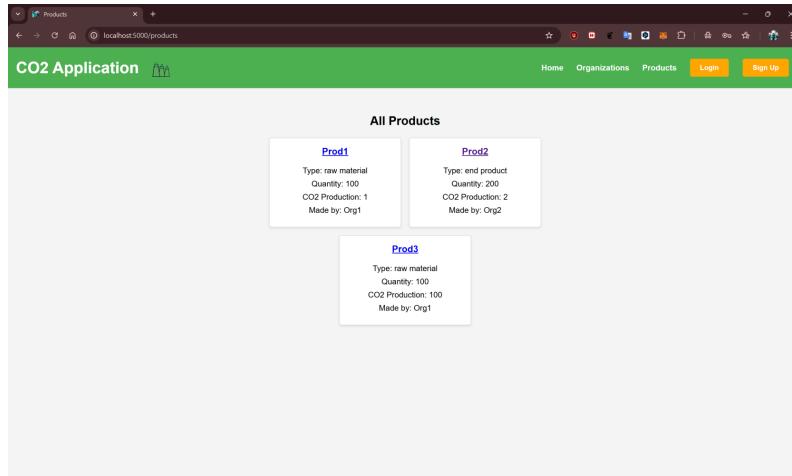


Figure 6.4: Pagina di visualizzazione di tutti i *Product*

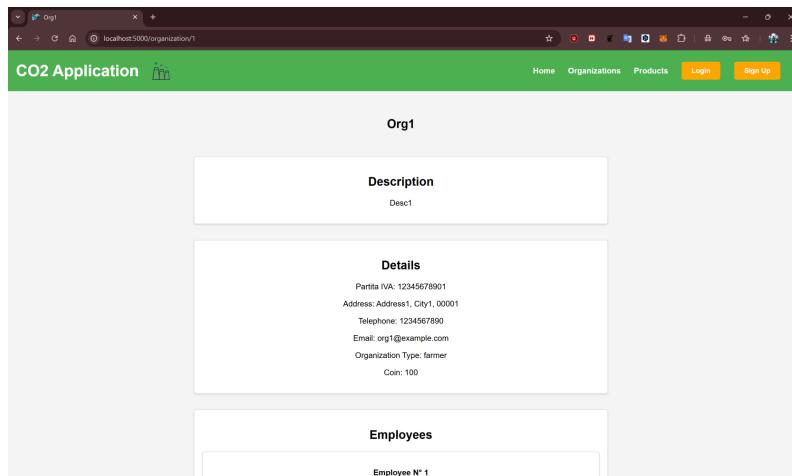


Figure 6.5: Pagina di visualizzazione relativo a una singola *Organization*

È importante precisare che questi dettagli sono accessibili cliccando sui nomi dei singoli elementi presenti sulla *Home Page*, come si evince dalla Figura 6.2.

6.3 Login e Registrazione

Il processo di registrazione, illustrato nella Figura 6.7, consente di inserire i dati relativi a un'organizzazione e ai suoi dipendenti (è richiesto almeno un dipendente). Dopo la compilazione del modulo, la richiesta non viene accettata immediatamente, ma rimane in attesa di verifica da parte dell'entità denominata "Oracolo". Quest'ultimo ha il compito di approvare o rifiutare la registrazione in base a criteri prestabiliti.

Una volta approvata la registrazione, l'utente può accedere alla pagina di *Login* (Figura 6.6) ed effettuare l'accesso in modo sicuro.

Durante il login, l'utente ha a disposizione un massimo di quattro tentativi per inserire correttamente *Username* e *Password*. Se, nei primi tre tentativi, le credenziali risultano errate, viene mostrato un messaggio che informa dell'accesso non riuscito.

Nel caso in cui l'accesso fallisca anche al quarto tentativo, l'utente dovrà attendere 30 secondi prima di poter riprovare. Anche in questo caso, un messaggio lo avviserà della sospensione temporanea.

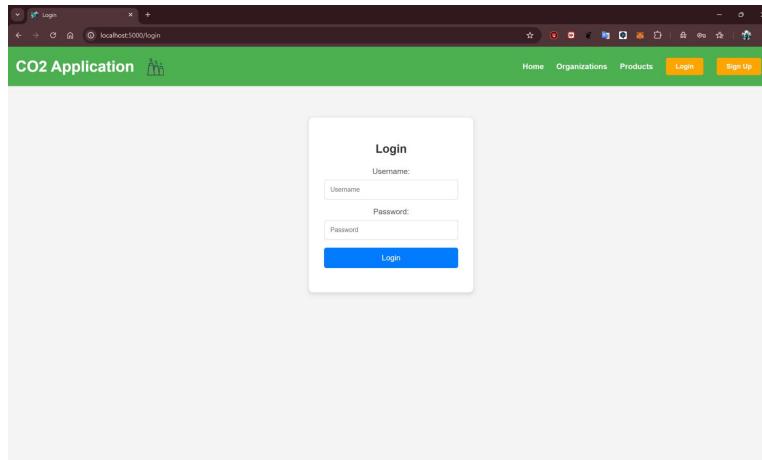


Figure 6.6: Form di Login

6.4 Livello 1

Dopo l'accesso, l'utente viene indirizzato a una delle tre possibili schermate *Home*, in base alla tipologia di *Employer* con cui si è autenticato. Indipendentemente dal tipo di *Employer*, è sempre disponibile l'opzione per modificare i propri dati personali tramite il pulsante **Update Personal Data**. Cliccando su questo pulsante, l'utente viene reindirizzato a una pagina (Figura 6.8) in cui sono visualizzati i dati personali, che possono essere aggiornati.

È possibile effettuare richieste per i prodotti delle altre aziende tramite il pulsante **Manage Product Requests** (Figura 6.9). Questo è valido per il *farmer*, il *producer* e il *seller*, mentre per il *carrier* il processo differisce, come spiegato al paragrafo 6.4.1.

Inoltre, è possibile visualizzare tutte le *delivery* (Figura 6.11) cliccando su **View my organization Delivery**.

Tra le azioni più rilevanti che i vari *Employer* possono compiere vi sono quelle relative ai *Coin*. In particolare, è possibile visualizzare le richieste di *Coin* tramite il pulsante **View Coin Requests**, che consente anche di accettarle. Inoltre, è possibile consultare l'elenco delle transazioni riuscite cliccando su **View Transactions** e l'elenco delle transazioni rifiutate tramite **View Rejected Transactions**.

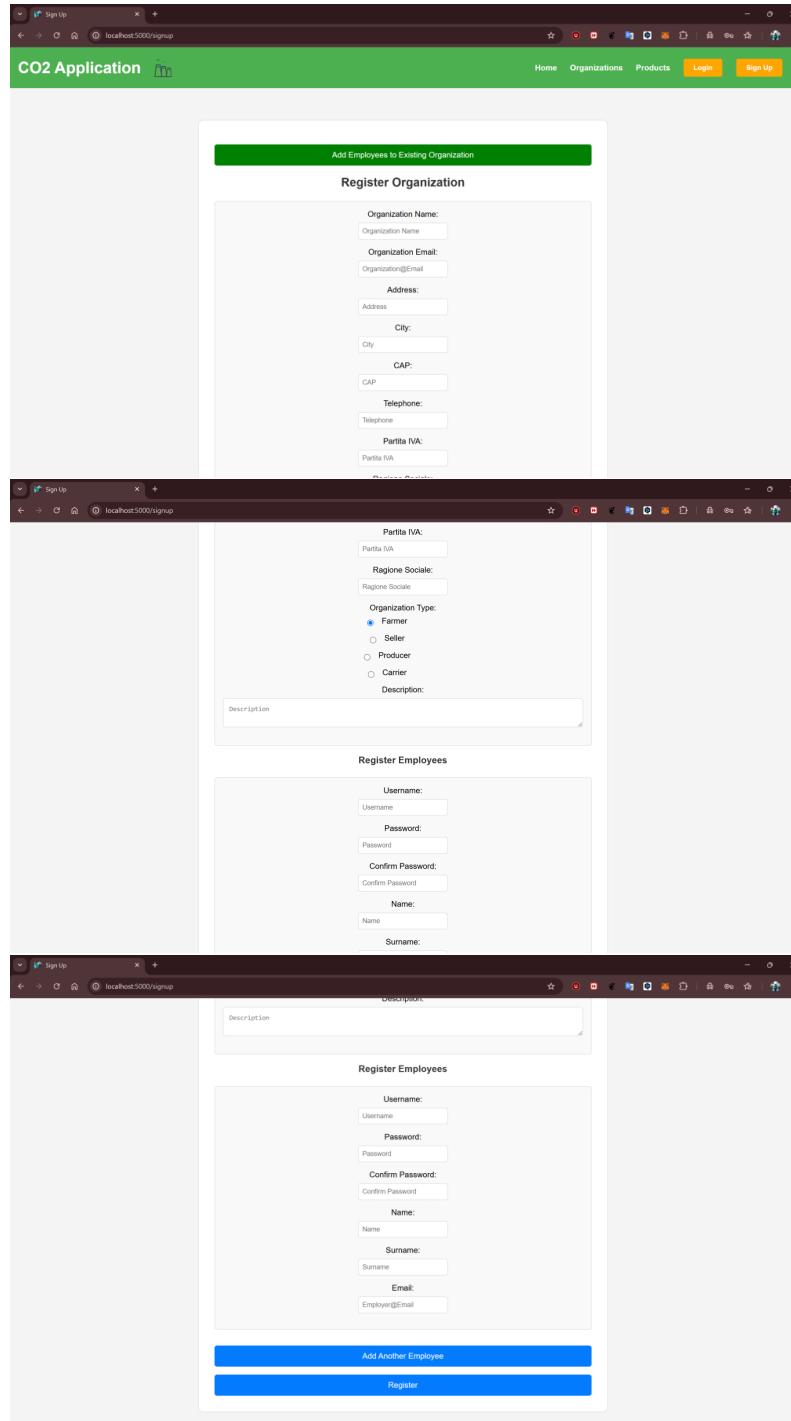


Figure 6.7: Registrazione

6.4.1 Farmer/Producer

Se l'utente è autenticato come *Farmer* o *Producer*, avrà accesso a una schermata Home, mostrata in Figura 6.12, in cui saranno disponibili diversi pulsanti per l'esecuzione di specifiche funzioni.

Nel dettaglio, è presente il pulsante **Manage my Organization Products** (Figura 6.13), che consente di aggiungere un nuovo prodotto con tutte le relative specifiche. Successivamente, vengono mostrati tutti i prodotti inseriti, che possono essere anche modificati.

The screenshot shows a web browser window titled "Update Personal Data" with the URL "127.0.0.1:5000/employer/employer_update_personal_data". The page has a green header bar with the text "CO2 Application" and icons for Home, Organizations, Products, Employee Home, and Logout. The main content area is titled "Update Personal Information" and contains the following form fields:

- Name:
- Surname:
- Email:
- Username:
- Password:
- Confirm Password:

A large green "Update Data" button is located at the bottom of the form.

Figure 6.8: Update Personal Data

The screenshot shows a web browser window titled "View other products" with the URL "127.0.0.1:5000/employer/manage_product_requests/view_other_products/". The page has a green header bar with the text "CO2 Application" and icons for Home, Organizations, Products, Employee Home, and Logout. The main content area is titled "Products from Other Organizations" and displays a single product entry:

Prod2
 ID: 2
 Type: end product
 Max Quantity: 200
 Organization: Org2

A green "Create Product Request" button is located at the bottom of the product entry.

Figure 6.9: Manage Product Requests

6.4.2 Carrier

Il **Carrier** (Figura 6.14) ha la possibilità di visualizzare le consegne effettuate tramite il pulsante **View my Organization Deliveries** (Figura 6.15). Quando un'azienda effettua una richiesta di prodotti tramite **View my organization product requests** e seleziona un'azienda *carrier* per la consegna, quest'ultima riceve la richiesta e inserisce i valori di CO₂ che verranno emessi durante il trasporto.

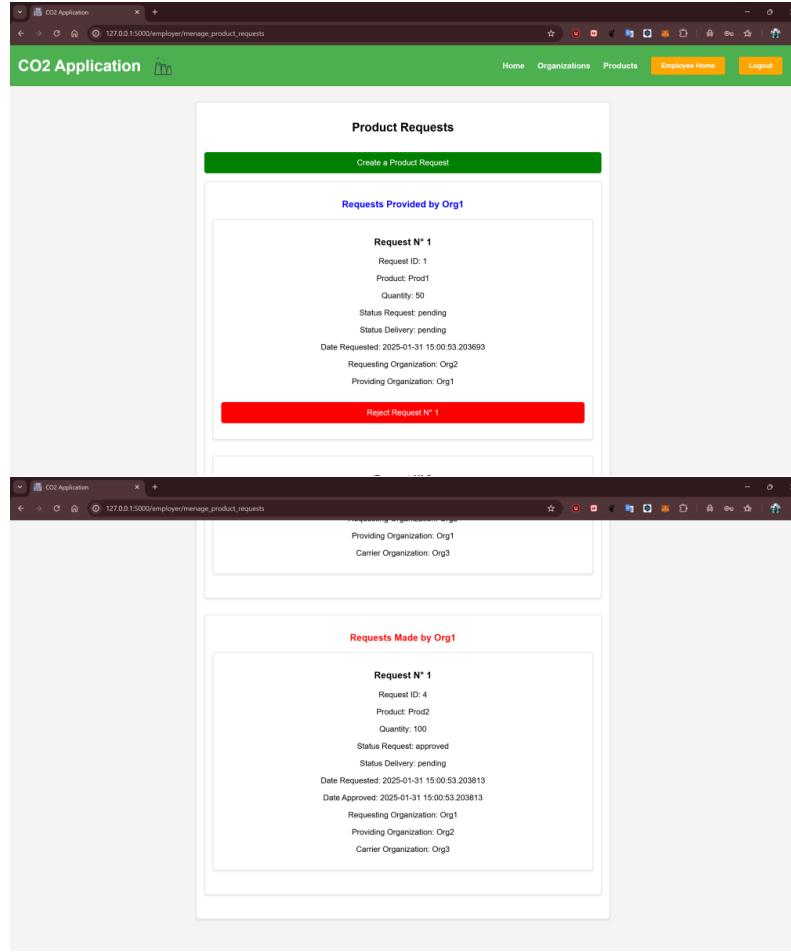


Figure 6.10: Product Request

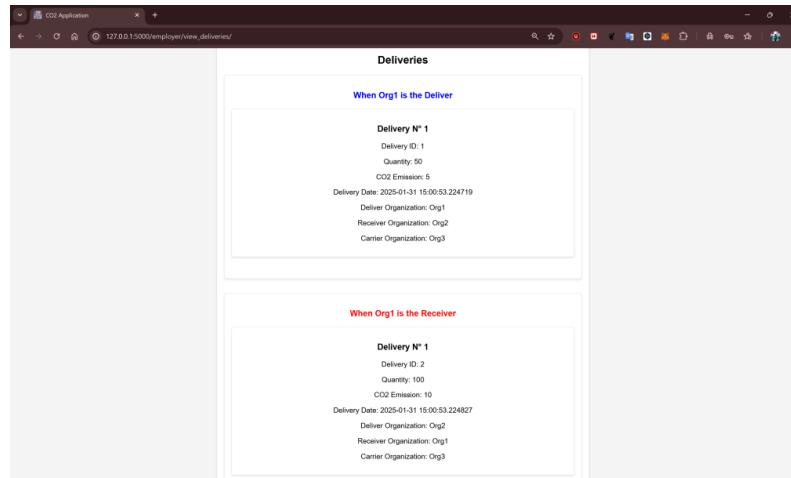


Figure 6.11: Delivery

6.4.3 Seller

Nella *Home* del *Seller* (Figura 6.16), l'utente non ha la possibilità di creare nuovi prodotti, ma può solamente richiederli. Inoltre, ha accesso alla gestione delle operazioni relative ai *Coin* e alle transazioni.

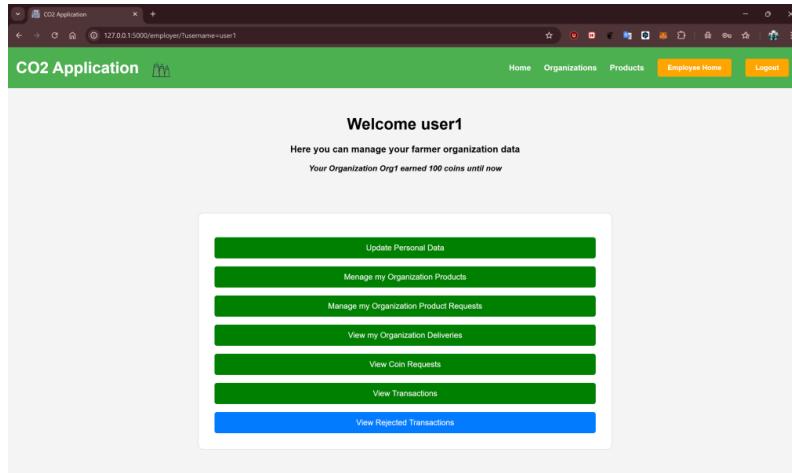


Figure 6.12: Home Employer (farmer/producer)

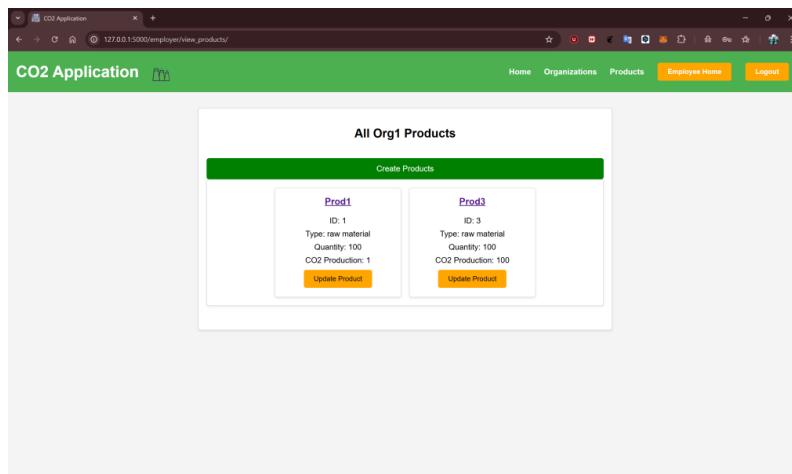


Figure 6.13: Manage my Organization Products

6.5 Livello 2

Come già menzionato nella Sezione 6.3, l'*Oracolo*¹ è responsabile della gestione e del controllo del sistema. Come mostrato in Figura 6.17, l'*Oracolo* ha la possibilità di visualizzare e, se necessario, approvare le richieste di registrazione delle diverse *Organization* tramite il pulsante *View all inactive organizations*.

Una volta accettata una richiesta, l'*Oracolo* può consultare anche le richieste di registrazione degli *Employer* associati alle *Organization* attive, utilizzando il pulsante *View all inactive employer in active organization*.

Inoltre, tramite il pulsante *Coin Transfer*, l'*Oracolo* può selezionare una *Organization* specifica e trasferire una determinata quantità di *Coin* a un'altra azienda.

Per garantire un controllo accurato del sistema, il pulsante *View Log file* consente all'*Oracolo* di monitorare tutte le attività svolte dagli utenti, facilitando così l'individuazione di eventuali anomalie o violazioni del software.

¹L'*Oracolo* può accedere a tutte queste funzionalità dopo aver effettuato il login nel sistema.

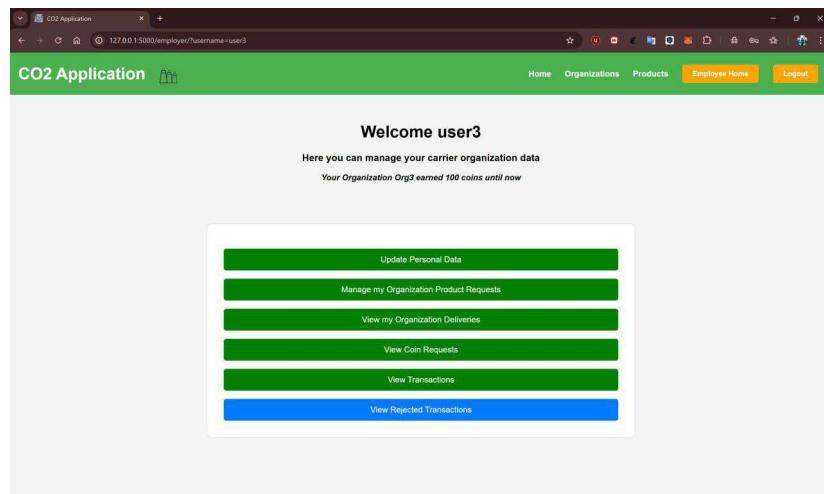


Figure 6.14: Carrier Home

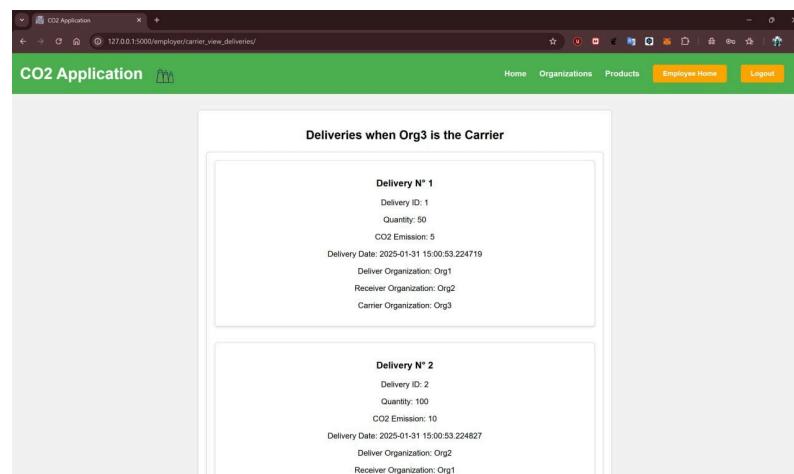


Figure 6.15: View my Organization Deliveries

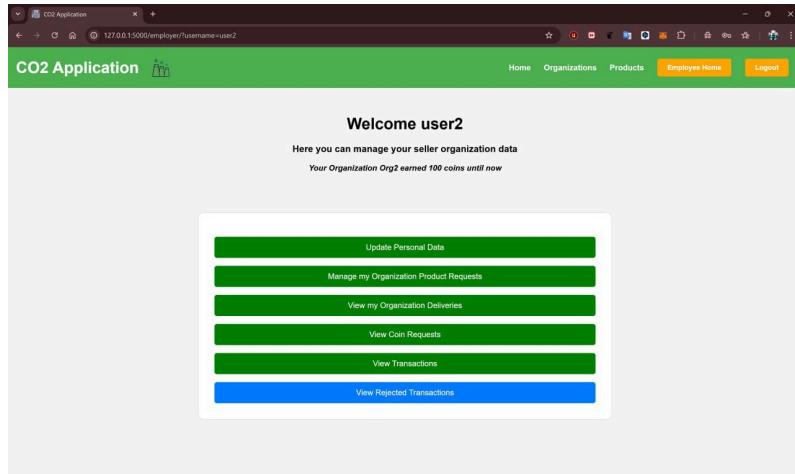


Figure 6.16: Seller Home

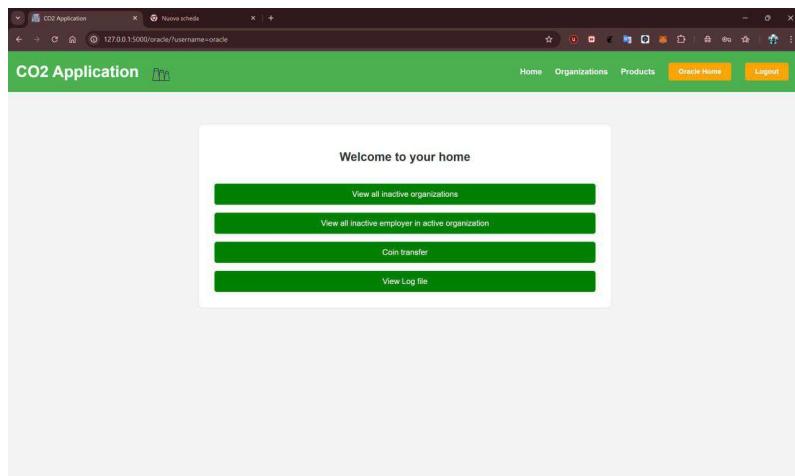


Figure 6.17: Oracle Home