



UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTA DI INGEGNERIA

Corso di Software Security and Blockchain

Progetto Sustainable Food Supply Chain

Docenti:

Prof. Luca Spalazzi

Studenti:

Andrea Giuliani
Lionel Djouaka Kelefack
Stefano Marinucci
Marco Di Vita

ANNO ACCADEMICO 2024/2025

Contents

1	Valutazione del rischio	2
1.1	Analisi dei requisiti	2
1.1.1	Diagramma senza attore sistema	2
1.1.2	Diagramma con attore sistema	4
1.2	Analisi preliminare del rischio	5
1.3	Identificazione del rischio	5
1.3.1	Identificazione degli asset	5
1.4	Analisi dei rischi	7
1.4.1	Valutazione economica ed esposizione dell'asset	7
1.4.2	Identificazione delle minacce	9
1.5	Decomposizione del rischio	12
1.5.1	Attacco agli asset	12
1.6	Riduzione del rischio	23
1.6.1	Identificazione dei controlli	23
1.6.2	Valutazione della fattibilità	23
1.6.3	Definizione dei requisiti di sicurezza	23
2	Progettazione sicura	26
2.1	Linee guida e buone pratiche	26
2.1.1	Principi di progettazione di Saltzer and Schroeder	27
2.1.2	OWASP: principi di sicurezza	27
2.1.3	Linee guida di Sommerville	28
2.2	Design architetturale	28
2.2.1	Isolation, Obfuscation, Monitoring	29
2.2.2	Distribution, Redundancy, Diversity	31
2.2.3	Analisi	32
2.2.4	Modellazione attraverso Markov	32
2.3	Scelta delle tecnologie e analisi delle debolezze	38
2.3.1	Tecnologie usate nello sviluppo	38
2.3.2	Analisi delle debolezze - SWOT	40

3	Implementazione	42
3.1	Programmazione sicura	42
3.2	Standard di codifica	43
3.3	Database	43
3.4	Autenticazione	44
3.5	On-chain	44
3.6	Action flow dell'utente	46
3.6.1	Registrazione	46
3.6.2	Login	47
3.6.3	Menu Profile	49
3.6.4	Menu Operation	50
3.6.5	Menu Report	52
3.7	Analisi statica	53

List of Figures

1.1	Pi* senza attore sistema	3
1.2	Pi* con attore sistema	4
1.3	Modello STRIDE	11
1.4	Misuse Case: Condivisione involontaria dei dati relativi ai carbon credit tra gli utenti	13
1.5	Abuse case: Accesso malevolo ai dati condivisi tra gli utenti	14
1.6	Misuse case: Condivisione accidentale delle credenziali di accesso al sistema	15
1.7	Abuse case: Accesso malevolo al sistema	16
1.8	Inserimento di dati errati sulle azioni compensative	17
1.9	Abuse case: inserimento di dati malevoli sulle azioni compensative	18
1.10	Attack tree: Accesso dannoso ai dati condivisi	19
1.11	Attack tree: Inserimento dati malevoli	20
1.12	Attack tree: Accesso malevolo al sistema	21
1.13	Continuo tabella STRIDE	22
1.14	Equazione ROI	23
1.15	Continuo tabella STRIDE	25
2.1	Prima parte dell'algoritmo	33
2.2	Seconda parte dell'algoritmo	33
2.3	Modello Prism	35
2.4	Etichette modello Prism	35
2.5	Prima proprietà	36
2.6	Prima proprietà - variante B	36
2.7	Verifica seconda proprietà	37
3.1	Schermata iniziale	46
3.2	Registrazione	47
3.3	Login	48
3.4	Menu utente	48
3.5	Menu profilo	49

3.6	Informazioni utente	49
3.7	Aggiornamento profilo	50
3.8	Cambio password	50
3.9	Menu operazioni	50
3.10	Cessione crediti	51
3.11	Registrazione operazione	51
3.12	Registrazione green action	52
3.13	Menu Report	52
3.14	Generazione report	52
3.15	Visualizzazione report	53

Chapter 1

Valutazione del rischio

1.1 Analisi dei requisiti

L'analisi dei requisiti è una fase cruciale nel ciclo di vita dello sviluppo del software e rappresenta un processo fondamentale all'interno dell'Ingegneria dei Requisiti (Requirement Engineering). Questa fase mira a identificare, comprendere e documentare in modo esauriente le necessità, le funzionalità e i vincoli del sistema software che si intende sviluppare. In particolare, si considerano l'insieme di attività volte a gestire i requisiti del sistema in maniera organizzata e sistematica, al fine di garantire che il software sviluppato soddisfi le esigenze degli utenti, dei clienti e degli stakeholder. L'obiettivo principale è individuare i requisiti funzionali, che riguardano le funzionalità e i servizi che il sistema deve fornire, e i requisiti non funzionali, che si configurano come vincoli sulle funzioni e/o i servizi offerti, descrivendo il modo in cui il sistema si comporta. Per la modellazione dei requisiti all'interno di questo progetto è stato scelto i*; si tratta di una notazione grafica utilizzata per modellare e analizzare l'ingegneria dei requisiti, concentrando l'attenzione sulle relazioni tra gli attori coinvolti in un sistema e sulle loro dipendenze reciproche. La modellazione dei requisiti è stata eseguita utilizzando un approccio agile, in maniera iterativa ed incrementale.

1.1.1 Diagramma senza attore sistema

Ogni attore è descritto attraverso le sue intenzioni (goal), compiti (tasks) e risorse coinvolte. Gli attori principali (intesi come ruoli) risultano essere:

- Farmer
- Carrier
- Seller

- Producer

Ad esempio, il **Farmer** si occupa di attività legate alla produzione primaria, come **coltivare** e **piantare alberi**, con l'obiettivo di una **buona produzione** e della riduzione della CO. Il **Producer** invece è responsabile della trasformazione della materia prima in **prodotto lavorato**, e si impegna nell'**ottimizzazione dei processi produttivi**, nell'**utilizzo di materie prime riciclate** e nel miglioramento dell'**efficienza energetica**. Il **Carrier** svolge il ruolo logistico, occupandosi del **trasporto** delle materie prime e dei prodotti finiti, e punta a **scegliere percorsi ottimali** per ridurre l'impatto ambientale. Il **Seller**, da parte sua, ha come obiettivo principale la **vendita**, ma è anche coinvolto in azioni di tipo sociale, come le **donazioni per la cura ambientale**.

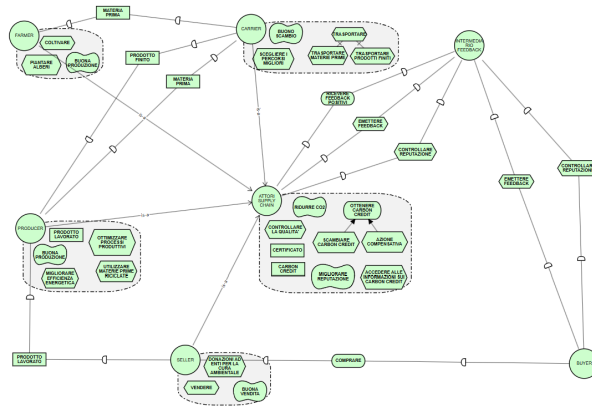


Figure 1.1: Pi* senza attore sistema

Presente anche il **Buyer** rappresenta il consumatore finale e, oltre all'acquisto, partecipa attivamente al sistema di **feedback** e **controllo della reputazione**, così come l'**Intermediario del feedback**, che svolge un ruolo centrale nella raccolta e diffusione delle valutazioni da parte degli utenti. Questi attori sono presenti nel diagramma pi* ma non sono il fulcro centrale del progetto, difatti non verranno considerati nella stesura del secondo e terzo capitolo, ma nella prima fase iniziale del progetto abbiamo comunque considerato questi elementi per andare a creare una situazione quanto piu' chiara e reale possibile.

Di particolare importanza il nodo **Attori Supply Chain** racchiude una serie di obiettivi trasversali, condivisi da tutti i partecipanti alla filiera: tra questi troviamo la volontà di **ridurre le emissioni di CO**, **ottenere e scambiare carbon credit**, **migliorare la reputazione** degli attori virtuosi e **controllare la qualità** delle operazioni registrate.

L'intero sistema è pensato per operare su blockchain, dove ogni attore può registrare le proprie azioni (es. produzione, trasporto, vendita), ottenere un punteggio in termini di sostenibilità ambientale e ricevere o cedere **carbon credit** in modo trasparente, verificabile e non modificabile. Questo modello evidenzia come il progetto **SFSCChain** si basi su un'architettura distribuita, che valorizza la collaborazione e la responsabilità degli attori, con l'obiettivo finale di promuovere una filiera alimentare sostenibile e tracciabile.

1.1.2 Diagramma con attore sistema

A differenza dal diagramma precedente, una novità fondamentale rispetto alla versione precedente è l'introduzione dell'attore denominato “**Piattaforma Supply Chain**”, che rappresenta la componente tecnologica centrale del sistema SFSCChain. Si tratta dell'infrastruttura software — sviluppata sulla base della blockchain — che consente agli attori della filiera di interagire in modo sicuro, tracciabile e decentralizzato.

La **Piattaforma Supply Chain** non è un attore umano, bensì il sistema stesso, responsabile della **gestione automatizzata delle operazioni critiche**. È proprio grazie a questa piattaforma che gli attori reali della filiera — come farmer, producer, carrier, seller e buyer — possono svolgere le proprie attività nel rispetto dei criteri di sicurezza, trasparenza e affidabilità.

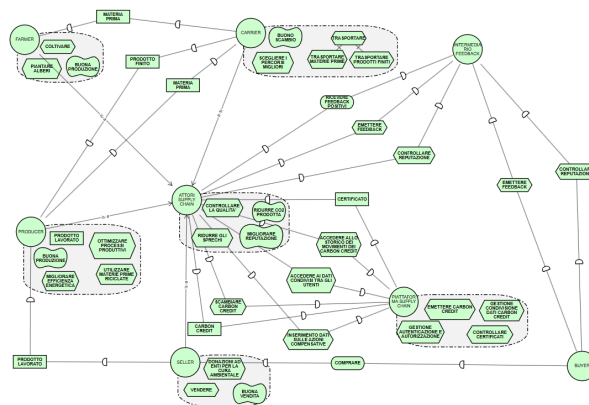


Figure 1.2: Pi* con attore sistema

Dal punto di vista funzionale, la piattaforma gestisce **l'autenticazione e l'autorizzazione degli utenti**, assicurando che solo attori legittimi possano accedere e agire nel sistema. Inoltre, si occupa della **generazione e gestione dei carbon credit**, fondamentali per incentivare comportamenti sostenibili. Le operazioni, come l'inserimento delle azioni compensative o

la condivisione dei dati tra gli utenti, vengono registrate attraverso **smart contract**, garantendo l'immutabilità e la verificabilità delle informazioni.

1.2 Analisi preliminare del rischio

Nella fase iniziale dell'Ingegneria dei requisiti, è fondamentale un'attenta valutazione dei potenziali rischi ai cui il sistema in fase di sviluppo potrebbe essere esposto. Questo perché la correzione di errori durante le fasi più avanzate dello sviluppo del software può rivelarsi estremamente costosa, in quanto implica la revisione completa di tutte le fasi del lavoro svolto, a partire dall'individuazione dell'errore fino alla fase iniziale di analisi dei requisiti. Tale processo richiede spesso la riprogettazione e l'implementazione di soluzioni per situazioni non considerate, con particolare attenzione agli errori legati alla sicurezza, che possono essere particolarmente onerosi e dannosi. Per affrontare questi rischi in modo efficace, è consigliabile adottare un approccio basato sulla gestione del rischio durante la specifica dei requisiti. Le attività principali svolte in questo contesto includono:

- **Identificazione del rischio**
- **Analisi del rischio**
- **Decomposizione del rischio**
- **Riduzione del rischio**

1.3 Identificazione del rischio

La fase di identificazione dei rischi è la prima fase cruciale nel processo di gestione del rischio. Durante questa fase, l'obiettivo principale è individuare e catalogare tutti i potenziali rischi che potrebbero influenzare il sistema che si sta progettando. Il primo passo è quello dell'identificazione degli asset, cioè delle risorse che rappresentano un certo valore per l'organizzazione e che potrebbero richiedere una protezione. Per ciascuno di essi occorre stabilire gli obiettivi funzionali e, in particolare, gli obiettivi di sicurezza e la politica di sicurezza da applicare.

1.3.1 Identificazione degli asset

Di seguito vengono riportati gli asset identificati e per ciascuno di essi i rispettivi obiettivi da rispettare e le politiche di utilizzo.

1. Inserimento dei dati sulle azioni compensative.
2. Accesso e condivisione relativi dei dati relativi ai carbon credit tra utenti autorizzati
3. Gestione di autenticazione e autorizzazione

Inserimento dei dati sulle azioni compensative

Indica la possibilità da parte dell'utente di poter inserire i propri dati relativi alle azioni effettuate per ridurre la CO₂. Gli obiettivi di sicurezza da rispettare sono:

- SO1 Integrità - I dati forniti non devono essere manomessi da altri utenti.
- SO2 Autenticazione - Solo l'utente autorizzato può fornire i dati.
- SO3 Autenticità - I dati forniti devono corrispondere a dati reali.

Accesso e condivisione dei dati relativi ai carbon credit tra utenti autorizzati

Questo asset indica la possibilità da parte degli utenti autorizzati di poter accedere alle informazioni condivise sulle azioni compensative e i carbon credit e di poter a loro volta condividere informazioni con gli altri. Obiettivi di sicurezza:

- SO1 Autorizzazione - Solo gli utenti autorizzati possono accedere ai dati condivisi.
- SO2 Disponibilità - I dati devono essere disponibili ogni qual volta richiesto dagli utenti.
- SO3 Confidenzialità - I dati non devono essere accessibili da utenti esterni. Gli utenti interni devono impegnarsi a non condividere dati con l'esterno.

Gestione di autenticazione e autorizzazione

L'asset si riferisce al processo di gestione e verifica dell'identità degli utenti, e alla gestione dell'accesso alle risorse e dati appropriati in base ai ruoli ricoperti. Obiettivi di sicurezza:

- SO1 Autenticazione - Devono essere rispettati i requisiti minimi per l'autenticazione degli utenti. Devono essere definiti inoltre i privilegi di accesso al sistema.
- SO2 Responsabilizzazione - Gli utenti devono rispettare le regole di accesso all'account.

1.4 Analisi dei rischi

L'analisi del rischio rappresenta una fase fondamentale all'interno di qualsiasi progetto che abbia a cuore la sicurezza del software. Essa si configura come un processo rigoroso e metodico volto a identificare, valutare e mitigare i potenziali eventi futuri che potrebbero compromettere l'integrità e la funzionalità del sistema in esame. L'obiettivo primario di questa fase è quello di ottenere una comprensione approfondita di cosa potrebbe accadere, fornendo una panoramica completa delle possibili minacce e delle loro implicazioni. Attraverso l'analisi del rischio, è possibile stimare la probabilità di un evento avverso e il suo potenziale impatto finanziario, sia in termini di costi diretti (ad esempio, per la riparazione del danno) che indiretti (ad esempio, perdita di reputazione o di clienti). Oltre a ciò, l'analisi del rischio permette di individuare le misure di sicurezza adeguate per mitigare o addirittura eliminare i rischi identificati. In questo contesto, si possono implementare diverse tipologie di controlli, tra cui controlli tecnici (firewall, sistemi di rilevamento delle intrusioni, crittografia), controlli procedurali (formazione sulla sicurezza per gli utenti, piani di risposta agli incidenti), e misure di riduzione della superficie di attacco (semplificazione dell'architettura del sistema, disattivazione di funzionalità non utilizzate, aggiornamento regolare del software). L'analisi del rischio non si esaurisce in un'unica fase, ma si configura come un processo continuo che richiede un monitoraggio costante e una revisione periodica. In questo modo, è possibile garantire che le misure di sicurezza adottate rimangano efficaci nel tempo e siano in grado di adattarsi alle nuove minacce e vulnerabilità che possono emergere.

1.4.1 Valutazione economica ed esposizione dell'asset

L'Asset Value Assessment e l'Exposure Assessment sono due pilastri fondamentali per la sicurezza informatica di qualsiasi organizzazione. Insieme, aiutano a identificare, valutare e proteggere i beni più preziosi dell'azienda,

minimizzando l'impatto di potenziali attacchi informatici

Valutazione economica

La valutazione economica dell'asset determina il valore patrimoniale di ciascun asset identificato nella fase precedente. Questo valore rappresenta l'importanza che l'asset ha per l'organizzazione in termini di:

- **Funzionalità:** l'impatto che l'asset ha sulla capacità dell'organizzazione di svolgere le proprie attività.
- **Dati:** la sensibilità e l'importanza dei dati contenuti nell'asset.
- **Reputazione:** il danno all'immagine e alla reputazione dell'azienda in caso di compromissione dell'asset.

La valutazione del valore patrimoniale aiuta a concentrare le risorse di sicurezza sugli asset più critici, garantendo una protezione ottimale.

Analisi esposizione dell'asset

L'analisi dell'esposizione dell'asset stima le potenziali perdite finanziarie e operative associate a ciascun asset in caso di attacco informatico riuscito. Si tratta di identificare i danni che l'organizzazione potrebbe subire se uno degli obiettivi di dependability (disponibilità, integrità, riservatezza) non venisse soddisfatto. L'esposizione viene quantificata in termini monetari, considerando fattori come:

- **Costi di ripristino:** il costo per ripristinare l'asset al suo stato originale dopo un attacco.
- **Perdita di dati:** il valore dei dati contenuti nell'asset che potrebbero essere persi o compromessi.
- **Interruzione del business:** il costo derivante dall'interruzione delle attività causata dall'attacco.

L'Exposure Assessment aiuta a comprendere le reali conseguenze di un attacco informatico e a giustificare gli investimenti in sicurezza.

Nella seguente tabella sono presenti tre colonne:

- **Asset:** specifica gli asset identificati,

- **Valore:** specifica il valore individuato per ciascuno di essi
- **Obiettivo:** specifica gli obiettivi di sicurezza che ciascun asset richiede di rispettare.

Asset	Valore	Obiettivo
Inserimento dei dati sulle azioni compensative.	Il rilascio dei token avviene in base ai dati relativi al consumo di CO2 da parte degli attori.	Integrità, Autenticazione, Autenticità
Accesso e condivisione relativi ai carbon credit dei dati tra utenti autorizzati	Facilitare la collaborazione tra i vari utenti coinvolti e la condivisione sicura dei dati.	Autorizzazione, Disponibilità, Confidenzialità
Gestione di autenticazione e autorizzazione	Verificare l'identità degli utenti attraverso meccanismi come credenziali, biometria, autenticazione a due fattori allo scopo di garantire un accesso sicuro. Assegnazione di autorizzazioni specifiche agli utenti in base al ruolo ricoperto.	Autorizzazione, Responsabilità

Table 1.1: Descrizione degli asset, dei valori associati e degli obiettivi di sicurezza

1.4.2 Identificazione delle minacce

L'identificazione delle minacce è un altro processo fondamentale per la sicurezza informatica. Essa consiste nell'individuare potenziali intrusioni o violazioni che potrebbero compromettere gli obiettivi di sicurezza di un sistema. Un modello ampiamente utilizzato per l'identificazione delle minacce è STRIDE.

Questo modello si basa su sei categorie di minacce:

- **Spoofing:** Assunzione di un'identità falsa.
- **Tampering:** Alterazione di dati o sistemi.
- **Repudiation:** Negazione di un'azione o di un evento.
- **Information Disclosure:** Divulgazione non autorizzata di informazioni.
- **Denial of Service:** Interruzione o limitazione dell'accesso a un servizio.

- **Elevation of Privilege:** Acquisizione di privilegi non autorizzati.

Per una visione più ampia che includa gli obiettivi di affidabilità, è stato sviluppato il modello DUAL-STRIDE. Questo modello estende STRIDE con tre nuove categorie:

- **Danger:** Pericolo per la salute o la sicurezza fisica.
- **Unreliability:** Incapacità di un sistema di funzionare correttamente.
- **Absence of Resilience:** Mancanza di capacità di recuperare da un guasto.

Nella fase finale di questa analisi, le minacce identificate vengono registrate in una tabella STRIDE. Nella Figura 1.1 , che rappresenta le prime colonne della tabella DUAL-STRIDE relativa alla fase di Threat identification, per ciascun asset viene inserita una X ogni volta che viene individuata una minaccia che potrebbe violare un requisito di sicurezza. Questo fornisce un quadro chiaro delle potenziali minacce per ciascun asset e aiuta a guidare le decisioni su come mitigare questi rischi. L'identificazione delle minacce è un passo fondamentale per la sicurezza informatica. L'utilizzo di modelli come STRIDE e DUAL-STRIDE facilita questo processo, permettendo di individuare e mitigare le potenziali minacce che potrebbero compromettere un sistema.

Asset	Value	Spoofing	Tampering	Repudiation	Information disclosure	DOS	Elevation of privilege	Danger	Unavailability	Absence of Resilience	Exposure
Inserimento dati sulle azioni compensative	120000-360000	x	x			x					80000-240000
Accesso e condivisione dei dati relativi ai carbon credit tra utenti autorizzati	120000-480000	x		x	x	x					90000-360000
Gestione autenticazione e autorizzazione	200000-400000	x		x		x	x		x		180000-360000

Figure 1.3: Modello STRIDE

1.5 Decomposizione del rischio

La fase di Risk Decomposition è un processo fondamentale nella gestione del rischio. Si tratta di scomporre il rischio in più componenti o fattori per analizzare e comprendere meglio la sua natura. Questo processo consente di identificare le azioni di mitigazione più efficaci. Durante questa fase, si valutano vari elementi. Ad esempio, si considera il tipo di minaccia, che può variare a seconda del contesto. Si esamina anche la vulnerabilità del sistema, che può essere influenzata da una serie di fattori, tra cui la sicurezza fisica e informatica, le procedure operative e la formazione del personale. Inoltre, si valuta l'impatto potenziale del rischio su persone, risorse o processi aziendali. Questo può includere danni fisici, perdite finanziarie, interruzioni delle operazioni o danni alla reputazione. Infine, si prendono in considerazione le misure preventive in atto. Queste possono includere politiche e procedure di sicurezza, sistemi di allarme, piani di emergenza e formazione del personale. In sostanza, la Risk Decomposition aiuta a individuare e valutare separatamente le componenti del rischio. Questo consente di ottenere una comprensione più approfondita della situazione e di sviluppare strategie di mitigazione più efficaci.

1.5.1 Attacco agli asset

L'Attack Assessment è un processo che prevede la valutazione degli attacchi e il modo in cui una minaccia può manifestarsi. Questo viene fatto ipotizzando vari scenari, chiamati casi d'uso, e valutando cosa potrebbe accadere in ciascuno di essi. Per esaminare gli attacchi, è utile immaginare scenari illegittimi, ovvero situazioni in cui le cose vanno male perché qualcuno sta usando il sistema in modo improprio, sia accidentalmente che intenzionalmente. Questi scenari sono classificati come casi di misuse, quando si tratta di un'azione sbagliata svolta accidentalmente, e casi di abuso, quando si tratta di un'azione sbagliata svolta intenzionalmente. In sintesi, l'Attack Assessment è un processo fondamentale per comprendere come una minaccia può realizzarsi e per sviluppare strategie efficaci per prevenire o mitigare tali minacce.

Asset "Accesso e condivisione dei dati relativi ai carbon credit tra utenti autorizzati"

I casi di uso, abuso e misuse vengono descritti tramite delle tabelle chiamate Schemi di Jacobson. Ad esempio, nel contesto della risorsa Data processing and input, sono stati riportati due scenari impropri, visualizzati nei relativi schemi di Jacobson. Il primo è un caso di misuse, denominato *Condivisione involontaria dei dati relativi ai carbon credit tra utenti* (Figura 1.2). In questa situazione, un utente autorizzato lascia i dati incustoditi, dando la possibilità ad un utente non autorizzato di visualizzarli e/o divulgarli.

Case Type	Misus	Case	MC-01-01
Case	Condivisione involontaria dei dati relativi ai carbon credit tra utenti		
Actors	Utente maldestro		
Description	Un utente autorizzato lascia incustoditi i dati condivisi dagli altri utenti. Di conseguenza, un utente non autorizzato è in grado di visualizzarli ed eventualmente divulgarli a chi normalmente non può averne accesso.		
Data	Accesso e condivisione dei dati relativi ai carbon credit tra utenti autorizzati		
Stimulus and	Utente poco preparato sulle procedure di sicurezza o negligente.		
Attack Flow 1	Un utente autorizzato lascia incustodito il dispositivo con cui ha effettuato l'accesso al sistema, consentendo ad un utente malintenzionato di accedere e condividere i dati condivisi da tutti gli utenti registrati.		
Attack Flow 2	Un utente autorizzato, senza intenzioni malevoli, condivide le informazioni condivise o partesse ad utenti malintenzionati. Questo può avvenire, ad esempio, tramite una foto condivisa dall'utente che contiene anche la schermata del dispositivo di accesso.		
Response and	Tutti i dati condivisi dagli utenti autorizzati o parte di essi finiscono nelle mani di utenti a cui normalmente non dovrebbero avere accesso.		
Non Functional Requirements			
Mitigations	Percorso di formazione e sensibilizzazione obbligatoria degli utenti autorizzati mirato ad istruire sui possibili rischi alla sicurezza e sulle loro conseguenze, deve essere svolto e completato prima di iniziare ad utilizzare il sistema. Implementazione di un meccanismo di disconnessione automatica dal sistema in caso di un periodo di inattività.		

Figure 1.4: Misuse Case: Condivisione involontaria dei dati relativi ai carbon credit tra gli utenti

Il secondo è, invece, un caso di abuso, denominato "accesso malevolo ai dati condivisi tra gli utenti". Rappresenta lo scenario in cui un utente malevolo, tramite attacco, riesce ad accedere ai dati relativi ai carbon credit, con lo scopo di utilizzare questi dati a suo vantaggio.

Case Type	Abuse Case	Case ID	AC-01
Case Name	Accesso malevolo ai dati condivisi tra gli utenti		
Actors	Attaccante		
Description	Un utente malevolo, tramite attacco informatico, riesce ad accedere ai dati relativi ai carbon credit condivisi o a parte di essi con l'obiettivo di usarli a proprio vantaggio.		
Data	Accesso e condivisione dei dati relativi ai carbon credit tra utenti autorizzati		
Stimulus and precondition	Preparazione inadeguata o mancante da parte degli utenti autorizzati. Il sistema dispone di meccanismi di sicurezza non adeguati.		
Attack Flow 1	L'attaccante, attraverso metodi come phishing, ottiene l'accesso ad informazioni riservate.		
Attack Flow 2	L'attaccante, tramite attacco bruteforce, riesce ad ottenere l'accesso ad informazioni riservate.		
Attack Flow 3	L'attaccante intercetta i dati condivisi inseriti erroneamente nel sistema da un utente maldestro.		
Response and Postcondition	L'attaccante può usare le informazioni acquisite per diversi scopi: ricattare utenti specifici, rivenderle ad altri, condividerle per arrecare danni alla piattaforma e agli utenti coinvolti.		
Non Functional Requirements			
Mitigations	Formazione del personale per la sensibilizzazione sui possibili tipi di attacco e sulle modalità di difesa come, ad esempio, autenticazione a due fattori. Implementazione di un numero limitato di tentativi di accesso, in caso di raggiungimento del limite il sistema blocca automaticamente la possibilità di accedere e solo un amministratore può riattivare la procedura previo avviso dell'utente.		

Figure 1.5: Abuse case: Accesso malevolo ai dati condivisi tra gli utenti

Asset "Gestione di autenticazione e autorizzazione"

Per il secondo asset preso in esame, un utente autorizzato, senza intenzioni malevole, condivide le sue credenziali di accesso.

Case Type	Misuse Case	Case ID	MC-02
Case Name	Condivisione accidentale delle credenziali di accesso al sistema		
Actors	Utente maldestro		
Description	Un utente autorizzato, senza avere intenzioni malevoli, condivide le credenziali di accesso a soggetti non autorizzati.		
Data	Gestione di autenticazione e autorizzazione		
Stimulus and precondition	Utente poco preparato sulle procedure di sicurezza o negligente.		
Attack Flow 1	Un utente autorizzato condivide accidentalmente le credenziali di accesso, permettendo così ad utenti malintenzionati di ottenerle ed eventualmente condividerle. Questo può avvenire, ad esempio, scrivendole su un foglio di carta che a sua volta viene lasciato incustodito.		
Attack Flow 2	Un utente autorizzato accede su un dispositivo appartenente ad un'altra persona senza però disconnettersi una volta finito il suo lavoro.		
Response and Postconditions	Utenti malintenzionati entrano in possesso delle credenziali, e possono usarle a loro piacimento. Alternativamente, in caso di mancata disconnessione, riescono ad accedere senza aver bisogno delle credenziali.		
Non Functional			
Mitigations	<p>Percorso di formazione e sensibilizzazione obbligatoria degli utenti autorizzati mirato ad istruire sui possibili rischi alla sicurezza e sulle loro conseguenze, deve essere svolto e completato prima di iniziare ad utilizzare il sistema.</p> <p>Implementazione di meccanismi aggiuntivi in fase di autenticazione: autenticazione a due fattori ed inserimento di una OTP (One Time Password). Implementazione di un meccanismo di disconnessione automatica dopo un determinato periodo di autenticazione oppure dopo un periodo di inattività.</p>		

Figure 1.6: Misuse case: Condivisione accidentale delle credenziali di accesso al sistema

Il caso di abuso invece, riguarda un utente malintenzionato che riesce ad effettuare l'accesso al sistema anche senza autorizzazione.

Case Type	Abuse Case	Case ID	AC-02
Case Name	Accesso malevolo al sistema		
Actors	Attaccante		
Description	Un utente malintenzionato riesce ad effettuare un accesso non autorizzato al sistema.		
Data	Gestione di autenticazione e autorizzazione		
Stimulus and precondition	Utente poco preparato sulle procedure di sicurezza o negligente. Sistemi di protezione non adeguati.		
Attack Flow 1	Un utente malintenzionato, tramite un attacco bruteforce, riesce ad ottenere le credenziali di accesso di un determinato utente.		
Attack Flow 2	Un utente malintenzionato riesce ad intercettare la richiesta di autenticazione di un determinato utente attraverso un attacco man in the middle e, di conseguenza, ad intercettare le credenziali di accesso.		
Attack Flow 3	Un utente malintenzionato, tramite social engineering, riesce ad ottenere le credenziali di accesso di uno o più utenti autorizzati.		
Attack Flow 4	Un utente malintenzionato ruba le credenziali da supporti fisici di un utente maldestro.		
Response and Postconditions	Si verifica un accesso malevolo al sistema, il malintenzionato ha accesso a tutte le informazioni degli utenti che impersona e alle informazioni condivise dagli altri utenti.		
Non Functional Requirements			
Mitigations	Percorso di formazione e sensibilizzazione obbligatoria degli utenti autorizzati mirato ad istruire sui possibili rischi alla sicurezza e sulle loro conseguenze, deve essere svolto e completato prima di iniziare ad utilizzare il sistema. Implementazione di meccanismi aggiuntivi in fase di autenticazione: autenticazione a due fattori, inserimento di una OTP (One Time Password), tentativi di accesso limitati, crittografia del canale di comunicazione. Implementazione di un numero limitato di tentativi di accesso, in caso di raggiungimento del limite il sistema blocca automaticamente la possibilità di accedere e solo un amministratore può riattivare la procedura previo avviso dell'utente.		

Figure 1.7: Abuse case: Accesso malevolo al sistema

Asset "Inserimento dei dati sulle azioni compensative"

Per il terzo asset considerato, abbiamo il caso di misuse "Inserimento di dati errati sulle azioni compensative", dove un utente autorizzato, inserisce erroneamente i dati relativi alle azioni compensative effettuate.

Case Type	Misuse Case	Case ID	MC-03
Case Name	Inserimento di dati errati sulle azioni compensative		
Actors	Utente maldestro		
Description	Un utente autorizzato, senza intenzioni malevoli, inserisce nella piattaforma dati sulle azioni compensative errati da un punto di vista semantico o del valore stesso dei dati.		
Data	Inserimento dei dati sulle azioni compensative		
Stimulus and precondition	Utente poco preparato all'uso della piattaforma o poco attento		
Attack Flow 1	Un utente autorizzato, anche se non ha intenzioni malevoli, inserisce un certificato in un formato diverso da quello atteso dal sistema, causando una sua interpretazione errata.		
Attack Flow 2	Un utente autorizzato, anche se non ha intenzioni malevoli, inserisce un certificato nel formato corretto ma contenente dati dal valore errato, causando un rilascio di carbon credit diverso da quello corrispondente alla realtà.		
Response and Non Functional Requirements	Il sistema non provvede al rilascio dei carbon credit dovuti oppure i dati dal valore errato alterano il numero dei carbon credit rilasciati.		
Mitigations	Maggiore attenzione e preparazione da parte degli utenti autorizzati nell'inserimento dei certificati. Sistemi di controllo, e validazione dei dati in input.		

Figure 1.8: Inserimento di dati errati sulle azioni compensative

Per il caso di abuso, inserimento di dati malevoli sulle azioni compensative, abbiamo il caso in cui un utente non autorizzato, intenzionalmente inserisce dati errati sulle azioni compensative, con lo scopo di alterare i dati degli utenti.

Case Type	Abuse Case	Case ID	AC-03
Case Name	Inserimento di dati malevoli sulle azioni compensative		
Actors	Attaccante		
Description	Un utente malevolo riesce a inserire informazioni malevoli all'interno del sistema con l'obiettivo di alterare i dati relativi alle azioni compensative svolte dall'utente.		
Data	Inserimento dei dati sulle azioni compensative		
Stimulus and precondition	Alterazione dei dati relativi alle azioni compensative della CO2		
Attack Flow 1	Un attaccante riesce ad ottenere accesso alla piattaforma sfruttando un meccanismo di autenticazione debole con l'obiettivo di alterare i dati relativi alle azioni per compensare la CO2 emessa, causando il mancato rilascio dei carbon credit e/o la perdita di reputazione.		
Attack Flow 2	Un attaccante crea dati falsi che riesce a inserire nella piattaforma attraverso un utente maldestro.		
Response and Postconditions	I dati malevoli inseriti alterano il numero di carbon credit rilasciati dal sistema.		
Non Functional Requirements			
Mitigations	Implementare le seguenti misure di sicurezza: autenticazione a più fattori, crittografia dei dati, formazione del personale sulla sicurezza informatica ed uso di software aggiornati e protetti. Creazione di un piano di risposta agli incidenti per affrontare rapidamente ogni violazione dei dati.		

Figure 1.9: Abuse case: inserimento di dati malevoli sulle azioni compensative

L'Attack Assessment prosegue con l'analisi dei rischi tramite gli attack trees, che sono uno strumento di analisi per modellare e visualizzare in modo sistematico i potenziali scenari di attacco che possono minacciare il software, attraverso una rappresentazione grafica della possibile sequenza di azioni che un aggressore può intraprendere per effettuare un attacco. Gli attack trees sono rappresentati da una struttura gerarchica con un nodo radice che rappresenta la tipologia di attacco che viene suddiviso in ulteriori nodi e foglie che delineano i passaggi e le azioni compiute dall'attaccante per raggiungere quell'obiettivo. Questo strumento permette di effettuare una chiara identificazione delle vulnerabilità di sistema, le debolezze di progettazione e le azioni malevole che potrebbero essere sfruttate da parte degli attaccanti, permettendo di identificare i rischi e le minacce e fornendo una base solida per lo sviluppo di strategie di mitigazione e di difesa.

Nella Figura 1.8 è riportato l'attack tree riferito all'asset **Accesso e condivisione relativi ai carbon credit tra utenti autorizzati**. Per questo asset sono stati individuati due principali scenari: uno di abuso intenzionale da parte di un attaccante e uno di misuse causato da un utente maldestro. Nel primo caso, un attaccante tenta di ottenere accesso dannoso ai dati condivisi, sfruttando vulnerabilità legate all'intercettazione dei dati o mediante tecniche di identity spoofing. Quest'ultima viene realizzata attraverso schemi di phishing o attacchi di password brute forcing, che permettono all'attaccante di impersonare un utente legittimo e ottenere accesso non autorizzato. Il fine ultimo è la lettura e diffusione indebita dei dati. Nel secondo scenario, un utente maldestro compromette la riservatezza dei dati condivisi lasciandoli incustoditi o esposti, facilitando così la loro divulgazione. Parallelamente, anche l'oracolo – soggetto autorizzato alla lettura dei dati – può diventare un punto critico se inavvertitamente divulga dati sensibili, rappresentando un ulteriore rischio per la confidenzialità. Questi scenari evidenziano come sia fondamentale implementare misure di sicurezza robuste, sia per limitare gli accessi non autorizzati sia per sensibilizzare gli utenti legittimi sulle corrette pratiche di gestione dei dati condivisi.

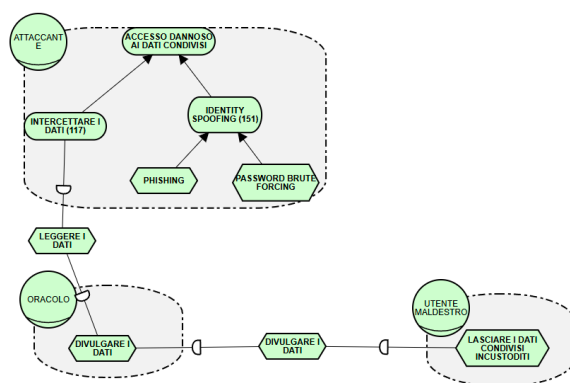


Figure 1.10: Attack tree: Accesso dannoso ai dati condivisi

Nella Figura 1.9 è riportato l'attack tree riferito all'asset **Inserimento dei dati sulle azioni compensative**. Per questo asset sono stati individuati un caso di abuso (Malicious Data Input) e uno di misuse (Invalid Data Entry). Nel primo scenario, un attaccante inserisce dati malevoli all'interno del sistema sfruttando due principali tecniche: l'abuso del meccanismo di autenticazione e la falsificazione dell'origine dei dati. Nel primo caso, l'attaccante esegue l'attacco dopo aver pianificato l'azione e raccolto informazioni utili, sfruttando potenziali errori presenti nel sistema per identificare vulnerabilità e ottenere accesso non autorizzato. Nel secondo

caso, l'attaccante crea e inserisce dati falsi al fine di falsificare la loro origine, compromettendo l'integrità delle informazioni condivise. Entrambe le strategie convergono sull'obiettivo finale di identificare il bersaglio, ovvero soggetti o componenti vulnerabili del sistema, con lo scopo di causare valori errati nel processo di input e trattamento dei dati, alterando la veridicità e affidabilità delle informazioni elaborate. Parallelamente, nel secondo scenario, un utente maldestro introduce involontariamente dati errati nel sistema – ad esempio informazioni personali imprecise – contribuendo involontariamente al degrado della qualità dei dati e aprendo possibili superfici di attacco. Questo attack tree evidenzia l'importanza di misure di controllo sugli input, meccanismi di validazione robusti e una corretta gestione delle identità e delle autorizzazioni.

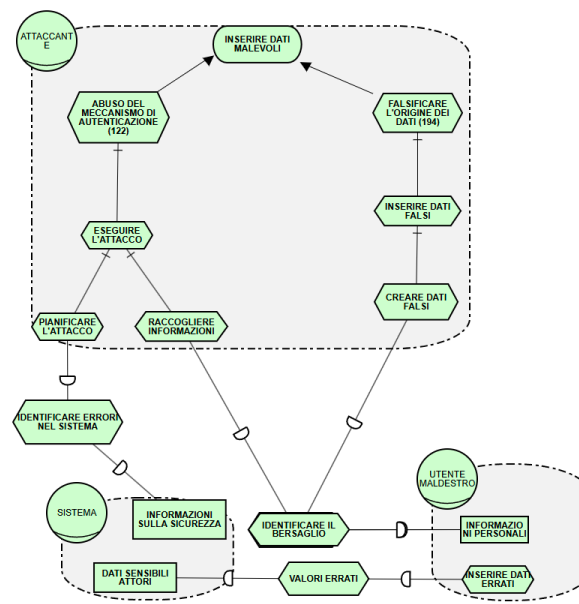


Figure 1.11: Attack tree: Inserimento dati malevoli

Nella Figura 1.12 è riportato l'attack tree riferito all'asset **Gestione autenticazione e autorizzazione**. Per questo asset sono stati individuati un caso di abuso da parte di un attaccante e uno di misuse involontario da parte dell'utente. Nel primo scenario, l'attaccante mira a ottenere accesso malevolo al sistema attraverso il furto delle credenziali di accesso. Le credenziali possono essere ottenute in diversi modi: tramite social engineering, furto fisico dei supporti, intercettazione delle credenziali in transito, o con tecniche di brute forcing. Una volta ottenute le credenziali, l'attaccante può compromettere il sistema accedendo ad aree riservate o manipolando dati sensibili come i dati ATM. Nel secondo scenario, un utente maldestro contribuisce involontariamente all'attacco, ad

esempio rendendo disponibili le proprie credenziali, lasciando esposte informazioni personali (come l'email) o comportandosi in modo negligente. Tali azioni facilitano l'ottenimento di accesso non autorizzato da parte di terzi. L'obiettivo finale dell'attaccante è accedere all'area riservata del sistema, aggirando i meccanismi di autenticazione. Questo scenario sottolinea la necessità di robuste politiche di gestione delle credenziali, sensibilizzazione degli utenti contro l'ingegneria sociale, e l'uso di sistemi di autenticazione forte per proteggere gli asset critici.

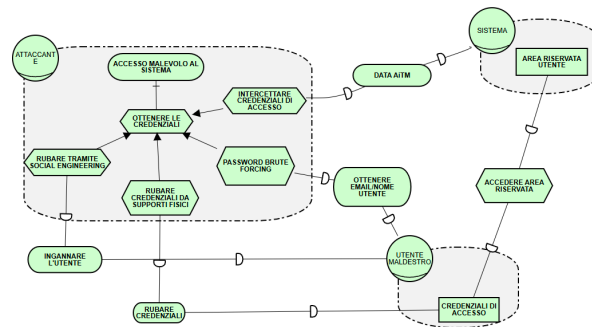


Figure 1.12: Attack tree: Accesso malevolo al sistema

Terminata questa fase, si riportano ulteriori colonne della tabella STRIDE come in Figura 1.13.

Exposure	Attack	CAPEC Pattern	Inherent Probability	Inherent Risk	Control
30000-240000	Inserimento malevolo dei dati	Abuso del meccanismo di autenticazione (122)	19%	15200-45600	Two-factor authentication
		Falsifica l'origine dei dati (194)	13%	10400-31200	Data source verification
30000-360000	Accesso dannoso ai dati condivisi	Interception (117)	19%	17100-68400	Data encryption
		Identity spoofing (151)	18%	16200-64800	Two-factor authentication
		Phishing (98)	15%	13500-54000	
		Password brute forcing (49)	14%	12600-50400	Login attempts limiting
180000-360000	Accesso malevolo al sistema	Social Engineering (403)	19%	34200-68400	User awareness
		Password brute forcing (49)	14%	25200-50400	Login attempts limiting
		Use of Known domain credentials (560)	16%	28800-57600	User awareness
		Interception (117)	19%	34200-68400	Data encryption

Figure 1.13: Continuo tabella STRIDE

1.6 Riduzione del rischio

La fase di Riduzione del Rischio si riferisce al processo di individuazione e attuazione di misure preventive di sicurezza, volte a diminuire la vulnerabilità e prevenire possibili attacchi. L'obiettivo principale è valutare le contromisure di sicurezza possibili per mitigare il rischio. Ridurre il rischio può significare diminuire la probabilità di attacco o ridurre l'impatto; nel migliore dei casi, può comportare la riduzione di entrambi.

1.6.1 Identificazione dei controlli

Considerando gli attacchi e i rischi associati a ciascuno di essi, si sono individuati meccanismi di controllo volti a limitare i danni. Nell'analisi delle diverse soluzioni possibili, non basta semplicemente elencare le singole soluzioni, ma è essenziale considerare anche la possibilità di combinare tali soluzioni. In questo modo, si può ottenere un approccio più completo e sinergico alla gestione dei rischi.

1.6.2 Valutazione della fattibilità

Per ogni meccanismo è stata condotta una valutazione completa che ha preso in considerazione non solo i costi di implementazione, ma anche la fattibilità e la praticità d'uso; in particolare, si è esaminata la concreta realizzabilità di ogni soluzione e la sua convenienza nel lungo periodo. Inoltre, è stata valutata l'influenza del nuovo meccanismo sull'usabilità complessiva del software, al fine di garantire un'esperienza utente ottimale. Non sempre la soluzione che minimizza maggiormente il rischio è quella più conveniente a livello economico, per questo motivo si è deciso di implementare un'ulteriore misura, il Return of Control che può essere espresso con la seguente formula:

$$\begin{aligned}\text{Return of Control} &= \frac{\text{Reduction of expected loss}}{\text{Control of cost}} - 1 \\ &= \frac{\text{Riskinherent}(R_i) - \text{Riskresidual}(R_r)}{\text{Control of cost}} - 1\end{aligned}$$

Figure 1.14: Equazione ROI

1.6.3 Definizione dei requisiti di sicurezza

In seguito ai risultati derivanti dallo studio di fattibilità, le misure di controllo da implementare sono state chiaramente stabilite. Sulla base dei

risultati ottenuti dall'analisi di fattibilità, sono state definite con chiarezza le misure di controllo da applicare. Al termine di questa fase è stata completata la tabella STRIDE (Figure 1.31 e 1.32). In particolare, sono state compilate le seguenti colonne:

- **Control:** meccanismi di controllo individuati per prevenire, mitigare o gestire le minacce precedentemente definite.
- **Cost:** stima del costo necessario per implementare un meccanismo di controllo.
- **Feasibility:** valutazione del grado di fattibilità del rispettivo controllo.
- **Residual Probability:** stima della probabilità di occorrenza residua di un attacco, ossia della probabilità che un attacco si verifichi dopo che sono stati adottati i meccanismi di controllo
- **Residual Impact:** stima dell'impatto residuo, ossia dell'effetto della verifica di un rischio una volta applicati i meccanismi di controllo
- **Residual Risk:** stima del rischio residuo, ossia del rischio rimanente dopo che sono stati adottati i meccanismi di controllo
- **RoC:** valore del Return of Control
- **Overall Cost:** Stima del costo complessivo

Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk	R.o.C	Overall Cost
Two-factor authentication	8000	L'autenticazione a due fattori può essere utilizzata per mitigare il rischio di abuso del meccanismo di autenticazione	10%	80000-240000	8000-24000	1.7	32.000
Data source verification	3000	La verifica della sorgente dei dati può essere utilizzata per mitigare il rischio di falsificazione dell'origine dei dati	5%		4000-12000	5.4	15.000
Data encryption	10.000	La crittografia dei dati può essere utilizzata per mitigare il rischio di intercettazione	5%		4500-18000	4	28000
Two-factor authentication	8000	L'autenticazione a due fattori può essere utilizzata per mitigare il rischio di identity spoofing	13%		11700-46800	1.3	54800
User awareness	1000	L'implementazione di un programma di formazione del personale può essere utile al fine di mitigare il rischio di phishing	12%	90000-360000	10800-43200	9.8	44200
Login attempts limiting	2000	L'implementazione di un limite al numero di tentativi di accesso può risultare efficace per mitigare il rischio di password brute forcing	4%		3600-14400	17	16400
User awareness	1000	L'implementazione di un programma di formazione del personale può essere utile al fine di mitigare il rischio di social engineering	16%		28800-57600	9.8	58600
Login attempts limiting	2000	L'implementazione di un limite al numero di tentativi di accesso può risultare efficace per mitigare il rischio di password brute forcing	4%	180000-360000	7200-14400	17	16400
User awareness	1000	L'implementazione di un programma di formazione del personale può essere utile al fine di mitigare il rischio di condivisione involontaria delle credenziali	8%		14400-28800	27.8	29800
Data encryption	10.000	La crittografia dei dati può essere utilizzata per mitigare il rischio di intercettazione	5%		9000-18000	4	28000

Figure 1.15: Continuo tabella STRIDE

Chapter 2

Progettazione sicura

Durante la fase di analisi dei requisiti, le specifiche definite spesso non sono sufficienti per la completa definizione di un software. Pertanto, è fondamentale progettare l'architettura del software e definire l'interazione tra i suoi componenti. Le scelte tecnologiche e l'architettura utilizzata hanno un impatto significativo sulla sicurezza e sul raggiungimento degli obiettivi di affidabilità per questo è importante valutare attentamente le opzioni disponibili al fine di garantire il rispetto degli obiettivi di sicurezza stabiliti. Durante la fase di progettazione, tutte le decisioni prese devono essere conformi alle policy di sicurezza identificate nella fase di analisi dei rischi.

2.1 Linee guida e buone pratiche

Nel contesto della progettazione di software, sono state sviluppate nel tempo delle linee guida e buone pratiche per sensibilizzare gli sviluppatori sui potenziali rischi alla sicurezza che possono presentarsi ed aiutarli nella progettazione di software sicuri. Queste linee guida forniscono indicazioni specifiche su come affrontare procedure del tipo: la gestione dei dati, protezione delle informazioni sensibili e prevenzione delle vulnerabilità. Tra i diversi cataloghi che definiscono queste linee guida e buone pratiche i più famosi sono:

- **Saltzer and Schroeder** (rilasciato nel 1975)
- **OWASP** (rilasciato nel 2016)
- **Sommerville** (rilasciato nel 2017)

Nello sviluppo di questo progetto sono stati presi in considerazione alcuni dei principi elencati in questi cataloghi citati.

2.1.1 Principi di progettazione di Saltzer and Schroeder

Sono stati seguiti i seguenti principi di design del software di Saltzer and Schroeder:

- **Least privilege:** secondo questo principio, ogni processo dovrebbe disporre solo dei privilegi strettamente necessari per svolgere le proprie funzioni. Nel progetto è stato applicato attraverso un sistema di controllo degli accessi basato sui ruoli, che garantisce agli utenti solo le autorizzazioni coerenti con le loro mansioni. Inoltre, è stata attuata una separazione dei compiti tra sistemi e persone, per ridurre il rischio che un singolo soggetto possa sfruttare impropriamente i propri permessi.
- **Keep it simple, stupid ("KISS"):** secondo tale principio, è bene mantenere i sistemi quanto più semplici possibili, sia per gli utenti che per gli sviluppatori, al fine di migliorarne la sicurezza; infatti, un sistema troppo complesso aumenta la probabilità di errori nella loro implementazione e nel loro utilizzo, con conseguente aumento della probabilità di comportamenti anomali ed esposizione ad attacchi.
- **Complete mediation:** tale principio afferma che ogni accesso a una risorsa deve essere autorizzato e verificato, non solo il primo. Questo significa che il sistema non deve fare affidamento su cache di permessi o controlli parziali: ogni volta che un'entità accede ad un file, una funzione o un dato, deve passare attraverso un controllo di accesso completo. Questo principio è stato applicato garantendo che tutte le richieste a risorse sensibili vengano sottoposte a controlli di autorizzazione, evitando che un utente, dopo un primo accesso valido, possa continuare ad agire senza ulteriori verifiche. In questo modo si riducono i rischi legati a escalation di privilegi o abusi successivi a un'autenticazione iniziale.

2.1.2 OWASP: principi di sicurezza

Dal catalogo OWASP sono stati seguiti i seguenti principi per la progettazione ed implementazione:

- **Defense in Depth:** stabilisce che la difesa contro gli attacchi è implementata attraverso più controlli di sicurezza. In questo modo se un controllo viene violato, gli altri presenti sono in grado di mitigare o

eliminare del tutto gli effetti dell'attacco. Corrisponde alla voce Avoid a single point of failure nel catalogo Sommerville.

- **Fail safe:** indica la capacità del software di raggiungere uno stato sicuro nel caso si verifichi un errore durante l'esecuzione, indipendentemente se causato da un attacco esterno oppure da difetti di progettazione e/o implementazione. In questo modo, il sistema ha maggiori probabilità di mantenere le condizioni di confidenzialità, integrità e disponibilità nonostante eventi imprevisti. Nel catalogo Sommerville è indicato con il nome Fail securely.
- **Economy of Mechanism** (noto anche come Keep it simple): consiglia di scegliere un'implementazione (del sistema o di una risorsa) semplice e intuitiva per gli sviluppatori. Questo perché le probabilità di un attacco aumentano insieme alla complessità dell'architettura del sistema, e vengono ulteriormente aumentate se il codice scritto è difficile da interpretare e mantenere.

2.1.3 Linee guida di Sommerville

Sulla base delle linee guida di Sommerville sono stati seguiti i seguenti principi per la progettazione ed implementazione:

- **Validate all inputs:** controllare che tutti gli input degli utenti rispettino forma e valori prestabiliti, in modo prevenire i problemi causati da input inattesi.
- **Log user actions:** consiglia di implementare un sistema di logging, in modo da analizzare le azioni degli utenti e identificare chi le ha compiute. Consiglia, inoltre, di informare gli utenti della presenza di questo sistema, in modo da disincentivare comportamenti irresponsabili.

2.2 Design architetturale

Nel campo dell'ingegneria del software, il design architetturale rappresenta la fase di progettazione di alto livello in cui viene definita la

struttura generale di un sistema software. Questo processo comprende l'identificazione dei componenti principali, la specifica delle loro interazioni e l'individuazione dei meccanismi di comunicazione tra di essi. L'obiettivo principale è realizzare un'architettura solida, scalabile e facilmente manutenibile, in grado di soddisfare sia i requisiti funzionali che quelli non funzionali del sistema. È stata scelta un'implementazione modulare basata sul pattern Model-View-Controller (MVC), con l'obiettivo di ridurre i punti di accesso e semplificare la progettazione complessiva del sistema. Il flusso operativo ha inizio dall'utente, il quale interagisce esclusivamente con la parte off-chain tramite linea di comando. Questo livello si occupa della gestione degli input e delle richieste provenienti dall'utente autenticato, inoltrandole ai moduli sottostanti. Tale approccio consente di limitare l'accesso diretto dell'utente al sistema, riducendo sia la superficie di attacco esterna sia i privilegi assegnati. La componente off-chain è responsabile dell'elaborazione delle richieste, interfacciandosi con il database per la gestione dei dati e comunicando con i models. Inoltre, può interagire con la parte on-chain attraverso le API di Web3, effettuando chiamate allo smart contract. Quest'ultimo gestisce le operazioni previste, incluse la creazione delle strutture dati e l'esecuzione delle azioni implementate al suo interno.

2.2.1 Isolation, Obfuscation, Monitoring

In generale, i meccanismi di dependability e security vengono definiti meccanismi di difesa, in quanto rappresentano entità o procedure incaricate di applicare specifiche policy relative al livello di fiducia attribuito a un sistema. La protezione dell'integrità delle risorse informatiche si basa su alcuni principi fondamentali:

- **Isolamento:** impedisce a un programma di interferire o accedere alle funzioni di un altro, garantendo così l'integrità e l'indipendenza dei processi.
- **Monitoraggio:** consente un controllo continuo del sistema, intervenendo ogniqualvolta venga rilevata un'azione non conforme alle politiche di sicurezza adottate.
- **Offuscamento:** protegge codice e dati, rendendoli leggibili solo a chi possiede una chiave segreta di decodifica.

Nell'applicazione sviluppata, questi principi sono stati attentamente considerati e integrati nella fase di implementazione. Nello specifico, l'isolamento

è stato assicurato attraverso una rigorosa gestione di permessi e ruoli, suddividendo l'applicazione in moduli con responsabilità ben definite. Ogni modulo, come descritto nella scelta architetturale, ha accesso esclusivamente alle risorse necessarie per eseguire le proprie funzioni, riducendo così i rischi in caso di compromissione di un singolo componente. Questo approccio è reso possibile dall'adozione del pattern MVC e dall'utilizzo di una blockchain locale come Hyperledger Besu, che fornisce un ambiente di esecuzione isolato e sicuro per il deployment degli smart contract. In aggiunta, sono state implementate policy di sicurezza basate sulla separazione logica dei dati, garantendo che solo specifiche componenti autorizzate possano accedere a determinate informazioni. Ciò avviene, ad esempio, attraverso la gestione dei ruoli utente in fase di registrazione e la limitazione dell'esecuzione di determinate azioni ai soli utenti autorizzati e riconosciuti.

Il monitoraggio all'interno dell'applicazione è stato implementato su più livelli per garantire un controllo efficace e continuo.

- A livello **Off-chain**, come descritto in precedenza e illustrato attraverso estratti di codice, è stato introdotto un sistema di log e monitoraggio delle eccezioni, che consente di individuare anomalie o comportamenti sospetti. Inoltre, viene mantenuto un registro dettagliato di tutte le attività, comprese le transazioni, le modifiche ai dati e gli accessi degli utenti, attraverso un apposito file di log delle azioni.
- A livello **On-chain**, il controllo degli accessi e la validazione delle condizioni sono stati implementati mediante costrutti come *modifier* e *require*. I modifier permettono di aggiungere restrizioni o condizioni alle funzioni, modificandone il comportamento, mentre la funzione *require* verifica che determinate condizioni siano soddisfatte prima dell'esecuzione del codice. Se il controllo di *require* fallisce, l'esecuzione della funzione viene interrotta e la transazione viene annullata, garantendo così un ulteriore livello di sicurezza e prevenzione di accessi non autorizzati.

2.2.2 Distribution, Redundancy, Diversity

Nella protezione delle risorse informatiche, vengono utilizzate diverse tecniche avanzate, come la ridondanza, la diversità e la distribuzione.

La ridondanza implica la duplicazione di componenti critici o funzioni di un sistema per garantire la continuità del servizio in caso di guasto di uno dei suoi componenti. La diversità, invece, si riferisce all'uso di tecnologie, approcci o località diverse per ridurre le vulnerabilità comuni. Infine, la distribuzione riguarda l'allocazione e la gestione delle risorse informatiche e dei servizi software su più località geografiche o sistemi.

Nel contesto della progettazione software, la ridondanza gioca un ruolo fondamentale. Un esempio pratico potrebbe essere una rete di server che ospitano un'applicazione web. Ogni server contiene una copia dell'applicazione e gestisce le richieste provenienti dai client. Se uno dei server si guasta o subisce un attacco, gli altri server continuano a funzionare senza interruzioni, assicurando che il servizio non venga compromesso.

Inoltre, la scelta dell'architettura dipende dalla struttura della Blockchain. Questa tecnologia consente di creare un ampio registro distribuito per la gestione di transazioni condivise tra diversi nodi di una rete. Ogni nodo possiede una copia della Blockchain e svolge la stessa funzione: riceve una transazione, la verifica e, se corretta, la propone per l'aggiunta al registro.

Un aspetto distintivo della Blockchain è che i dati, una volta scritti in un blocco, non possono essere modificati retroattivamente senza alterare tutti i blocchi successivi. Questo processo richiede il consenso della maggioranza della rete, grazie al protocollo di validazione. Di conseguenza, le transazioni sulla Blockchain sono considerate imm modificabili.

La Blockchain, quindi, funge da registro immutabile che, per sua natura, implementa il concetto di ridondanza delle copie e distribuzione su più nodi, contribuendo alla protezione e affidabilità del sistema.

2.2.3 Analisi

La progettazione e implementazione delle tecniche di difesa e salvaguardia descritte in precedenza si è rivelata cruciale per l'analisi del sistema e la gestione del rischio, con l'obiettivo di valutare e migliorare la robustezza e l'affidabilità complessiva del sistema.

In particolare, le tecniche adottate hanno permesso di incrementare la sopravvivenza del sistema, assicurandone la capacità di continuare a operare anche in caso di guasti o attacchi esterni. L'applicazione, nel suo complesso, si è dimostrata robusta e affidabile, in grado di rispondere efficacemente a situazioni critiche.

Inoltre, è stata effettuata una valutazione delle capacità del sistema nel gestire dati ambigui, tramite un'analisi specifica di ambiguità. Ciò è stato verificato attraverso l'inserimento di input incompleti o anomali, e il sistema ha mostrato di essere in grado di prendere decisioni accurate anche in presenza di tali input, garantendo tolleranza agli errori.

Infine, l'implementazione di sistemi di mitigazione contro attacchi noti, come quelli descritti nei modelli STRIDE e Jacobson, ha consentito una valutazione di resistenza efficace contro questa tipologia di attacchi e intrusioni, rafforzando ulteriormente la sicurezza e la protezione del sistema.

2.2.4 Modellazione attraverso Markov

Nel processo di sviluppo di un software sicuro, è fondamentale poter analizzare il codice scritto per vedere se soddisfa delle proprietà necessarie per il

suo corretto funzionamento. Per farlo, è necessario come prima cosa rappresentare ogni modulo rilevante del software tramite un modello di Markov. Il modello può essere, a seconda dei casi, una catena di Markov a tempo discreto (DTMC) oppure un processo decisionale di Markov (MDP).

ASTRAZIONE DEL CODICE

In seguito, è stata effettuata la modellazione del processo di login dell'utente. L'algoritmo di partenza è suddiviso essenzialmente in due parti ed ha la seguente struttura:

```
def login_menu(self):
    print(Fore.YELLOW + "Type 'exit' at any prompt to cancel and go back.\n" + Style.RESET_ALL)
    while True:
        if not self.controller.check_attempts() and self.session.get_timeout_left() <= 0:
            self.session.reset_attempts()
        if self.session.get_timeout_left() <= 0 and self.controller.check_attempts():
            public_key = input("Insert public key: ")
            if public_key.lower() == 'exit': return -3
            private_key = getpass.getpass("Private Key: ")
            if private_key.lower() == 'exit': return -3
            username = input("Insert username: ")
            if username.lower() == 'exit': return -3
            passwd = getpass.getpass("Insert password: ")
            if passwd.lower() == 'exit': return -3
            login_code, user_role = self.controller.login(username, passwd, public_key, private_key)
            if login_code == 0:
                print(Fore.GREEN + "\nYou have successfully logged in!\n" + Style.RESET_ALL)
                self.user_menu(username, user_role)
            elif login_code == -1:
                print(Fore.RED + "\nThe credentials you entered are wrong!\n" + Style.RESET_ALL)
            elif login_code == -2:
                print(Fore.RED + "\nToo many login attempts!\n" + Style.RESET_ALL)
                return -1
            else:
                print(Fore.RED + "\nMax number of attempts reached!\n" + Style.RESET_ALL)
                print(Fore.RED + f'You will be in timeout for: {int(self.session.get_timeout_left())} s' + Style.RESET_ALL)
                return -2
```

Figure 2.1: Prima parte dell'algoritmo

```
login(self, username: str, password: str, public_key: str, private_key: str):
    if self.check_attempts() and self.db_ops.check_credentials(username, password, public_key, private_key):
        creds: Credentials = self.db_ops.get_creds_by_username(username)
        user_role = creds.get_role()
        user = self.db_ops.get_user_by_username(username)
        self.session.set_user(user)
        return 0, user_role
    elif self.check_attempts():
        self.session.increment_attempts()
        if self.session.get_attempts() == self.__n_attempts_limit:
            self.session.set_error_attempts_timeout(self.__timeout_timer)
            return -1, None
    else:
        return -2, None
```

Figure 2.2: Seconda parte dell'algoritmo

Dato che le credenziali sono inserite dall'utente a runtime, il codice presenta uno spazio di stato con cardinalità potenzialmente infinita. Come primo passo è stata effettuata l'astrazione di quest'ultimo seguendo l'enunciato del teorema di astrazione. Alla fine del processo è stato individuato l'insieme: $S = \{attempts, timeout_limit, public_key, private_key, username, password, login_status\}$

Per ciascun elemento dell'insieme è stato associato un significato specifico:

- ***attempts***: Intero che indica il numero di tentativi di accesso effettuati dall'utente. Può assumere i valori nell'intervallo $[0, 5]$.
- ***timeout_limit***: Valore booleano per indicare se l'utente è sotto timeout (true) oppure no (false).

- **public_key**: Contiene il valore della chiave pubblica del portafoglio associato all'account.

- **private_key**: Contiene la chiave privata del portafoglio.

- **username**: Contiene il nome utente associato all'account.

- **password**: Contiene la password dell'utente.

- **login_status**: Valore che indica come si è conclusa la procedura di login. Può assumere i seguenti valori:

- **login_code_null**: Valore non ancora inizializzato.

- **login_code_success**: Login effettuato con successo.

- **login_code_fail**: Tentativo di login fallito senza raggiungere il numero massimo di tentativi.

- **login_code_attempt_limit**: Tentativo di login fallito perché è stato raggiunto il numero massimo di tentativi.

Le singole credenziali possono assumere quattro possibili valori:

- **field_null**: Valore nullo e non ancora inizializzato.

- **field_ok**: Valore correttamente inserito.

- **field_wrong**: Valore inserito in modo errato.

- **field_exit**: L'utente ha inserito la parola riservata exit, necessaria per interrompere la procedura di login.

Con l'astrazione appena descritta, si passa da uno spazio di stato avente cardinalità infinita ad uno di dimensione finita ed enumerabile. Lo spazio finale è infatti composto da 12288 possibili stati, ma nella pratica solo una piccola parte di essi sono raggiungibili.

MODELLAZIONE CON PRISM

La realizzazione del modello è stata effettuata tramite il software **Prism**. Prism è pensato per realizzare facilmente questi tipi di modelli, per poi farne il debug e simularne il comportamento. Lo scopo ultimo del suo utilizzo è verificare se il modello (e quindi, di conseguenza, l'algoritmo di partenza) soddisfa determinate proprietà indicate dall'utente con conseguente calcolo delle probabilità di avvenimento.

L'immagine seguente mostra il modello implementato. Dato che tutte le componenti dello spazio di stato sono già inizializzate alla partenza, esso presenta un unico stato iniziale, ovvero la situazione in cui ci si trova quando si tenta di fare il login per la prima volta:

$$S_0 = \{0, false, field_null, field_null, field_null, field_null, login_code_null\}$$

L'inserimento delle credenziali è condizionato da delle probabilità. Per la chiave pubblica (il primo valore inserito) la probabilità di inserire il valore corretto è nettamente più alta rispetto agli altri possibili casi poiché

quasi sempre questo valore non viene inserito digitando sulla tastiera ma tramite copia-incolla da un punto in cui è già visualizzato (come, ad esempio, la schermata del proprio wallet). Sempre seguendo questa logica, è molto più facile accorgersi se è stato inserito un valore sbagliato e, di conseguenza, è molto più probabile che l'utente tenti di uscire direttamente dalla procedura senza continuare e vedere il proprio tentativo fallire.

Questa logica viene applicata anche a tutte le altre credenziali. All'inserimento corretto di un valore corrisponde una probabilità maggiore di inserire correttamente anche quello successivo. Al contrario, man mano che viene inserito un valore errato, la probabilità di arrivare a fine procedura diminuisce sempre di più.

```

1 // Imports
2 const int attempts_limit = 5;
3 const int login_code_success = 0;
4 const int login_code_fail = -1;
5 const int login_code_attempts_limit = -2;
6 const int login_code_timeout = -3;
7 const int field_ok = 0;
8 const int field_wrong = 1;
9 const int field_limit = 2;
10 const int field_miss = -1;
11
12 module login
13 attempts : [0..attempts_limit] limit 0;
14 timeout_limit : bool limit false;
15 login_status : (login_code_miss, login_code_success) limit login_code_miss;
16 public_key : (field_miss, field_miss) limit field_miss;
17 private_key : (field_miss, field_miss) limit field_miss;
18 username : (field_miss, field_miss) limit field_miss;
19 password : (field_miss, field_miss) limit field_miss;
20
21 // The user enters his public key
22 [] public_key_field_ok & private_key_field_miss & username_field_miss & password_field_miss & timeout_limit & attempts_attempts_limit ->
23   0.8 (public_key_field_ok | login_status = login_code_success) + 0.1 (public_key_field_wrong | login_status = login_code_miss)
24   + 0.1 (public_key_field_miss | login_status = login_code_miss);
25 [] public_key_field_miss & private_key_field_ok & username_field_miss & password_field_miss & timeout_limit & attempts_attempts_limit ->
26   0.8 (public_key_field_miss | login_status = login_code_miss) + 0.1 (public_key_field_wrong | login_status = login_code_miss) + 0.1 (public_key_field_miss | login_status = login_code_attempts_limit);
27 + 0.1 (public_key_field_miss | login_status = login_code_miss) & (attempts = 0);
28
29 // The user enters his private key
30 [] public_key_field_ok & private_key_field_ok & username_field_miss & password_field_miss -> 0.8 (private_key_field_ok)
31 + 0.1 (private_key_field_wrong) + 0.1 (private_key_field_miss);
32 [] public_key_field_wrong & private_key_field_ok & username_field_miss & password_field_miss -> 0.1 (private_key_field_ok)
33 + 0.1 (private_key_field_wrong) + 0.8 (private_key_field_miss);
34 [] public_key_field_miss -> (public_key_field_miss | private_key_field_miss | (password_field_miss | (username_field_miss)));
35
36 // The user enters his username
37 [] public_key_field_ok & private_key_field_ok & username_field_miss & password_field_miss -> 0.7 (username_field_ok)
38 + 0.2 (username_field_wrong) + 0.1 (username_field_miss);
39 [] public_key_field_wrong | private_key_field_wrong & username_field_miss & password_field_miss -> 0.2 (username_field_ok)
40 + 0.1 (username_field_wrong) + 0.7 (username_field_miss);
41 [] private_key_field_miss -> (public_key_field_miss | private_key_field_miss | (password_field_miss | (username_field_miss)));
42
43 // The user enters his password
44 [] public_key_field_ok & private_key_field_ok & username_field_ok & password_field_miss -> 0.7 (password_field_ok)
45 + 0.2 (password_field_wrong) + 0.1 (password_field_miss);
46 [] public_key_field_wrong | private_key_field_wrong | username_field_ok & password_field_miss -> 0.2 (password_field_ok)
47 + 0.1 (password_field_wrong) + 0.7 (password_field_miss);
48 [] username_field_miss | password_field_miss -> (public_key_field_miss | private_key_field_miss | (password_field_miss | (username_field_miss)));
49
50 [] public_key_field_ok & private_key_field_wrong | username_field_ok & password_field_wrong & attempts_attempts_limit ->
51 (login_status = login_code_attempts_limit) & (timeout_limit = true) & (public_key_field_miss | private_key_field_miss);
52 (username_field_miss | password_field_miss) // login_fail_attempts_limit
53
54 [] public_key_field_ok & private_key_field_wrong | username_field_miss & password_field_wrong & attempts_attempts_limit ->
55 (attempts_attempts_limit) & (login_status = login_code_timeout) & (public_key_field_miss | private_key_field_miss);
56 (username_field_miss | password_field_miss) // login_fail_no_attempts_limit
57
58 // login success
59 [] public_key_field_ok & private_key_field_ok & username_field_ok & password_field_ok & attempts_attempts_limit -> (login_status = login_code_success);
60
61 [] attempts_attempts_limit -> (timeout_limit = true) & (login_status = login_code_fail) & (public_key_field_miss | private_key_field_miss);
62 (username_field_miss | password_field_miss);
63 endmodule

```

Figure 2.3: Modello Prism

Il modello raggiunge uno stato finale se:

- L'utente riesce ad accedere correttamente entro il numero di tentativi consentiti.
- Viene raggiunto il numero massimo di tentativi utilizzabili e viene dato un timeout all'utente. Per tutta la durata del timeout all'utente verrà impedito l'accesso, anche in caso di credenziali corrette.

Per indicare convenientemente questi stati, oltre che altre situazioni particolari, sono state usate delle **etichette**:

```

65 // Labels
66 label "login_success" = (login_status=login_code_success);
67 label "login_fail_no_attempts_limit" = (timeout_limit & (login_status=login_code_attempts_limit));
68 label "login_fail_no_attempts_limit" = (login_status=login_code_fail);
69 label "final_state" = (login_status=login_code_success) & ((timeout_limit & (login_status!=login_code_success)));

```

Figure 2.4: Etichette modello Prism

- **login_success**: Indica lo stato in cui il login è terminato con successo.

- **login_fail_attempts_limit**: Stato in cui il tentativo di login è fallito a causa del raggiungimento del numero di tentativi.
- **login_fail_no_attempts_limit**: Stato in cui il tentativo di accesso è fallito.
- **final_state**: Indica uno stato finale del modello. È una generalizzazione delle tre etichette precedenti.

VERIFICA DELLE PROPRIETÀ

Le proprietà del sistema sono indicate tramite la classificazione **Safety-Progress**. In particolare, le proprietà del modello da verificare sono:

- L'algoritmo termina in modo sicuro (Safety – una cosa buona accade sempre): Si traduce nel fatto che il modello riesce ad arrivare sempre in uno degli stati finali possibili. È possibile indicarla in due modi diversi:
 - $P \geq 1[F(\backslash final_state)]$
 - $P \geq 1[F("login_success") | ("login_fail_no_attempts_limit") | ("login_fail_attempts_limit")]$
- L'utente riesce almeno una volta ad accedere con successo (Guarantee – una cosa buona accade almeno una volta): $Pmin = ?[F(\backslash login_success)]$

L'analisi della prima proprietà ha confermato, per entrambe le varianti, la sua validità. Il valore ottenuto è infatti pari a 1:

```
Model checking: P>=1 [ F ("login_success") | ("login_fail_no_attempts_limit") | ("login_fail_attempts_limit") ]
Building model...
Computing reachable states...
Reachability (BFS): 14 iterations in 0.00 seconds (average 0.000000, setup 0.00)
Time for model construction: 0.02 seconds.
Type: MDP
States: 481 (1 initial)
Transitions: 1197
Choices: 877
Transition matrix: 440 nodes (6 terminal), 1197 minterms, vars: 14r/14c/3nd
Probability bound in formula is 0/1 so not computing exact probabilities...
Prob0E: 9 iterations in 0.00 seconds (average 0.000000, setup 0.00)
Prob1A: 1 iterations in 0.00 seconds (average 0.000000, setup 0.00)
yes = 481, no = 0, maybe = 0
Property satisfied in 1 of 1 initial states.
Time for model checking: 0.01 seconds.
Result: true
```

Figure 2.5: Prima proprietà

```
Model checking: P>=1 [ F "final_state" ]
Probability bound in formula is 0/1 so not computing exact probabilities...
Prob0E: 10 iterations in 0.00 seconds (average 0.000000, setup 0.00)
Prob1A: 1 iterations in 0.00 seconds (average 0.000000, setup 0.00)
yes = 481, no = 0, maybe = 0
Property satisfied in 1 of 1 initial states.
Time for model checking: 0.0 seconds.
Result: true
```

Figure 2.6: Prima proprietà - variante B

L'analisi della seconda, invece, mostra come la probabilità minima che si verifichi è 94,5% dopo che sono accadute circa 10 iterazioni nel modello:


```

Model checking: Pain=? { F "login_success" }
ProbeE: 10 iterations in 0.00 seconds (average 0.000000, setup 0.00)
ProbeIA: 11 iterations in 0.00 seconds (average 0.000000, setup 0.00)
yes = 10, no = 87, maybe = 384
Computing remaining probabilities...
Engine: Hybrid

Building hybrid MTBD matrices... [nm=4, levels=14, nodes=630] [29.5 KB]
Adding sparse bits... [levels=14-14, num=4, compact=4/4] [5.8 KB]
Creating vector for yes... [dist=2, compact] [1.0 KB]
Allocating iteration vectors... [3 x 3.8 KB]
TOTAL: [47.5 KB]

Starting iterations...
Iterative method: 67 iterations in 0.00 seconds (average 0.000000, setup 0.00)
Value in the initial state: 0.9453878352996681
Time for model checking: 0.003 seconds.
Result: 0.9453878352996681 (+/- 9.409547818673861E-6 estimated; rel err 9.953108626250974E-6)

```

Figure 2.7: Verifica seconda proprietà

2.3 Scelta delle tecnologie e analisi delle debolezze

2.3.1 Tecnologie usate nello sviluppo

- Linguaggio di Programmazione Principale: **Python**

Python è un linguaggio di programmazione versatile e molto utilizzato, noto per la sua sintassi semplice e l'ampia disponibilità di moduli e librerie. La sua popolarità lo rende una scelta ideale per lo sviluppo di software complessi. La scelta di utilizzare Python è stata motivata anche dall'esperienza comprovata di tutti i membri del team con questo linguaggio, che facilita lo sviluppo rapido e l'adozione di best practices nel progetto.

- Simulatore Blockchain: **Ganache**

Ganache è un simulatore di blockchain Ethereum che consente lo sviluppo e il testing di applicazioni decentralizzate in un ambiente locale. La sua adozione è stata suggerita per la possibilità di testare e sviluppare smart contract senza necessità di interagire con la rete principale di Ethereum. Ganache offre funzionalità avanzate come la generazione di blocchi, la gestione dei conti e una console per il debug delle transazioni, integrandosi perfettamente con Visual Studio Code per un flusso di lavoro efficiente.

- Database: **SQLite3**

SQLite3 è un database SQL compatto e indipendente, ideale per applicazioni embedded e distribuite. Grazie alla sua semplicità e all'efficienza operativa, rappresenta una soluzione efficace per la gestione dei dati in questo software. Le ottime prestazioni nelle operazioni di lettura e scrittura hanno contribuito a rendere più rapido ed efficace il testing in tempo reale dell'applicazione.

- Sviluppo di Smart Contract: **Solidity**

Solidity è un linguaggio di programmazione specificamente progettato per la scrittura di smart contract sulla blockchain Ethereum. Consente la creazione di contratti autonomi che eseguono automaticamente transazioni sulla blockchain, in base a condizioni predefinite. L'uso di Solidity è fondamentale per automatizzare e decentralizzare

la gestione dei dati medici privati all'interno del software, garantendo sicurezza e trasparenza nelle operazioni.

- **IDE: Visual Studio**

L'utilizzo di Visual Studio come ambiente di sviluppo integrato (IDE) offre numerosi vantaggi, tra cui la possibilità di sviluppare applicazioni Python con funzionalità complete per scrittura, debug, testing e deployment del codice. Inoltre, la sua integrazione nativa con Git facilita la gestione del codice sorgente, il monitoraggio delle modifiche e la collaborazione tra i membri del team. L'estensione integrata per Ganache potenzia ulteriormente il flusso di lavoro, semplificando l'interazione con la blockchain durante lo sviluppo.

- **Controllo Distribuito del Codice: Git e GitHub**

L'adozione di Git e GitHub come strumenti per il controllo distribuito del codice assicura una collaborazione efficiente tra i membri del team. Git consente di lavorare simultaneamente su diverse parti del codice, mantenendo una cronologia completa delle modifiche e facilitando la risoluzione dei conflitti. GitHub funge da repository centralizzato per il codice, garantendo un accesso sicuro e controllato, mentre consente anche di ripristinare versioni precedenti in caso di necessità.

- **Prove smart contract: Remix**

Remix è un framework per lo sviluppo di interfacce utente web che si integra perfettamente con il software, permettendo la creazione di interfacce utente dinamiche e reattive. Questo strumento garantisce un'esperienza utente fluida e intuitiva, mentre l'approccio basato sui componenti facilita la collaborazione tra i membri del team, consentendo una suddivisione del lavoro più efficace e modulare.

- **Test Blockchain Privata: Besu**

Besu è un client Ethereum open source progettato per ambienti enterprise, che si integra perfettamente con reti blockchain sia pubbliche che private. Grazie alla sua architettura modulare e alla compatibilità con vari algoritmi di consenso, Besu offre una base solida per la creazione di applicazioni distribuite scalabili e sicure. Questo strumento garantisce trasparenza e tracciabilità delle transazioni, mentre

il supporto per gli smart contract e l'interoperabilità con altri strumenti della blockchain enterprise facilita la collaborazione tra team di sviluppo, promuovendo una gestione più efficiente e controllata delle infrastrutture decentralizzate.

2.3.2 Analisi delle debolezze - SWOT

Un'analisi delle debolezze sotto vari punti di vista è fondamentale per un miglioramento costante del software, del team e del processo di sviluppo. Questa consente di individuare i punti critici del software e del processo di sviluppo che potrebbero causare problemi o limitare le prestazioni. Tale consapevolezza permette al team di concentrare gli sforzi di miglioramento sui settori che ne hanno più bisogno. In seguito, le debolezze identificate possono essere convertite in obiettivi di miglioramento per aumentare la qualità complessiva del software. Attraverso l'analisi delle debolezze, il team può implementare correzioni, ottimizzazioni e nuove strategie per affrontare i problemi e ridurre i rischi. Infatti, conoscere le debolezze del software e del processo di sviluppo consente di ridurre i rischi associati a errori, vulnerabilità e problemi di prestazioni. Affrontare le debolezze in modo proattivo aiuta a prevenire potenziali incidenti e a proteggere l'integrità del sistema, oltre che massimizzarne l'efficienza operativa e della produttività del team. L'analisi SWOT (Strengths, Weaknesses, Opportunities, Threats) consente di trasformare le debolezze in opportunità identificando punti di forza che possono essere sfruttati per affrontare le sfide. Trasformare le debolezze in opportunità permette al team di sviluppo di capitalizzare sui propri punti di forza, affrontare i problemi con efficacia e perseguire obiettivi di miglioramento strategici che portano a un progresso significativo nel tempo. Pertanto, l'analisi critica del sistema SFSCchain ha seguito tale linea guida; ci si è concentrati sui seguenti punti focali:

1. Complessità di una BlockChain Ethereum:

Debolezze: La gestione delle transazioni, dei contratti intelligenti e delle interazioni con la rete Ethereum può essere complicata e richiedere una curva di apprendimento significativa.

Opportunità: A monte dello sviluppo del software, c'è stata un'importante formazione universitaria del team, attraverso le lezioni del corso di Software Security e Blockchain dell'UNIVPM, al fine di acquisire competenze avanzate nella gestione della blockchain Ethereum.

2. **Rischi di sicurezza:**

Debolezze: La sicurezza dei dati sensibili è una preoccupazione critica e richiede una progettazione e un'implementazione rigorose.

Opportunità: Implementare meccanismi avanzati di sicurezza per proteggere i dati non autorizzati e da violazioni della privacy.

3. **Dipendenza da librerie di terze parti:**

Debolezze: Dipendenza da diverse librerie di terze parti che potrebbero essere deprecate o smettere di essere mantenute.

Opportunità: Monitorare costantemente lo sviluppo delle librerie di terze parti utilizzate e valutare alternative o soluzioni personalizzate in caso di cambiamenti significativi o discontinuità delle librerie.

4. **Curva di apprendimento Solidity:**

Debolezze: Lo sviluppo di contratti intelligenti in Solidity richiede una conoscenza approfondita e profonda.

Opportunità: Il team, a seguito delle lezioni tenute e dalle esercitazioni, ha acquisito competenze in Solidity e nella programmazione sicura dei contratti.

Chapter 3

Implementazione

La fase di implementazione rappresenta un passaggio cruciale nello sviluppo di un software. È importante tenere presente che possono emergere errori durante l'implementazione che rischiano di compromettere il risultato finale. Tali errori non si limitano a semplici differenze tra il codice scritto e il progetto originario. Infatti, anche quando il codice rispetta apparentemente quanto previsto dal design, le scelte tecniche effettuate in fase di sviluppo possono introdurre vulnerabilità. Queste fragilità sono spesso connesse alle caratteristiche specifiche del linguaggio di programmazione adottato e al modo in cui esso gestisce le operazioni a livello semantico.

3.1 Programmazione sicura

Al fine di mitigare i rischi è possibile fare riferimento a diverse linee guida che promuovono pratiche di programmazione sicura ed efficace. In particolare, in questo progetto sono state seguite le seguenti:

- **Limitare la visibilità delle informazioni:** questa linea guida riprende il principio del *minimo privilegio*, secondo il quale ogni componente deve poter accedere solo ai dati strettamente necessari per svolgere la sua funzione. Limitando l'accesso, si evita che un modulo possa inavvertitamente modificare lo stato interno di un altro modulo, prevenendo bug e vulnerabilità.
- **Verificare la validità di tutti gli input:** questo evita che un programma possa assumere comportamenti imprevisti che possono rappresentare potenziali minacce per la sicurezza del software.
- **Gestione delle eccezioni:** è stata implementata attraverso l'utilizzo del costrutto *try - except* e la stampa dei *log* di errore.

- **Fornire funzionalità di riavvio:** ciò permette di evitare fallimenti silenziosi delle componenti esterne richiamate dal sistema. Ad esempio, questa funzionalità è stata garantita in fase di connessione alla blockchain *Hyperledger BESU*: in caso di fallimento di un primo tentativo di connessione ad uno dei nodi disponibili, vi è la possibilità da parte dell'utente di ritentare la connessione.

3.2 Standard di codifica

Il linguaggio utilizzato per lo sviluppo dello smart contract è Solidity, il più diffuso per le blockchain compatibili con Ethereum. Per aumentarne la sicurezza, si è fatto riferimento allo standard di codifica previsto per l'ottenimento della certificazione *EEA EthTrust Certification*. Questo standard definisce un insieme di requisiti minimi per la conduzione di audit su smart contract, con l'obiettivo di stabilire criteri qualitativi uniformi nel campo della sicurezza. Gli audit conformi a EthTrust possono ottenere una certificazione suddivisa in tre livelli di sicurezza, ognuno con requisiti specifici e crescenti in termini di garanzie.

Il codice sviluppato rispetta i requisiti essenziali del *livello M* (secondo dei tre livelli), tra cui:

- Gestione rigorosa dei controlli di accesso tramite ruoli chiaramente definiti
- Validazione degli input per prevenire errori e accessi non autorizzati
- Uso di librerie standard e affidabili (OpenZeppelin) per l'implementazione dei token ERC20
- Tracciamento puntuale delle operazioni attraverso eventi

Tali caratteristiche assicurano una base solida di sicurezza e conformità ai requisiti minimi richiesti da EthTrust.

3.3 Database

Nel progetto è stato utilizzato *SQLite3* come database locale per la gestione degli utenti. In particolare, il database è costituito da tre tabelle, due delle quali fanno riferimento alle informazioni degli utenti (tabelle *Users* e *Credentials*), e una utile al salvataggio dei report delle operazioni effettuate (tabella *Reports*). Nella tabella *Users* troviamo le informazioni anagrafiche

degli utenti e il loro ruolo, mentre nella tabella Credentials vi sono le credenziali, tra cui password, chiave pubblica e chiave privata.

La password dell'utente viene protetta tramite l'algoritmo *scrypt*, una funzione di derivazione crittografica progettata per essere resistente agli attacchi a forza bruta grazie all'elevato utilizzo di memoria e CPU. Il risultato dell'operazione (*digest*) è un hash univoco ottenuto combinando la password con un *salt* casuale e parametri di sicurezza ben definiti. Per verificarne la correttezza in fase di login, si applica nuovamente la stessa funzione *scrypt* alla password inserita, usando i parametri e il salt originari: se l'hash ottenuto coincide con quello salvato, l'accesso viene consentito.

Per quanto riguarda la chiave privata, è stata adottata una protezione tramite crittografia simmetrica AES, dove la password dell'utente (opportunamente trasformata in chiave tramite *scrypt*) viene usata per cifrarla. In questo modo, solo chi conosce la password può decifrare la chiave privata, proteggendola da accessi non autorizzati.

3.4 Autenticazione

Per accedere alla piattaforma l'utente deve inserire password, chiave pubblica e chiave privata. L'inserimento di queste ultime due oltre alla password si configura come un sistema di autenticazione a due fattori (2FA), il che rende il processo più sicuro. L'autenticazione a due fattori (2FA) è considerata più sicura rispetto alla sola password perché combina due categorie distinte di prove di identità, rendendo molto più difficile per un attaccante impersonare l'utente.

Ogni utente ha la possibilità di cambiare la propria password attraverso la funzione *Change password* presente nel menu; ciò è possibile solamente previo l'inserimento della vecchia password dell'utente, in modo da garantire l'esecuzione legittima della funzione.

Per quanto riguarda le operazioni di scrittura sulla blockchain, viene richiesto all'utente l'inserimento della chiave privata: questo è necessario per la firma delle transazioni e viene richiesta solo quando necessario per limitare il tempo in cui essa rimane attiva in memoria, al fine di limitarne l'esposizione e prevenire potenziali leak.

3.5 On-chain

Ogni attore della filiera agroalimentare è soggetto a un meccanismo di ricompensa o penalizzazione basato sulle emissioni di Co2 generate durante

le proprie operazioni. L'adozione della blockchain apporta i seguenti benefici specifici:

- **Tracciabilità immutabile delle emissioni:** ogni operazione che produce emissioni viene registrata su blockchain tramite smart contract. Questo garantisce che i dati relativi alla Co2 siano verificabili, non modificabili e disponibili per audit futuri.
- **Attribuzione trasparente di crediti e debiti:** gli smart contract automatizzano la distribuzione di ricompense a chi adotta pratiche sostenibili, e l'applicazione di costi a chi eccede determinati limiti. Questo elimina possibili manipolazioni o favoritismi da parte di terzi.
- **Responsabilità individuale e firma digitale:** ogni transazione è firmata digitalmente e associata a un singolo attore, che ne risulta quindi responsabile.
- **Sistema equo e verificabile:** tutti i partecipanti operano secondo regole condivise e automatiche, verificate dalla blockchain. I dati sono accessibili in lettura, ma protetti da modifiche non autorizzate, offrendo un sistema affidabile e auditabile.

Inoltre, la blockchain offre vantaggi significativi in termini di robustezza. Essendo una rete distribuita su molteplici nodi indipendenti, garantisce la continuità operativa anche in presenza di guasti o compromissioni di singoli nodi. Di conseguenza, il sistema evita interruzioni o perdite di dati dovute a problemi hardware o attacchi localizzati, assicurando l'accessibilità e l'integrità delle informazioni. Nel progetto è stata utilizzata Hyperledger Besu, un client Ethereum conforme agli standard enterprise, con una rete privata composta da quattro nodi e configurata per adottare l'algoritmo di consenso *QBFT* (*Quorum Byzantine Fault Tolerant*). Questa scelta assicura tolleranza ai guasti bizantini e permette di mantenere l'integrità del consenso anche in presenza di nodi malfunzionanti o malevoli, purché il numero di nodi onesti superi i due terzi del totale. Di conseguenza, il sistema evita interruzioni o perdite di dati dovute a problemi hardware o attacchi localizzati, assicurando l'accessibilità, la disponibilità e la resilienza delle informazioni on-chain. La registrazione degli utenti è gestita esclusivamente dall'account amministratore, che è stato preconfigurato al momento dell'inizializzazione della rete Besu. Questa scelta è motivata dal fatto che un nuovo utente, al momento della registrazione, non dispone di ETH, la valuta nativa necessaria per eseguire transazioni sulla blockchain.

(inclusa la chiamata al metodo di registrazione). Pertanto, non è in grado di interagire autonomamente con gli smart contract, e ha bisogno che sia l'admin a effettuare la registrazione iniziale.

3.6 Action flow dell'utente

L'utente può interagire con il programma tramite una CLI (Command Line Interface), ovvero un'interfaccia a riga di comando che permette di interagire tramite dei comandi testuali. All'avvio del programma, non essendo ancora stato compilato e caricato nessuno smart contract, verrà tentato il deploy del contratto sulla blockchain. In caso di esito positivo, verrà mostrato all'utente il menu principale, altrimenti si dà all'utente la possibilità di ritentare il deploy o uscire dal programma.

3.6.1 Registrazione

La prima schermata (Figura 3.1) presenta all'utente tre opzioni tra cui scegliere (inserendo il relativo numero): registrazione, login o uscita dal programma.

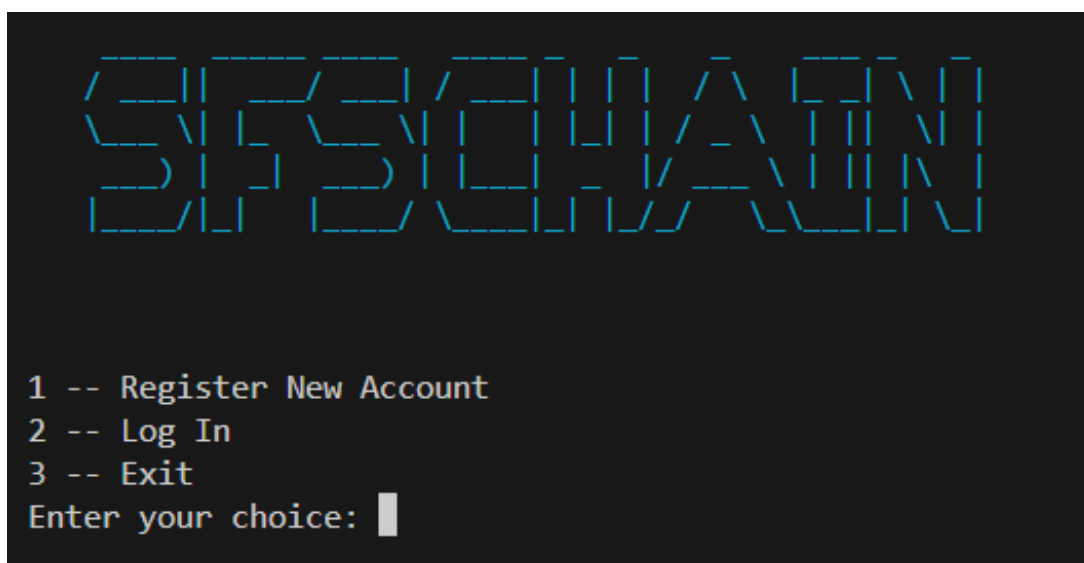


Figure 3.1: Schermata iniziale

Scegliendo **Registration**, all'utente verrà richiesto di inserire la propria chiave pubblica e chiave privata del wallet. L'inserimento è protetto tramite la funzione *getpass*, che impedisce la visualizzazione del contenuto digitato a schermo. Sono inoltre previsti controlli sia sul formato delle chiavi, sia sulla validità della coppia chiave pubblica–privata, che deve:

- Rispettare i criteri di struttura previsti
- Rappresentare una coppia corretta
- Non essere già stata utilizzata in precedenza

Successivamente, verranno richiesti username, password e il ruolo dell'utente, da selezionare tra una delle seguenti opzioni: *Farmer*, *Producer*, *Carrier* o *Seller*.

```
Please, enter your wallet credentials.
Public Key: 0x8630Aef947346b2EeC708ce80987E7f98044b755
Private Key:
Confirm Private Key:
Enter your personal information.
Username: andrea_giuliani
Insert your role:
(F) Farmer
(P) Producer
(C) Carrier
(S) Seller
Your choice: 
```

Figure 3.2: Registrazione

La password inserita deve rispettare un formato specifico, verificato tramite espressione regolare (*regex*). Nel caso in cui la password non soddisfi i requisiti richiesti, oppure la conferma della password non corrisponda a quella inizialmente inserita, all'utente verrà chiesto di ripetere l'inserimento. Successivamente, verranno richieste le informazioni personali per completare il processo di registrazione. In qualsiasi momento è possibile interrompere il processo di registrazione digitando 'exit' in uno qualunque dei campi richiesti. Per evitare inconsistenze, le informazioni vengono salvate sia nel database locale che sulla blockchain solo al termine dell'inserimento completo di tutti i dati richiesti. Terminato il processo di registrazione, si accederà al menu utente.

3.6.2 Login

Se si è già registrati, è possibile effettuare il login (seconda opzione nella schermata iniziale) inserendo chiave pubblica, chiave privata, username e password (Figura 3.3). In caso di credenziali errate, viene incrementato il contatore dei tentativi: al superamento del quinto tentativo, il programma

torna automaticamente al menu principale e il login è bloccato per 180 secondi. Questo meccanismo rappresenta una misura di protezione contro gli attacchi a forza bruta, limitando il numero massimo di tentativi consentiti.

```
Insert public key: 0x8630Aef947346b2EeC708ce80987E7f98044b755
Private Key:
Insert username: andrea_giuliani
Insert password:
You have successfully logged in!
```

Figure 3.3: Login

Una volta effettuato il login, si accede al menu utente che presenta diverse opzioni, le quali a loro volta danno accesso a diversi sotto-menu:

- *Profile*: Permette di visualizzare le informazioni dell'utente, aggiornarle e cambiare la password.
- *Operation*: Permette di accedere al sotto-menu relativo alle operazioni, tra cui controllo del saldo, cessione crediti, registrazione operazioni o green action.
- *Report*: Permette di generare nuovi report o visualizzare quelli già esistenti e salvati nel DB locale.
- *Ask for credit*: Mostra una lista degli utenti registrati alla piattaforma con relative informazioni di contatto; l'utente può quindi contattare direttamente utilizzando tali informazioni per chiedere la cessione di uno o più crediti.
- *Logout*: Permette di effettuare il logout dalla piattaforma.

```
MENU
1 -- Profile
2 -- Operation
3 -- Report
4 -- Ask for credit
5 -- Log out
Choose an option: 
```

Figure 3.4: Menu utente

3.6.3 Menu Profile

Il menu **Profile**, come accennato in precedenza, dà all'utente la possibilità di visualizzare le informazioni anagrafiche, aggiornarle e cambiare la password (Figura 3.5).

```
PROFILE MENU (press Enter to go back)
1 -- View Profile
2 -- Update Profile
3 -- Change Password
Choose an option: █
```

Figure 3.5: Menu profilo

Le scelte possibili sono:

1. *View Profile*: effettuando questa scelta, saranno stampati a schermo: Username, Name, Last Name, Birthdate, Company Name, Role e Phone (Figura 3.6).

```
User INFO

Username: andrea_giuliani
Name: Andrea
Last Name: Giuliani
Birth Date: 1998-09-08
Company Name: Univpm
Role: FARMER
Phone: 3278547856
```

Figure 3.6: Informazioni utente

2. *Update profile*: selezionando questa opzione, l'utente può modificare i seguenti dati: Nome, Cognome, Data di nascita e Numero di telefono. Per confermare le modifiche, è richiesta la chiave privata, necessaria per registrare le informazioni aggiornate sulla blockchain. I campi lasciati vuoti manterranno i valori attuali.

```
Update profile function
Name [Andrea]: Andrea
Lastname [Giuliani]: Giuliani
Date of birth (YYYY-MM-DD) [1998-09-08]: 1999-09-08
Phone [3278547856]: 3562145785
Insert private key to confirm the transaction:
Profile updated successfully!
```

Figure 3.7: Aggiornamento profilo

3. *Change Password*: attraverso questa opzione l'utente ha la possibilità di cambiare la sua password, andando ad inserire prima quella attuale e successivamente due volte quella desiderata, la quale dovrà sempre rispettare i criteri citati precedentemente (Figura 3.8).

```
PROFILE MENU (press Enter to go back)
1 -- View Profile
2 -- Update Profile
3 -- Change Password
Choose an option: 3
Do you want to change your password (Y/n): Y
Old Password: To5sODRnPr+J2IFG
New password:
Confirm new password:

Password changed correctly!
```

Figure 3.8: Cambio password

3.6.4 Menu Operation

Il menu **Operation** dà all'utente la possibilità di controllare il proprio saldo, cedere crediti, registrare operazioni o green action (Figura 3.9).

```
OPERATION MENU (press Enter to go back)
1 -- Check Balance
2 -- Give Credit
3 -- Make Operation
4 -- Make Green Action
Choose an option: 
```

Figure 3.9: Menu operazioni

Le scelte possibili sono:

1. *Check Balance*: effettuando questa scelta è possibile controllare il proprio saldo (salvato sulla blockchain) che verrà stampato su schermo.
2. *Give Credit*: selezionando questa opzione è possibile cedere uno o più crediti (se il saldo è sufficiente) ad un altro utente. Verrà chiesto l'ammontare di crediti che si desidera cedere, lo username dell'utente destinatario e una conferma dell'intenzione di procedere con l'operazione. Per completare la transazione è richiesta la chiave privata e viene stampato l'indirizzo pubblico dell'utente al quale saranno ceduti i crediti (Figura 3.10).

```
How many credits do you want to give? 1
Insert the username of the account you want to give credit (type 'exit' to go back):
marco
Do you want to proceed with the operation? (Y/n): Y
You will give credits to the address: 0xd3237b2285F4c062304d3135F741D00d8cE4AD1e
Insert private key to confirm the transaction:
Your credits have been successfully given to: marco
```

Figure 3.10: Cessione crediti

3. *Make Operation*: selezionando questa opzione è possibile registrare un'operazione. Ogni attore dispone di un set predefinito di operazioni tra cui scegliere, in conformità con il principio del *minimo privilegio*, secondo cui ciascun attore può eseguire solo le operazioni strettamente legate al proprio ruolo. Accanto a ciascuna operazione sono indicati i valori di riferimento della CO₂ emessa, che rappresentano la soglia oltre la quale verranno scalati dei crediti, mentre al di sotto della quale si otterranno crediti. In particolare, nell'esempio riportato in Figura 3.11, è mostrato il set di operazioni disponibili per un *Farmer*. Come si può notare, non sono stati né guadagnati né scalati crediti, poiché la quantità di CO₂ emessa è pari alla soglia di riferimento.

```
Available FARMER Operations:
1. Soil Preparation (reference: 12 tons CO2 per hectare)
2. Sowing (reference: 3 tons CO2 per hectare)
3. Fertilization (reference: 7 tons CO2 per hectare)
4. Irrigation (reference: 5 tons CO2 per hectare)
5. Harvesting (reference: 8 tons CO2 per hectare)

Select the operation (1-5): 1
Enter number of hectares (or units) for 'Soil Preparation': 1
Insert actual CO2 emission for 'Soil Preparation' (in tons): 12
Insert private key to confirm the transaction:
The operation has been correctly recorded. No credit variation: emission equals threshold.
```

Figure 3.11: Registrazione operazione

4. *Make Green Action*: effettuando questa scelta l'utente ha la possibilità di registrare una green action. In questo caso non sono presenti delle azioni predefinite, ma viene richiesto all'utente di descrivere l'azione da registrare e qual è l'ammontare di CO₂ che è possibile risparmiare. Per ogni tonnellata di CO₂ dichiarata come risparmio, l'utente otterrà un credito. Come mostrato in Figura 3.12, un esempio di azione green per il *Farmer* potrebbe essere quella di utilizzare fertilizzanti organici al posto di quelli chimici.

```
Make a Green Action
Enter the description of the green action you performed: Utilizzo di fertilizzanti organici
Enter the amount of CO2 saved (in tons): 1
Insert private key to confirm the transaction:
Green action registered. 1 tons of CO2 saved credited to your wallet.
```

Figure 3.12: Registrazione green action

3.6.5 Menu Report

Il menu **Report** permette di generare e visualizzare dei report che raccolgono tutte le operazioni effettuate in un arco temporale definito dall'utente (Figura 3.13).

```
REPORT MENU (press Enter to go back)
1 -- Generate New Report
2 -- View Report
Choose an option: 
```

Figure 3.13: Menu Report

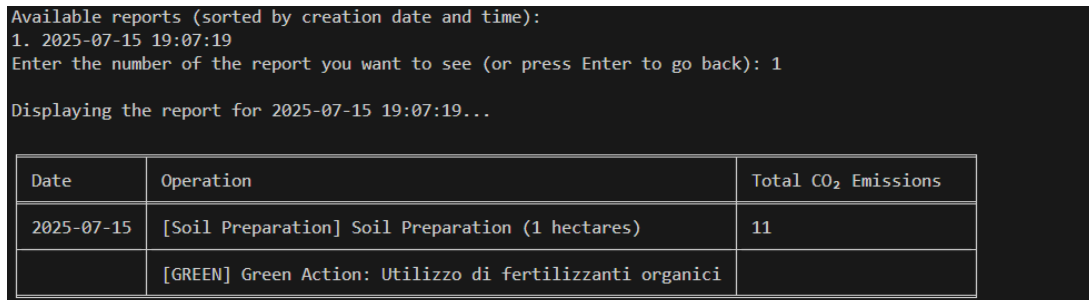
Le possibili scelte sono:

1. *Generate New Report*: questa opzione consente all'utente di generare un nuovo report, che verrà salvato nel database locale. All'utente verrà richiesto di specificare la data di inizio e di fine del periodo di riferimento per le operazioni da includere nel report. Sono presenti controlli sulle date che impediscono di inserire date future o una data di fine periodo antecedente quella di inizio periodo.

```
Insert the first day of the operation you want to insert in the report (YYYY-MM-DD): 2025-07-15
Insert the date of the last operation you want to put into the report (YYYY-MM-DD): 2025-07-15
Your report is ready
```

Figure 3.14: Generazione report

2. *View Report*: i report generati tramite la funzione precedentemente menzionata possono essere visualizzati scegliendo questa opzione. Vengono quindi mostrati tutti i report salvati nel DB, ordinati per data e ora di creazione. Ad ogni report è associato un numero che permette di scegliere quello desiderato. Una volta scelto il report da visualizzare, vengono mostrate le date delle operazioni, la loro descrizione e il bilancio della CO₂ emessa. Per bilancio si intende la differenza tra la CO₂ emessa e quella "risparmiata" attraverso le azioni green. Nell'esempio in Figura 3.15, la prima operazione corrispondeva ad un'emissione pari a 12 tonnellate di CO₂, mentre l'azione green permetteva di risparmiare 1 tonnellata, per cui viene mostrata la differenza pari a 11 tonnellate.



Date	Operation	Total CO ₂ Emissions
2025-07-15	[Soil Preparation] Soil Preparation (1 hectares)	11
	[GREEN] Green Action: Utilizzo di fertilizzanti organici	

Figure 3.15: Visualizzazione report

3.7 Analisi statica

Per garantire la qualità, sicurezza e correttezza del codice dello smart contract, è fondamentale utilizzare strumenti di analisi statica che esaminano il codice senza eseguirlo, evidenziando potenziali errori, vulnerabilità e problemi stilistici. In particolare, sono stati presi in considerazione *Solhint* e *Remix Solidity Analyzer*. *Solhint* è un linter open-source specificamente progettato per il linguaggio Solidity. Fornisce un insieme di regole configurabili per individuare errori di sintassi, violazioni delle best practice, problemi di sicurezza e di stile, aiutando gli sviluppatori a scrivere codice più sicuro e manutenibile. *Remix Solidity Analyzer* è un plugin integrato nell'ambiente di sviluppo Remix, uno degli IDE più diffusi per Solidity. Questo strumento esegue un'analisi statica del codice, evidenziando vulnerabilità comuni, problemi di ottimizzazione e suggerendo miglioramenti per la sicurezza e l'efficienza degli smart contract.

L'uso combinato dei due permette di ottenere una verifica più completa del codice, facilitando l'identificazione precoce di problematiche potenzial-

mente critiche prima della fase di deploy su blockchain.

Di seguito sono riportate le principali problematiche segnalate da Solhint:

- **Visibilità non esplicitamente dichiarata nei costruttori:** Solhint raccomanda di specificare esplicitamente la visibilità delle funzioni, inclusi i costruttori. Sebbene dalla versione 0.7.0 di Solidity in poi la visibilità dei costruttori sia implicita (`public`), dichiararla rende il codice più chiaro e conforme alle best practice.
- **Messaggi di errore troppo lunghi nei `require(...)`:** un messaggio di errore fornito all'interno di una istruzione *require* risulta troppo lungo. Per ragioni di leggibilità e risparmio di gas, è consigliabile mantenere i messaggi sotto i 32 caratteri e usare formulazioni sintetiche ma chiare.
- **Utilizzo di `block.timestamp` nella logica di business:** l'utilizzo del timestamp del blocco (`block.timestamp`) può essere soggetto a leggere manipolazioni da parte dei miner e, se utilizzato per decisioni critiche, può introdurre vulnerabilità. Tuttavia, nel contratto in oggetto il timestamp è impiegato unicamente per scopi descrittivi e di registrazione storica delle azioni, quindi il rischio è trascurabile e l'uso è accettabile.

In sintesi, le segnalazioni di Solhint non evidenziano bug funzionali ma suggeriscono miglioramenti stilistici, di leggibilità e conformità agli standard di sviluppo. Tali accorgimenti contribuiscono alla qualità del codice e facilitano la manutenzione e l'evoluzione del contratto nel tempo.

Per quanto riguarda le problematiche evidenziate da Remix Analyzer, oltre a quelle già evidenziate da Solhint, troviamo:

- **Nomi simili di variabili:** nomi troppo simili possono generare confusione nella lettura e manutenzione del codice, aumentando il rischio di bug dovuti a fraintendimenti. Tuttavia in alcuni contesti sono state utilizzate appositamente variabili con nomi simili.
- **Uso di *require* invece di *assert*:** Solhint segnala che in alcuni casi è preferibile controllare alcune condizioni con *assert*, che si usa per verifiche interne che non devono mai fallire se non per bug. Tuttavia, nelle varie funzioni del contratto, l'uso di *require* è corretto perché verifica condizioni che possono legittimamente risultare false, come il controllo che mittente e destinatario siano utenti registrati. In questo caso, *require* gestisce input esterni non validi in modo appropriato,

evitando consumi eccessivi di gas e permettendo una gestione pulita degli errori.

In sintesi, le segnalazioni di Remix Solidity Analyzer non evidenziano errori funzionali critici, ma indicano aspetti migliorabili in termini di leggibilità e ottimizzazione dei costi di esecuzione.