Relazione Gruppo 2

Anno Accademico 2021/2022

Indice

Lista dei codici mostrati

Elenco delle tabelle

1 Scopo del progetto

Lo scopo del nostro progetto è quello di realizzare un'applicazione per la gestione della produzione e dell'acquisto di prodotti mediante l'utilizzo di una blockchain.
Le fasi che sono risultate necessarie per lo sviluppo del progetto sono:

- Raccolta dei requisiti: in questa fase si è andati a raccogliere tutti i requisiti necessari per capire al meglio cosa il software dovrà essere in grado di fare;
- Definizione delle specifiche: i requsiti vengono trasformati in specifiche, si inizia a ragionare in termini di architetture del software ovvero in termini di descrizione delle varie componenti del software. Questa fase viene anche chiamata fase di design del software;
- Implementazione del software: si inizia a scrivere il codice del software
- Testing: una volta terminata la fase di implementazione del software questo dovrà essere collaudato, mediante una serie di test volti a verificare se il software implementato è corretto e risponde ai requsiti

2 Raccolta e analisi dei requisiti

2.1 Diagrammi *i**

Il diagramma dei casi d'uso viene utilizzato per tradurre i requisiti, raccolti mediante interviste, in un primo modello che ha l'obiettivo di indicare in quali scenari il software che stiamo progettando dovrà operare ovvero quali sono le funzionalità del software e quali sono gli attori con i quali il software dovrà interagire.

2.2 Errori durante l'ingegnerizzazione dei requisiti

La scoperta di problemi di sicurezza in fase di analisi dei requisiti di progettazione risulta essere molto più economico che concentrarsi, su questi, solamente nella fase di sviluppo e di collaudo. L'analisi del rischio effettuata durante l'analisi dei requisiti viene chiamata **valutazione preliminare del rischio**.

2.3 Valutazione del rischio

Una valutazione del rischio consiste nel comprendere quali sono gli asset da proteggere in relazione alla necessità di difenderlo, in base al suo valore. Bisogna valutare l'esposizione ovvero l'impatto a cui si va incontro nel caso in cui questo venga violato o danneggiato. Per valutare le possibili minacce a cui è esposto l'asset ci si basa sul modello STRIDE.Una volta individuate le minacce e l'impatto, bisogna valutare se ci sono delle vulnerabilità che possano rendere le minacce un qualcosa di concreto, si ha quindi quella che viene chiamata una **valutazione dei possibili attacchi**, iniziando a ragionare come un possibile attaccante.
Una volta individuate le mosse che può compiere un attaccante, si stabiliscono quali possono essere le misure di controllo, di mitigazione o di riduzione del rischio da prendere. Quindi per ogni asset, tenendo in considerazione i possibili attacchi che si è pensato possano avvenire, si vanno a prendere determinate scelte tecnologiche per evitare che quell'attacco si verifichi. Di queste bisogna però tenere in considerazione la fattibilità, l'accettabilita ed i costi.
A questo punto è terminata l'analisi dei requisiti perché si è arrivati ad avere delle specifiche.

2.4 Modello STRIDE

Per identificare le minacce andiamo ad utilizzare il modello STRIDE il cui nome deriva dalle iniziali delle sei categorie di possibili minacce in questo modello considerate e che vengono riportate nella tabella 2.

Minaccia	"Traduzione"	Proprietà violata	Definizione
Spoofing	Furto d'dentità	Autenticazione	Impersonare qualcun altro o qualcos'altro
Tampering	Alterare/Modificare	Integrità	Modificare dati o codice
Repudiation	Ripudio	Responsabilità	Negare l'aver compiuto un'azione
Information Disclosure	Diffusione delle informazioni	Confidenzialità	Diffondere delle informazioni a qualcuno che non è autorizzato a vederle
Denial of service	Indisponibilità di un servizio	Disponibilità	Disabilitare o peggiorare un servizio agli utenti
Elevation of Privilege	Innalzamento dei privilegi	Autorizzazione	Ottenere "capacità" senza autorizzazione

Tabella 1: Modello STRIDE

Nell'analisi delle minacce per i nostri scopi, considereremo anche la violazione degli obiettivi riportati nella tabella.

Minaccia	"Traduzione"	Proprietà violata	Definizione
Danger	Pericolo	Sicurezza	Mettere in pericolo qualcuno o qualcosa
Unreliability	Inaffidabilità	affidabilità	Incapacità di essere affidabili o creduti
Absence of Resilience	assenza di resilienza	Resilienza	Incapacità di ristabilirsi velocemente da un incidente informatico

Tabella 2: Modello DUA

2.5 Individuazione degli attacchi

Una volta individuate le minacce bisogna individuare i possibili attacchi. Esistono una serie di strumenti che si possono utilizzare per comprendere i possibili attacchi:

- **CASI D'USO (USE CASE)**: sono una lista di azioni o eventi che tipicamente definiscono l'interazione tra un attore ed un sistema per raggiungere un obiettivo. L'attore può essere un uomo o un altro sistema esterno.
- **CASI D'USO IMPROPRIO (MISUSE CASE)**: un caso di uso improprio è l'inverso di un caso d'uso, una funzione che il sistema non dovrebbe permettere. Ovvero una completa sequenza di azioni fatta da un mis-actor, che risulta in una perdita per l'organizzazione o a qualche specifico stakeholder. Un mis-actor è l'inverso di un attore ovvero qualcuno che non intenzionalmente o accidentalmente, da il via ad un caso di utilizzo improprio e che il sistema non supporta nel fare ciò.
- **CASI D'ABUSO (ABUSE CASE)**: è un tipo di interazione completa tra un sistema ed un o più attori, dove il risultato dell'interazione è dannoso per il sistema, per uno degli attori o per uno degli stakeholder.

Nell'ambito di questo progetto useremo il **misuse** case per intendere un attacco involontario, mentre con **abuse case** si intenderanno gli attacchi volontari ovvero fatti con l'intenzione di nuocere.
Per ogni specifico elemento di un caso d'uso, c'è uno specifico elemento nei diagrammi *i**. La struttura rimane la medesima anche per ABUSE e MISUSE CASE. Va considerato che non per forza tutti i campi vanno riempiti, si riempiono solo quelli necessari.

2.6 Attack Tree

Oltre alla descrizione di un attacco mediante l'utilizzo di Abuse e Misuse case, questa deve essere fatta anche mediante gli attack tree.
Il concetto di attack tree deriva direttamente dal concetto di fault tree (ovvero albero dei guasti), che vengono utilizzati da chi si occupa di manutenzione e cerca di progettare il sistema in modo da essere affidabile da una parte o resiliente dall'altra.
Con gli attack tree si vanno quindi a vedere i vari modi con i quali è possibile attaccare un determinato asset per andare quindi a violare quella determinata proprietà o quel determinato obiettivo di sicurezza associato a quell'asset. Con questi modi di attacco si va poi a dare una valutazione sulla probabilità.

2.7 Mitigazione del rischio

Si deve ora ragionare su quali sono le possibili scelte che mi permettono di mitigare un determinato rischio. Il modello STRIDE (esteso) ci permette di indicare per ogni possibile minaccia e per ogni possibile asset delle "contromisure".
Una volta individuate le possibili contromisure che possono essere in alternativa tra di loro o possono essere utilizzate insieme, bisogna valutare:

- fattibilità: tecnologica, accetazione del rischio;
- costi delle contromisure fattibili: quanto costa implementare una data soluzione;
- probabilità di attacco residua;
- rischio residuo a seguito della contromisura.

Si è fatta una prima valutazione del rischio, partendo soltanto da una certa minaccia ad un determinato asset, si è così determinato il rischio, si sono trovate poi delle soluzioni andando poi a calcolare il rischio residuo. Si ha quindi da una parte il rischio residuo, dall'altra il costo complessivo. Il rischio iniziale deve risultare maggiore del rischio residuo del costo, per decidere se implementare quelle misure di sicurezza.

2.8 CAPEC E ATTACK

Per arrivare a compilare un ABUSE CASE o un MISUSE CASE, significa che si è ragionato sul fatto che un dato attacco o un utilizzo errato del software si possano verificare.

2.8.1 CAPEC - Enumerazione e classificazione dei comuni pattern di attacco

Questo catalogo viene gestito e mantenuto dal MITRE. Qui viene nominato esplicitamente il concetto di pattern che è legato strettamente al concetto di design pattern. Un design pattern nasce dall'idea che se strutturare codice in un certo modo in passato ci è tornato utile in progetti precedenti per risolvere un problema, probabilmente quando si ripresenterà un problema simile, si può adottare la stessa struttura.
Su questa scia, ragionando in termini di sicurezza, in particolare di software sicuro, è possibile notare che, anche nel modo in cui un software viene attaccato e anche nel modo in cui certi obiettivi di sicurezza di un software

vengono violati, si possono individuare dei pattern, in quanto gli attaccanti tendono a ripetere determiante modalità di azioni.
Si è provato quindi a stabilire quali sono i pattern che vengono seguiti nei vari attacchi. Questo anche perché il primo passo per costruire un sistema sicuro è conoscere il proprio nemico, se sappiamo come agisce chi ci vuole attaccare, siamo pronti a difenderci. Un discorso analogo può essere fatto per gli utenti che agiscono in modo errato involontariamente.
Il concetto di pattern di attacco si collega ad altri concetti, in particolare:

- Attack/Threat trees;
- Fault trees;
- Pattern di sicurezza, un pattern "positivo" che garantisce determianti obiettivi di sicurezza.

2.8.2 ATTACK

ATTACK è sempre gestisto dal MITRE. Questo repository può essere utilizzato sia nell'analisi preliminare del rischio, sia in fase di analisi del rischio on design time. Tale catalogo si concentra su come i singoli passi del flusso descritto in CAPEC possano essere svolti, ripontando quindi le tattiche e le tecniche che possono essere utilizzate dagli attaccanti. Data la strategia, ogni attaccante può utilizzare tattiche e tecniche diverse per portare a termine l'attacco.
Il termine **tattica** sta in qualche modo a rappresentare il perché di un attacco, ovvero l'esecuzione di una certa azione che vantaggio da all'attaccante. Invece, come fare ad eseguire un determinato attacco me lo dice la **tecnica**; quindi, per ogni tattica si possono poi utilizzare tecniche diverse.

2.8.3 CAPEC vs. ATTACK

CAPEC è nato con l'intento di dare un aiuto in fase di progettazione del software, contiene i pattern di attacco che riguardano la social engineering. ATTACK è molto orientato sul capire come l'attaccante può fare certe cose, questo repository torna comunque utile in fase di progettazione. Nasce con l'intento di difendere un'infrastruttura quindi capire come progettare al meglio le infrastrutture, rilevare in tempo gli attacchi e prendere i provvedimenti necessari. Infine è anche uno strumento che fa test di penetrazione, quindi che fa in generale collaudi di sicurezza, perché suggerisce alcuni modi in cui si può attaccare (suggerisce alcune tecniche) quindi si può poi utilizzarle per mettere alla prova il software, l'infrastruttura, il sistema in generale.

3 Progettazione del software

Progettare un software sicuro significa decidere quale architettura il nostro software dovrà avere, tenendo conto di buone pratiche di progettazione. Ogni scelta che viene presa in linea di principio dovrebbe essere guidata da un'analisi del rischio, scegliendo quindi alternative che mi permettono di ridurre il rischio, mantenendo i costi accettabili.
Per eseguire la fase di progettazione in maniera corretta esistono una serie di linee guida da rispettare, in modo da renderci consapevoli di quali possono essere i problemi di sicurezza che si andranno ad incontrare durante la fase di progettazione.
Esistono tre diverse tipologie di principi di progettazione: Saltzer e Schroeder, OWASP e Sommerville. Sommerville si rifà in un certo senso ai primi due principi ma aggiunge anche altre indicazioni di interesse che non sono condivise con questi. I principi enunciati da sommerville sono di più alto livello perché sono più metodologici che di implementazione vera e propria.

3.1 Analisi del rischio

Come per la fase preliminare, anche in questa fase è necessaria un'ulteriore analisi del rischio. Quando si va ad analizzare quali sono le tecniche di attacco che possono essere utilizzate, ovvero come queste si possono concretizzare, è necessario effettuare una serie di analisi e controlli ulteriori per arrivare a stimare la probabilità di attacco ed il possibile impatto. In particolare, si possono svolgere quattro tipologie di analisi:

- **Analisi di resistenza agli attacchi**: ci si basa sul modello STRIDE andando ad analizzare le minacce, formalizzando con ABUSE e MISUSE CASE e indicando eventuali attacchi che rendono concreta questa minaccia, considerando i cataloghi CAPEC e ATTACK. Il tutto viene poi formalizzato mediante gli attack tree;

- **Analisi delle ambiguità**: parte con l'obiettivo di scoprire ciò che non è noto, ovvero scoprire problemi nuovi. Questo tipo di analisi deve essere fatta da un team di analisti diversi da coloro che hanno implementato il codice;

- **Analisi delle debolezze**: si basa esclusivamente sulle scelte tecnologiche ed architetturali prese; anche in questo caso è possibile consultare cataloghi che riportano le debolezze e le vulnerabilità. Bisogna fare attenzione, nella parte di software da noi scritta, a non introdurre una

serie di debolezze in quantò queste potrebbero introdurre delle vulnerabilità, cercando anche di non fare affidamento sulle componenti di terze parti;

- **Analisi di sopravvivenza**: si analizza se il software attaccato è in grado di sopravvivere agli attacchi, andando a vedere come questo si comporta in caso di attacco. Si considerano quindi gli attack tree descritti nelle tre analisi precedenti, provando a simulare gli attacchi previsti e vedere come si comporta il software, ovvero se l'attacco si può effettivamente verificare, se il software è in grado di riconoscerlo, se è in grado di reagire ed infine se è in grado di sopravvivere al dato attacco.

4 Programmazione sicura

Il codice da noi scritto può riflettere degli errori di progettazione, spesso le vulnerabilità dipendono però da errori di programmazione, altri errori invece dipendono fortemente dal linguaggio di programmazione che stiamo utilizzando. I linguaggi di programmazione fanno quindi la differenza, alcuni sono più esposti di altri ad attacchi.

4.1 Dependable e secure programming guidelines

A prescindere dal linguaggio di programmazione che si utilizza, è possibile indicare una serie di linee guida per una programmazione sicura. Rispetto alle linee guida di Saltzer e Schneider che sono a livello di progettazione, queste sono di livello più basso, ovvero proprio di scrittura dei programmi.

- **Limitare la visibiltà di informazioni in un programma**: solo chi è autorizzato può accedere ai dati con oppurtune operazioni predefinite;
- **Controllare che tutti gli input siano corretti**: è necessario fare controlli sull'intervallo, sulla dimensione, sulla rappresentazione e sulla ragionevolezza;
- **Gestire tutte le eccezioni**: la gestione delle eccezioni aiuta il sistema ad essere resiliente;
- **Minimizzare l'utilizzo di costrutti che sono più proni all'errore**: si cerca di evitare o ridurre al minimo l'utilizzo di costrutti che hanno una elevata probabilità di farci commettere degli errori;

- **Fornire capacità di ripresa**: è importante che il sistema sia in grado in breve tempo di ristabilire le normali condizioni di lavoro, in vista di un malfunzionamento;
- **Controllare i limiti degli array**:è necessario fare attenzione a non utilizzare troppo spesso array che non hanno una dimensione prestabilita, in quanto c'è il rischio che si vada a scrivere su aree di memoria già scritte;
- **Includere timeout quando si chiamano componenti esterni**: se si stanno utilizando dei componenti esterni, non ci si può affidare completamente a questi; si imposta una finestra di tempo entro la quale la componente esterna dovrà rispondere, a fronte della richiesta, avendo fissato già un piano B in caso di non ricezione di risposta;
- **Dare un nome significativo alle costanti**: se nel programma si utilizzano valori che sono delle costanti è necessario assegnare un nome a tale valore per poi andarla ad utilizzare nel resto del programma.

Queste indicazioni che a prima vista sembrano indicazioni di buona programmazione se rispettate, non portano a problemi di programmazione che possono facilmente diventare problemi di sicurezza.
Esistono cataloghi che possono tornarci utili quali i CWE ma esistono anche cataloghi specifici per ogni lunguaggio di programmazione per arrivare poi alle linee guida creati e gestiti dal CERT.

5 Blockchain

Una blockchain è un registro digitale condiviso e a prova di manomissione che registra le transazione in una rete peer-to-peer pubblica o privata. Distribuito a tutti i nodi membri della rete, la blockchain registra, in maniera permanente, in una catena sequenziale di blocchi hashlink crittografici, la storia delle transazioni che si verificano tra i peer nella rete.
I portafogli contenenti le cripto-valute non sono legate al proprietario mediante il suo nome ma mediante dei numeri che, nello specifico, sono le chiavi pubbliche e private.
Il modo più semplice di pensare a una blockchain è quello di pensare ad un registro. Le scritture che vengono fatte all'interno di una blockchain prendono il nome di transazioni.
Il contratto tra le due parti è garantito da tanti testimoni, si utilizza quindi un'architettura distribuita con diversità, ogni testimone scrive nel registro del suo nodo. Si opta quindi per un **Distributed Ledger Tecnologies** ovvero

un tipo di database condiviso, replicato e sincronizzato tra i membri di un rete decentralizzata. Il libro mastro distribuito registra le transazioni, come lo scambio di beni o dati, tra i partecipanti della rete. Si considerano quindi:

- La struttura dati: utilizziamo una lista concatenata dove ogni blocco corrisponde a una pagina del registro quindi su ogni blocco sono riportate piu transazioni. Il legame tra un blocco e un altro è l'hash del blocco precedente (blockchain);
- Protocollo di registrazione della transazione: vanno definite le operazioni da compiere;
- Algoritmo del consenso: quello che è scritto nei vari registri deve essere consistente in quanto i nodi validatori si devono mettere d'accordo tra di loro.

Il problema del consenso è un problema difficile in quanto, nel momento in cui i nodi si devono mettere d'accordo sorgono problemi di vario tipo: la rete induce dei ritardi, la rete puo essere partizionata, dei nodi possono avere dei guasti o dei problemi di connessione, qualche nodo potrebbe agire in maniera diversa da quello che ci si aspetta. Bisogna quindi trovare degli algoritmi che permettano ai nodi di arrivare a delle decisioni. In base all'algoritmo di **fault tolerance** che si utilizza, cambia il numero di nodi sani che servono per validare una transazione.

5.1 Smart contract

Il concetto di smart contract è nato indipendentemente dal concetto di blockchain, l'obiettivo infatti è quello di formalizzare quanto scritto in un contratto "legale" nel linguaggio naturale e contenere le istruzioni per eseguire le clausole contrattuali. Questi contratti possono essere quindi formalizzati come un programma ovvero un insieme di istruzioni da eseguire.
Lo smart contract altro non è che codice la cui esecuzione è delegata ai vari nodi; il validatore deve mandare in esecuzione un metodo, vedere il risultato prodotto e poi confrontare i vari risultati ottenuti.
Nella realtà, gli smart contract non possono sostituire a tuti gli effetti i contratti legali. Per noi uno smart contract è un programma, quindi esso può avere vulnerabilità, errori ed inoltre non scala bene in quanto non è il massimo dell'efficienza far eseguire lo stesso pezzo di codice a molti nodi. Se si rileva che uno smart contract ha un problema, bisogna "cancellarlo" e scriverne uno nuovo.

5.1.1 Ethereum

Ethereum è una blockchain pubblica, per interagire con essa serve un Account, cioè un portafoglio; esso ha un ID di 20 bytes e uno stato, lo stato rappresenta il saldo del portafoglio . Per questa blockchain abbiamo 2 tipi di contratti che sono quelli esterni, cioè quelli associati agli utenti, e quelli di tipo 'Contract' a cui sono associati degli smart contract. In ethereum abbiamo una moneta (ethercoin) che viene utilizzata per pagare le transazioni.
Quorum è una versione di Ethereum aziendale privata e con permessi. Questa blockchain è gratuita, cioè di default non possiede una propria moneta, però è possibile definirne una, quindi le transazioni vengono eseguite senza utilizzare soldi.

5.1.2 Solidity

Solidity è un linguaggio di programmazione ad oggetti al alto livello utilizzato per implementare gli smart contract; si differenzia dagli altri linguaggi di programmazione soprattutto per il fatto che opera su blockchain. In solidity oltre ai classici tipi di dato abbiamo:

- **contract**: viene utilizzato per definire un nuovo contratto con tutte le operazioni ad esso associate;
- **address**: indica un indirizzo, ovvero l'identità che serve per identificare un attore;
- **mapping(chiave=>valore) public nomehashtable**: indica un hashtable dove nella chiave si indica la chiave con cui si vuole entrare, mentre in value si inserisce il valore che sarà letto, associato alla chiave in ingresso;
- **event nomevento(address from,addres to, uint amount)**: è una struttura dati che in questo caso indica un bonifico perché prende una somma di monete dall'address "from" e li invia all'address "to";
- **constructor()**: altro non è che il costruttore;
- **function nomefunzione(parametri)**: rappresenta il costrutto per definire una funzione;
- **require(condizione)**: viene utilizzato per porre delle condizioni/limitazioni.

Se si definisce un attributo come pubblico, questo è accessibile dagli altri smart contract, mentre per la struttura dati hashtable, averla resa pubblica significa che esiste una funzione che preso in ingresso un indirizzo mi restituisce un numero intero che è il bilancio.
Un modo che si ha per comunicare al mondo esterno quanto avviene nella blockchain è usando gli **event**. Quindi emettere eventi è un qualcosa che può essere fatto per comunicare con l'esterno.
Gli **address** hanno tutti la stessa lunghezza pari a 20 byte; con il modificatore payable possiamo dividere gli address in due tipi:

- address: quell'indirizzo non riceverà mai ether coins da uno smart contract;
- address payable: quell'indirizzo riceverà ether coin da uno smart contract.

Quando si utilizzano i tipi referenziati, si hanno a disposizione tre tipi di memoria:

- Memory: la memorizzazione è limitata alla vita della chiamata di una funzione esterna;
- Storage: la memorizzazione permane per tutto il tempo in esiste lo smart contract;
- CallData: la memorizzazione è limitata alla vita di una funzione esterna.

6 Testing

In questa fase il software viene collaudato, questo lo si fa mediante una serie di test volti a vedere se il software implementato è corretto, non contiene errori, risponde ai requisiti che sono stati esposti. La parte di collaudo si divide a sua volta in due, una parte più statica, dove il codice viene controllato senza essere effettivamente eseguito, ed una parte dimanica durante la quale il software viene eseguito e testato in tutte le sue parti.

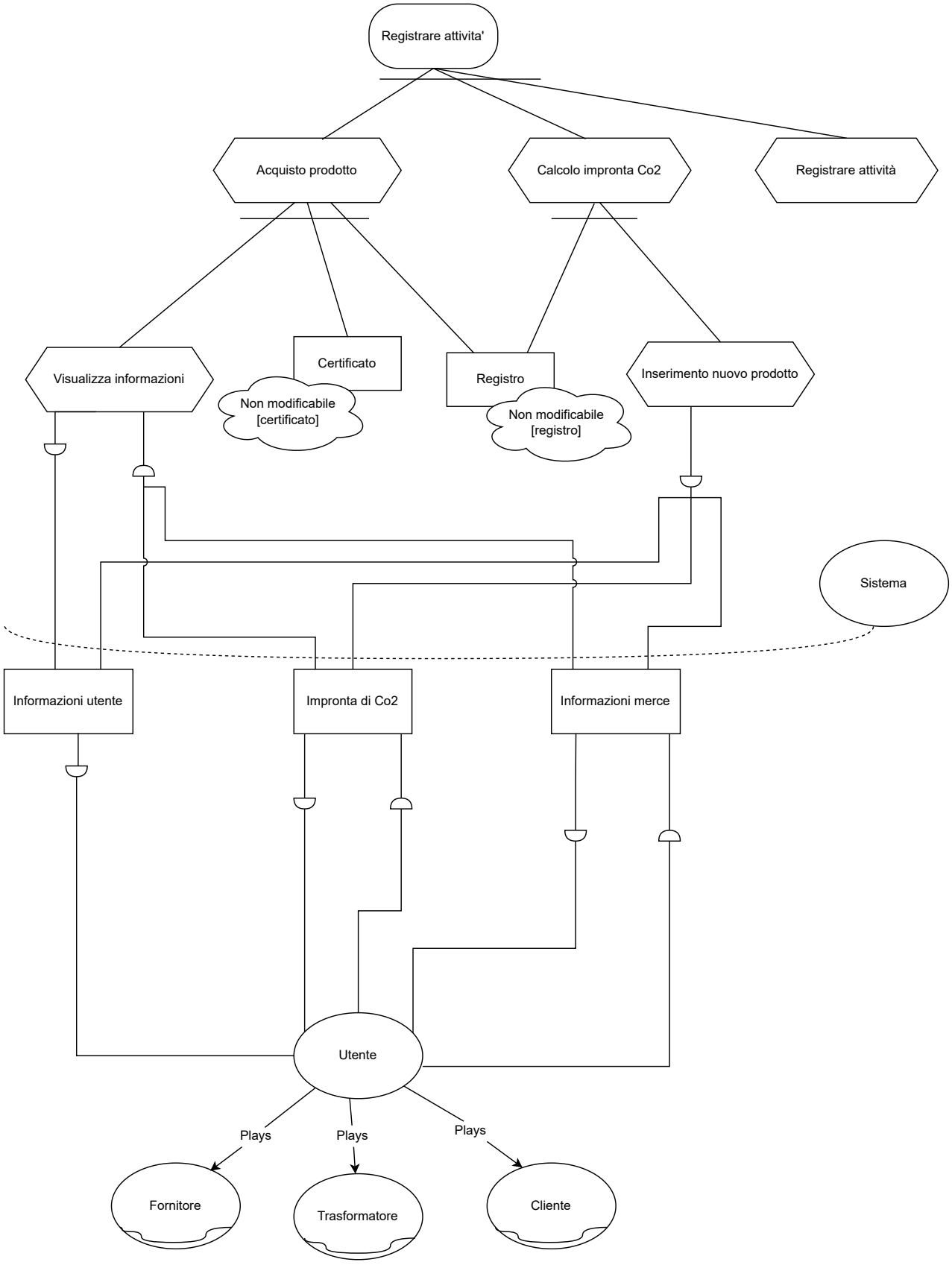
7 Realizzazione del software

Al termine di questa parte introduttiva, dove abbiamo spiegato quali sono gli step e gli strumenti che abbiamo utilizzato per la realizzazione di questa applicazione, andiamo a riportare e descrivere tutti i risultati e i documenti che sono stati prodotti durante i vari i step di sviluppo dell'applicazione. Nelle pagine a seguire verranno riportati:

- **Use Case** $\longrightarrow$ per descrivere come l'utente deve interagire con il sistema per raggiungere il suo obiettivo;
- **Abuse Case** $\longrightarrow$ per descrivere come l'utente che ha intenzioni malevole, può interagire con il sistema;
- **Misuse Case** $\longrightarrow$ per descrivere uno scenario negativo dove il sistema non ha il comportamento che ci si aspetta, perchè l'utente che vi interagisce commette degli errori (in maniera involontaria).

8 Diagramma dei casi d'uso

Attraverso il diagramma *i** andiamo a rappresentare quali sono le operazioni che un dato utente può compiere.
Per quanto riguarda la nostra applicazione, l'utente può impersonarsi in tre tipologie di attore: Fornitore, Trasformatore e Cliente. Per compiere una qualsiasi azione, l'utente dovrà indicare l'indirizzo di portafoglio associato a ciascun ruolo.


Registrare attivita'
Acquisto prodotto
Calcolo impronta Co2
Registrare attività
Visualizza informazioni
Certificato
Non modificabile [certificato]
Registro
Non modificabile [registro]
Inserimento nuovo prodotto
Sistema
Informazioni utente
Impronta di Co2
Informazioni merce
Utente
Plays
Plays
Plays
Fornitore
Trasformatore
Cliente

Andiamo ora a vedere nel dettaglio le operazioni che possono essere compiute da ciascun attore.

8.1 Inserimento di una nuova materia prima o di un nuovo prodotto

In questo caso d'uso l'obiettivo è l'inserimento di una materia prima dal parte del produttore e l'inserimento di un nuovo prodotto trasformato da parte del trasformatore. In entrambi i casi è obbligatorio fornire le proprie informazioni utente, al fine di poter registrare tale attività. In questa fase, è di fondamentale importanza, oltre che inserire le informazioni della merce, inserire anche l'emissione di CO2 che è stata prodotta. Una volta che il l'inserimento è andato a buon fine, tale operazione non può essere più modificata.

Case Type	Use Case	Case ID	case-01
Case Name	Inserimento di una nuova materia prima o prodotto trasformato		
Actors	Fornitore e Trasformatore		
Description	In questo caso d'uso gli attori inseriscono un nuovo prodotto nel sistema che può essere una materia prima prodotta dal fornitore oppure un prodotto trasformato, inserito dal trasformatore.		
Data	Informazioni dell'utente, informazioni sulla merce e l'impronta di Co2.		
Stimulus and preconditions	Gli attori devono inserire le proprie informazioni utente.		
Basic Flow	1) L'utente inserisce le sue informazioni personali. 2) Sceglie l'operazione 3) Inserisce i dati e conferma l'invio		
Alternative Flow	Nessuno		
Exception Flow	1) I dati di input non sono inseriti correttamente 2) Il sistema non riceve correttamente i dati che sono stati inseriti		
Response and Postconditions	Conferma del completamento dell'operazione		
Non Functional Requirements	Integrità, responsabilità, disponibilità, affidabilità, sicurezza		
Comments	Una volta effettuato l'inserimento non è più possibile effettuare delle modifiche su quel prodotto		

8.2 Acquisto di una materia prima o di un prodotto

In questo caso d'uso l'obiettivo è l'acquisto di una materia prima dal parte del trasformatore e l'acquisto di un prodotto trasformato da parte del consumatore. In entrambi i casi è obbligatorio fornire le proprie informazioni utente, al fine di poter registrare tale attività. Una volta che l'acquisto è andato a buon fine, tale operazione non può essere più modificata.

Case Type	Use Case	Case ID	case-02
Case Name	Acquisto di una materia prima o di un prodotto		
Actors	Trasformatore, cliente		
Description	In questo caso d'uso l'attore acquista un prodotto.		
Data	Informazioni dell'utente e informazioni merce.		
Stimulus and preconditions	L'attore deve inserire le proprie informazioni personali.		
Basic Flow	1) L'attore inserisce le proprie informazioni personali. 2) Sceglie l'operazione. 3) L'attore inserisce le informazioni sul prodotto che intende acquistare. 4) Visualizza l'impronta totale di Co2 di quel prodotto. 5) Conferma l'acquisto del prodotto.		
Alternative Flow	Nessuno		
Exception Flow	1) I dati di input non sono inseriti correttamente 2) Il sistema non riceve correttamente i dati che sono stati inseriti 3) Il sistema non visualizza correttamente l'impronta totale di Co2 del prodotto selezionato 4) Il sistema non restituisce il certificato o ne restituisce uno associato ad un altro prodotto.		
Response and Postconditions	Conferma del completamento dell'operazione e consegna del certificato.		
Non Functional Requirements	Integrità, responsabilità, disponibilità, affidabilità, sicurezza		
Comments	Una volta effettuato l'acquisto non è più possibile effettuare delle modifiche su quella operazione		

Al termine di questa fase di analisi, dopo aver disegnato i diagrammi *i*∗ e scritto i casi d'uso, abbiamo individuato i gli asset della nostra applicazione, cioè gli elementi più importanti che devono essere protetti. Li riportiamo nella seguente tabella.

8.3 Tabella degli asset da proteggere

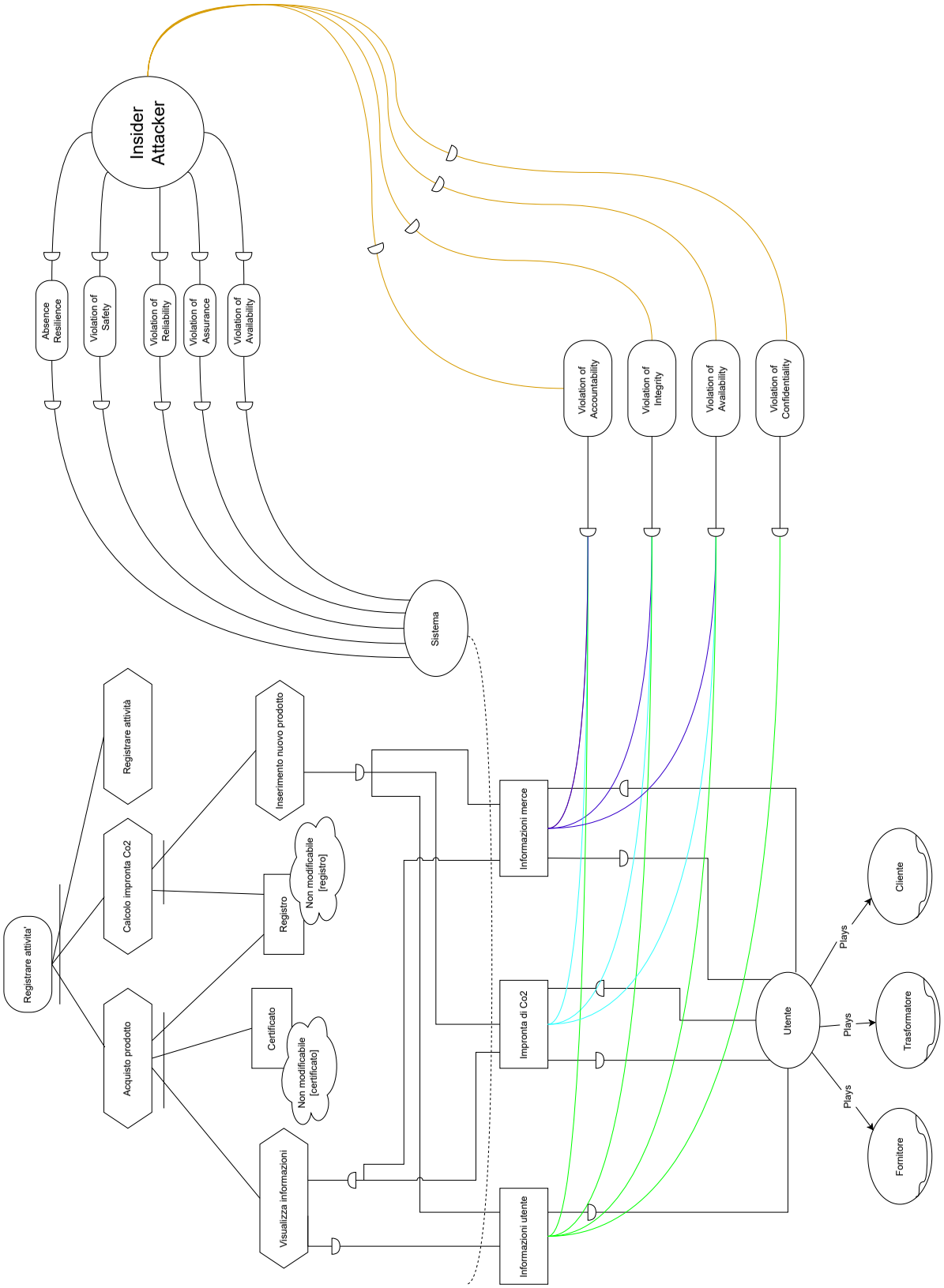
Asset	Value	Objective	Exposure
Certificato	**7**	Il certificato non può essere modificato dopo essere stato creato, in questo modo possiamo garantire il corretto funzionamento del sistema. Le informazioni del certificato devono essere non modificabili e coerenti con le attività a cui si riferisce. È una risorsa che deve essere sempre a disposizione per permettere agli utenti di acquistare il prodotto. La correttezza dei dati all'interno del certificato è fondamentale per osservare eventuali anomalie del sistema.	7 - La modifica del certificato comporta un danno all'utente che effettua un'operazione di acquisto, in quanto comprerebbe un prodotto con delle emissioni non corrette. 5 - Dati non corretti scambiati tra applicazioni del sistema portano a un funzionamento non corretto dello stesso. 7 - Se la risorsa viene resa non disponibile, il sistema non è in grado di erogare i suoi servizi. 7 - I servizi di acquisto rimangono offline fino a quando la risorsa non viene resa di nuovo disponibile.
Registro	**7**	Il registro non può essere modificato dopo che un'operazione viene registrata, in questo modo possiamo garantire il corretto funzionamento del sistema. Le informazioni contenute nel registro devono essere non modificabili e coerenti con le attività che sono state eseguite. L'integrità dell'informazione permette di risalire agli utenti che hanno compiuto quell'azione. La correttezza dei dati all'interno del registro è fondamentale per osservare eventuali anomalie del sistema.	7 - La modifica del registro comporta un'incoerenza tra le operazioni che sono state eseguite. 7 - Tale modifica comporta anche un'ulteriore perdita di integrità dei dati contenuti nel certificato. 7 - Se un'attività viene modificata, successivamente si ha una perdita di responsabilità da parte dell'utente che ha compiuto tale azione. 7 - Se la risorsa non viene resa disponibile, il sistema non è in grado di erogare i suoi servizi. 7 - I servizi rimarranno offline finquando la risorsa non viene resa nuovamente disponibile.
Registrare attività	**7**	Questa attività non deve essere modificata, ovvero il suo comportamento non deve essere alterato. In questo modo è possibile garantire che l'utente sia correttamente autorizzato ad eseguiere una determinata azione. Il corretto funzionamento di questa attività comporta il corretto funzionamento del sistema.	7 - Se l'attività viene bloccata, il sistema non è più in grado di funzionare finché tale attività non viene ripristinata. 6 - Se non vengono rispettati i criteri di autorizzazione, si avrà una violazione dei privilegi necessari, agli utenti del sistema, per eseguire determinate azioni.
Informazioni utente	**5**	Questa risorsa non deve essere accessibile ad altri utenti diversi dal proprietario, in questo modo possiamo garantire che altri utenti non compiano azioni con un'identità diversa dalla loro. Gli utenti possono compiere azioni solo con le loro informazioni utente, in questo modo è garantita la responsabilità. Le informazioni utenti devono essere note solo al suo proprietario, ovvero ne è proibita la condivisione.	5 - Se le informazioni utente sono accessibili ad altri utenti, essi potranno compiere delle azioni fraudolente nei confronti del proprietario. 5 - Se gli utenti possono compiere azioni inserendo informazioni di cui non sono i proprietari, allora non è possibile garantire la responsabilità, in quanto l'identità dell'utente non corrisponde a chi compie l'azione. 5 - Le informazioni utente non devono essere condivise, altrimenti abbiamo una fuga di informazioni.

9 Diagramma dei casi d'abuso

Attraverso il diagramma $i*$ andiamo a rappresentare quali sono le operazioni che un utente malintenzionato potrebbe compiere. Anche in questo caso, l'utente può impersonarsi in tre tipologie di attore: Fornitore, Trasformatore e Cliente.

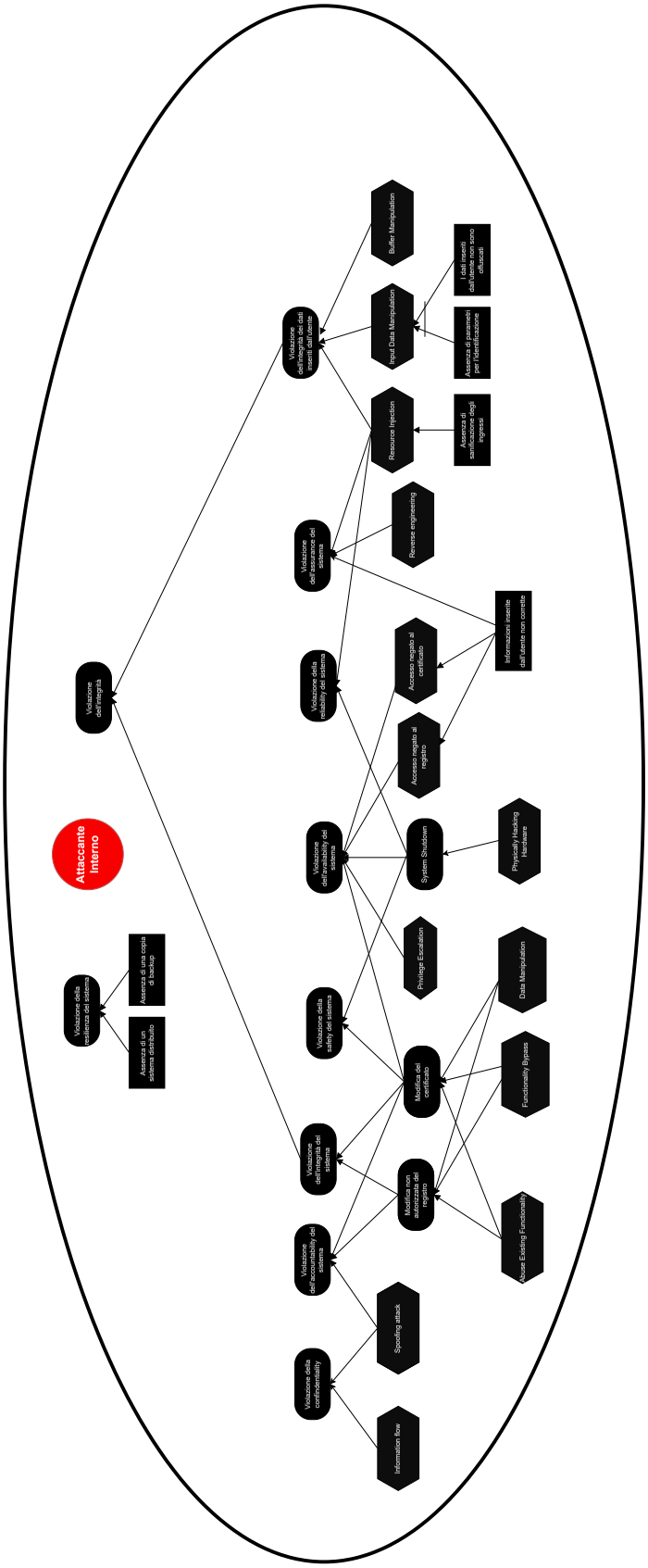
9.1 Attaccante interno al sistema

Come attaccante interno al sistema si pensa ad un utente malintenzionato in grado di attaccare il sistema interagendo direttamente con il dispositivo su cui è installato il software. Di seguito viene rappresentato il diagramma $i*$ relativo all'attaccante interno


Insider Attacker
Absence Resilience
Violation of Safety
Violation of Reliability
Violation of Assurance
Violation of Availability
Violation of Accountability
Violation of Integrity
Violation of Availability
Violation of Confidentiality
Sistema
Registrare attivita'
Registrare attività
Calcolo impronta Co2
Acquisto prodotto
Inserimento nuovo prodotto
Registro
Non modificabile [registro]
Certificato
Non modificabile [certificato]
Visualizza informazioni
Informazioni merce
Impronta di Co2
Informazioni utente
Utente
Plays
Plays
Plays
Cliente
Trasformatore
Fornitore

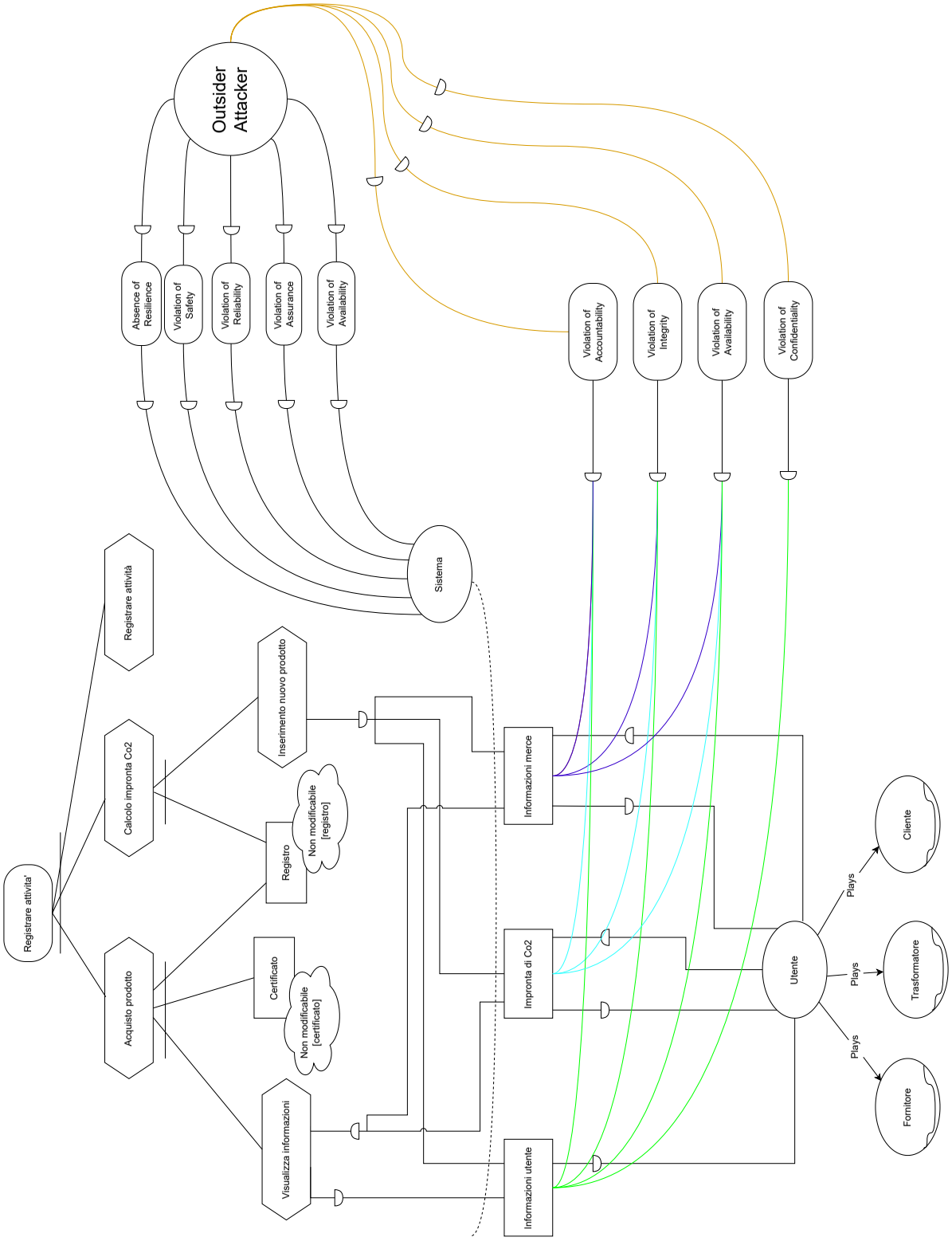
9.2 Attack tree attaccante interno

Con il seguente attack tree vengono rappresentati tutti gli attacchi che possono essere attuati da parte dell'attaccante interno. Tutti gli attacchi riportati sono stati presi dai cataloghi CAPEC e ATTACK.


Attaccante interno
Violazione dell'integrità
Violazione della reliability del sistema
Violazione dell'availability del sistema
Violazione della safety del sistema
Violazione dell'integrità dei dati
Violazione dell'accountability del sistema
Violazione della confidentiality
Violazione della resilienza del sistema
Assenza di una copia di backup
Assenza di un sistema distribuito
Violazione dell'autorizzazione del sistema
Violazione dei vincoli di inserti dall'utente
Buffer Manipulation
Input Data Manipulation
Resource Injection
Reverse engineering
I dati inseriti dall'utente non sono efficaci
Assenza di parametri per l'identificazione
Assenza di sanificazione degli ingressi
Informazioni inserite dall'utente non corrette
Accesso negato al certificato
Accesso negato al registro
System Shutdown
Privilege Escalation
Modifica del certificato
Modifica autorizzazioni del registro
Spoofing attack
Information flow
Physically Hacking Hardware
Data Manipulation
Functionality Bypass
Abuse Existing Functionality

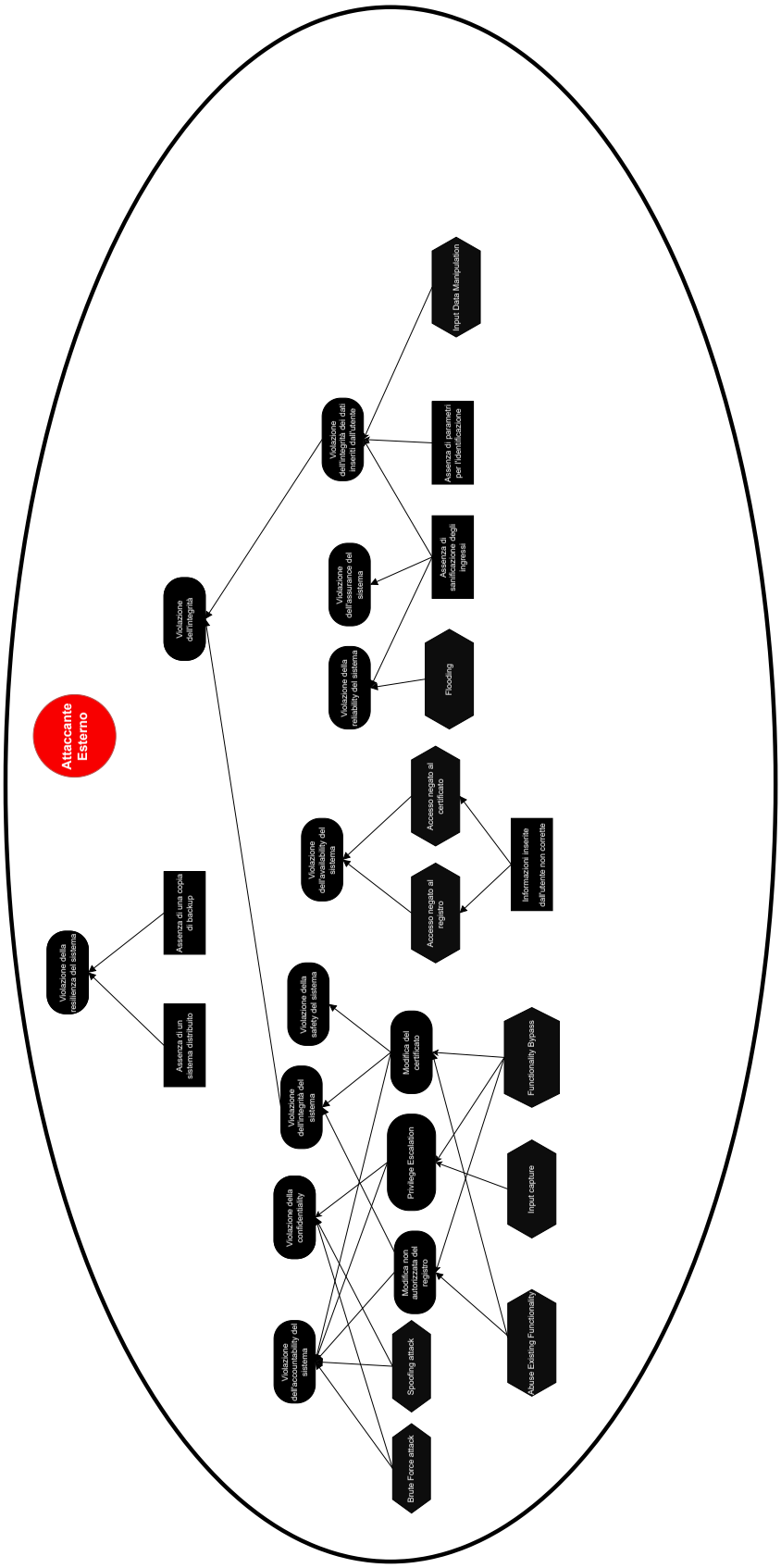
9.3 Attaccante esterno al sistema

Come attaccante esterno al sistema si pensa ad un utente malintenzionato in grado di attaccare il sistema senza interagire direttamente con il dispositivo su cui è installato il software. Di seguito viene rappresentato il diagramma *i*∗ relativo all'attaccante interno


Outsider Attacker
Absence of Resilience
Violation of Safety
Violation of Reliability
Violation of Assurance
Violation of Availability
Violation of Accountability
Violation of Integrity
Violation of Availability
Violation of Confidentiality
Sistema
Registrare attività
Calcolo impronta Co2
Acquisto prodotto
Registrare attivita'
Inserimento nuovo prodotto
Registro
Non modificabile [registro]
Certificato
Non modificabile [certificato]
Visualizza informazioni
Informazioni merce
Impronta di Co2
Informazioni utente
Utente
Plays
Plays
Plays
Cliente
Trasformatore
Fornitore

9.4 Attack tree attaccante esterno

Con il seguente attack tree vengono rappresentati tutti gli attacchi che possono essere attuati da parte dell'attaccante esterno. Tutti gli attacchi riportati sono stati presi dai cataloghi CAPEC e ATTACK.


Attaccante Esterno
Violazione della resilienza del sistema
Assenza di una copia di backup
Assenza di un sistema distribuito
Violazione dell'integrità
Violazione dell'integrità dei dati inseriti dall'utente
Violazione dell'assurance del sistema
Violazione della reliability del sistema
Violazione dell'availability del sistema
Violazione dell'integrità del sistema
Violazione della safety del sistema
Violazione della confidentiality
Violazione dell'accountability del sistema
Input Data Manipulation
Assenza di parametri per l'identificazione
Assenza di sanificazione degli ingressi
Flooding
Accesso negato al certificato
Accesso negato al registro
Informazioni inserite dall'utente non corrette
Modifica del certificato
Privilege Escalation
Modifica non autorizzata del registro
Spoofing attack
Brute Force attack
Functionality Bypass
Input capture
Abuse Existing Functionality

Andiamo ora a vedere nel dettaglio le operazioni che possono essere compiute da un utente malintenzionato.

9.5 Azioni compiute da un utente con identità rubata

In questo caso l'attaccante si è appropriato delle informazioni che permettono all'utente di eseguire operazioni di sua competenza. L'utente malevolo può quindi compiere operazioni, impersonandosi uno degli attori presenti nel sistema.

Case Type	Abuse Case	Case ID	case-01
Case Name	Azioni compiute da un utente con identità rubata.		
Actors	Sistema, attaccante interno ed esterno.		
Description	L'attaccante ha precedentemente acquisito le informazioni generali di un altro utente e può compiere a suo nome delle azioni come, ad esempio, l'inserimento e acquisto di un prodotto.		
Data	Registro e certificato.		
Stimulus and preconditions	L'attaccante ha precedentemente acquisito le informazioni generali di un altro utente. L'attaccante esterno oltre alle informazioni deve precedentemente ottenere la possibilità di eseguire del codice arbitrario sulla macchina del sistema.		
Attack Flow 1	L'attaccante inserisce le informazioni di un altro utente e inizia a compiere le azioni di inserimento e di acquisto di un prodotto		
Response and Postconditions	Il registro e il certificato contengono delle informazioni non corrette, che non sono state inserite dal proprietario delle informazioni.		
Non Functional Requirements	Viene violata la confidenzialità, la responsabilità e la disponibilità.		
Mitigations	Ogni utente sarà dotato di una chiave pubblica e una chiave privata.		
Comments	La chiave sarà lunga 32 byte, tutti generati casualmente.		

9.6 Modifica non autorizzata del registro

Il registro su cui vengono salvate tutte le operazioni compiunte dall'inizio fino a quel momento viene modificato dall'attaccante. Le informazioni contenute all'interno di esso risulteranno quindi alterate, con una conseguente compromissione dello stesso.

Case Type	Abuse Case	Case ID	case-02
Case Name	Modifica non autorizzata del registro.		
Actors	Sistema, attaccante sia interno che esterno.		
Description	Viene modificato il registro all'interno del quale sono contenute le tracce di tutte le operazioni eseguite dagli utenti (inserimento e acquisto prodotto). Quindi vengono modificate le informazioni in esso contenute.		
Data	Registro		
Stimulus and preconditions	Il registro può essere modificato (attack flow 1). Sono presenti bug all'interno delle funzioni che non sono ancora stati sistemati (attack flow 2 - 3). L'attaccante esterno deve precedentemente ottenere la possibilità di eseguire del codice arbitrario sulla macchina del sistema.		
Attack Flow 1	L'attaccante riesce ad accedere al file che contiene il nostro registro e ne modifica il contenuto.		
Attack Flow 2	L'attaccante sfrutta una vulnerabilità presente all'interno delle funzioni del sistema, come ad esempio l'assenza di un controllo, e grazie ad essa riesce a modificare il contenuto del registro abusando delle operazioni compiute dalla funzione.		
Attack Flow 3	L'attaccante sfrutta una vulnerabilità presente all'interno delle funzioni del sistema, e grazie ad essa riesce a modificare il contenuto del registro superando dei controlli imposti dalla funzione stessa.		
Response and Postconditions	Il registro non contiene informazioni corrette, i dati contenuti al suo interno non corrispondono a quelli inseriti dall'utente, quindi, il sistema non è affidabile.		
Non Functional Requirements	Viene violata la responsabilità e l'integrità.		
Mitigations	Utilizzeremo le Blockchain per rendere questa risorsa immutabile. Inoltre le funzioni saranno sempre monitorate per verificare la presenza di bug e anomalie. Infine il sistema utilizza un'architettura distribuita per aumentare la validità della Blockchain.		
Comments	Viene utilizzata la Blockchain di ethereum che si chiama quorum. Utilizziamo più calcolatori per avere il sistema distribuito.		

9.7 Modifica non autorizzata del certificato

Il certificato contiene le informazioni riguardanti la materia prima o il prodotto trasformato. Se questo viene modificato i dati riguardanti gli elementi vengono compromessi.

Case Type	Abuse Case	Case ID	case-03
Case Name	Modifica non autorizzata del certificato		
Actors	Sistema, attaccante sia interno che esterno.		
Description	Viene modificato il certificato all'interno del quale sono contenute tutte le informazioni riguardanti i prodotti inseriti. Quindi vengono modificate le informazioni in esso contenute.		
Data	Certificato.		
Stimulus and preconditions	Il certificato può essere modificato (attack flow 1). Sono presenti bug all'interno delle funzioni che non sono ancora stati sistemati (attack flow 2 - 3). L'attaccante ha accesso ai dati inseriti dall'utente. L'attaccante esterno deve precedentemente ottenere la possibilità di eseguire del codice arbitrario sulla macchina del sistema.		
Attack Flow 1	L'attaccante riesce ad accedere al file che contiene il nostro certificato e ne modifica il contenuto.		
Attack Flow 2	L'attaccante sfrutta una vulnerabilità presente all'interno delle funzioni del sistema, come ad esempio l'assenza di un controllo, e grazie ad essa riesce a modificare il contenuto del certificato abusando delle operazioni compiute dalla funzione.		
Attack Flow 3	L'attaccante sfrutta una vulnerabilità presente all'interno delle funzioni del sistema, e grazie ad essa riesce a modificare il contenuto del certificato superando dei controlli imposti dalla funzione stessa.		
Attack Flow 4	L'attaccante riesce ad accedere ai dati inseriti dall'utente mentre vengono scambiati tra l'applicazione e la Blockchain e ne modifica il contenuto.		
Response and Postconditions	Il certificato non contiene informazioni corrette, i dati contenuti al suo interno non corrispondono a quelli inseriti dall'utente, quindi, il sistema non è affidabile.		
Non Functional Requirements	Viene violata la responsabilità, l'integrità, la disponibilità e la sicurezza.		
Mitigations	Utilizzeremo le Blockchain per rendere questa risorsa immutabile. Inoltre le funzioni saranno sempre monitorate per verificare la presenza di bug e anomalie. Infine il sistema utilizza un'architettura distribuita per aumentare la validità della Blockchain.		
Comments	Viene utilizzata la Blockchain di ethereum che si chiama quorum. Utilizziamo più calcolatori per avere il sistema distribuito.		

9.8 Violazione della disponibilità del sistema

Il questo caso, l'attaccante rende non più disponibile il servizio offerto dal software in modo tale che nessuno possa interagire con esso.

Case Type	Abuse Case	Case ID	case-04
Case Name	Violazione del availability del sistema		
Actors	Sistema, attaccante sia interno che esterno.		
Description	Il sistema non è disponibile, quindi non è in grado di soddisfare le richieste fatte dagli utenti. Viene violata la availability del sistema.		
Data	Il sistema.		
Stimulus and preconditions	Accesso fisico alla macchina su cui è installato il sistema (Attack flow 1). Sfruttando una vulnerabilità del sistema l'attaccante riesce a modificare i suoi privilegi (Attack flow 2). L'attaccante esterno deve precedentemente ottenere la possibilità di eseguire del codice arbitrario sulla macchina del sistema.		
Attack Flow 1	L'attaccante può eseguire del codice che spegne il sistema oppure spegnere fisicamente la macchina su cui è installato il sistema.		
Attack Flow 2	L'attaccante sfrutta gli scarsi controlli in fase di autenticazione e riesce ad ottenere privilegi superiori a quelli previsti per il suo ruolo.		
Response and Postconditions	Il sistema dopo aver subito uno di questi attacchi non è disponibile o le sue risorse non lo sono, quindi non è in grado di soddisfare le richieste degli utenti.		
Non Functional	Viene violata l'availability.		
Mitigations	Andremo ad utilizzare le Blockchain, in questo modo il sistema sarà distribuito, e il malfunzionamento di un nodo non comporta il malfunzionamento del sistema. Inoltre la fase di autenticazione sarà eseguita dagli smart contract le cui funzioni non possono essere modificate.		
Comments	Nessuno.		

9.9 Violazione dell’affidabilità del sistema

Il questo caso, il sistema non si comporta come l’utente si aspetta. L’utente non è in grado di interagire in maniera corretta con esso.

Case Type	Abuse Case	Case ID	case-05
Case Name	Violazione del reliability		
Actors	Sistema, attaccante sia interno che esterno.		
Description	Il sistema non risponde in maniera corretta alle richieste dell'utente, ovvero il sistema non si comporta come l'utente si aspetta. Viene violata l'affidabilità del sistema.		
Data	Sistema, Certificato.		
Stimulus and preconditions	Il sistema non effettua controlli sul traffico, inoltre non ha una contromisura per limitare la frequenza delle richieste effettuate da un'entità (Attack flow 1) . L'assenza di sanificazione degli ingressi nel sistema permette all'attaccante di iniettare stringhe, codice eseguibili oppure ha fisicamente accesso al sistema(Attack flow 2-3). L'attaccante esterno deve precedentemente ottenere la possibilità di eseguire del codice arbitrario sulla macchina del sistema.		
Attack Flow 1	L'attacante invia un numero elevato di richieste per consumare di tutte le risorse del sistema.		
Attack Flow 2	L'attaccante sfrutta l'assenza di controlli sui dati di input per inserire stringhe di codice eseguibile.		
Attack Flow 3	L'attaccante può eseguire del codice che spegne il sistema oppure spegnere fisicamente la macchina su cui è installato il sistema.		
Response and Postconditions	Il sistema dopo aver subito uno di questi attacchi non è più in grado di soddisfare le richieste degli utenti o non le soddisfa nel modo corretto.		
Non Functional Requirements	Viene violata la reliability.		
Mitigations	Andremo ad utilizzare le Blockchain, in questo modo il sistema sarà distribuito, e il malfunzionamento di un nodo non comporta il malfunzionamento del sistema. La possibilità che l'attaccante possa compromettere tutti i nodi che compongono l'infrastruttura risulta essere minore, rendendo quindi il sistema più affidabile. Inoltre vengono aggiunti dei controlli per sanificare gli ingressi.		
Comments	Nessuno.		

9.10 Violazione dell’affidabilità del sistema

Le informazioni riguardanti le materie prime ed i prodotto che vengono inserite dall’utente sono corrette, queste vengono però compromesse dall’attaccante che intercetta le richieste tra utente e sistema.

Case Type	Abuse Case	Case ID	case-06
Case Name	Violazione dell'integrita' dei dati inseriti dall'utente		
Actors	Sistema, attaccante sia interno che esterno.		
Description	L'attaccante intercetta le richieste che l'utente invia al sistema (inserimento e acquisto di un prodotto) e ne modifica il contenuto.		
Data	Certificato		
Stimulus and preconditions	I dati scambiati tra l'utente e il sistema non sono protetti o offuscati, mancano dei parametri per l'identificazione degli utenti. Il buffer del sistema non è protetto.		
Attack Flow 1	I dati che vengono inseriti nel certificato sono compromessi perché l'attaccante è riuscito a modificarli durante l'invio della richiesta al sistema.		
Response and Postconditions	I dati inviati dall'utente sono stati alterati dall'attaccante quindi non sono più integri.		
Non Functional Requirements	Viene violata l'integrità		
Mitigations	Andremo ad utilizzare la crittografia asimmetrica in modo che i dati inseriti dall'utente non siano visibili e modificabili per un attaccante, in questo modo i dati vengono registrati correttamente nel certificato.		
Comments	Nessuno.		

9.11 Violazione della confidenzialità per le informazioni utente

Il sistema permette ad utenti che non hanno i permessi necessari di accedere a funzionalità che non sono di loro competenza.

Case Type	Abuse Case	Case ID	case-07
Case Name	Violazione della confidenzialità (Informazioni utente)		
Actors	Sistema, attaccante sia interno che esterno.		
Description	Il sistema permette, ad utenti che non hanno i permessi necessari, di ottenere informazioni a cui non possono accedere.		
Data	Informazioni utente.		
Stimulus and preconditions	Il sistema restituisce troppe informazioni all'utente, alcune delle quali possono essere usate per un attacco e non sono necessarie. Inoltre il sistema richiede informazioni confidenziali (credenziali di accesso). L'attaccante esterno deve precedentemente ottenere la possibilità di eseguire del codice arbitrario sulla macchina del sistema.		
Attack Flow 1	L'attacante sfrutta le informazioni che gli vengono comunicate dal sistema per accedere a informazioni a cui non deve poter accedere.		
Response and Postconditions	L'attaccante ha avuto accesso a informazioni a cui non poteva accedere.		
Non Functional Requirements	Viene violata la confidenzialità.		
Mitigations	Andremo ad utilizzare le Blockchain, in questo modo un utente deve inserire nel sistema solo il suo indirizzo di portafoglio che è pubblico. Inoltre, il sistema comunica all'utente solo le informazioni strettamente necessarie, le quali non possono essere usate per un attacco.		
Comments	Nessuno.		

9.12 Violazione della sicurezza del sistema

L'utente non interagisce in maniera corretta con il sistema, generando errori che potrebbero non far funzionare lo stesso.

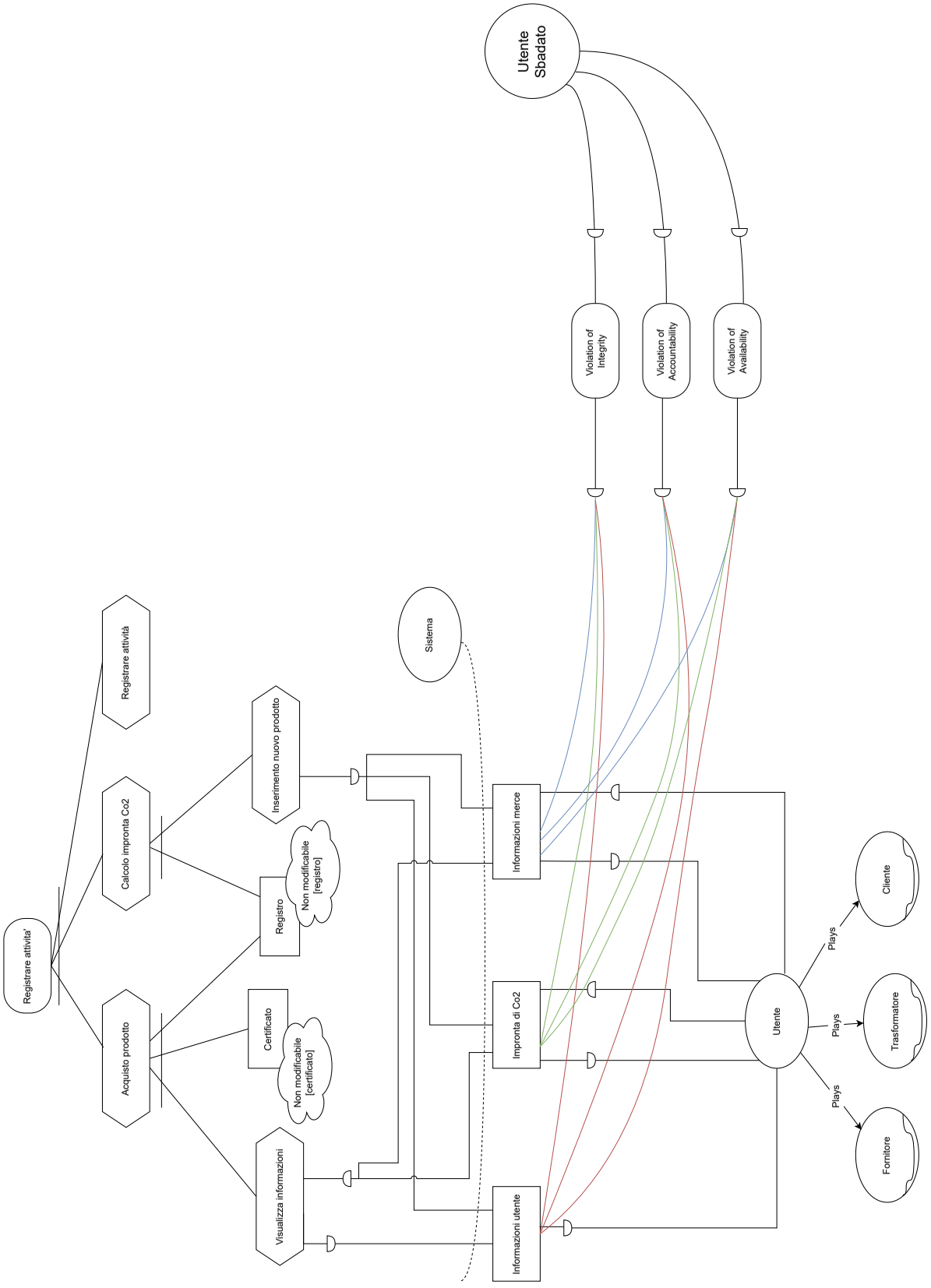
Case Type	Abuse Case	Case ID	case-08
Case Name	Violazione dell' assurance		
Actors	Sistema, attaccante sia interno che esterno.		
Description	Il sistema si trova a dover gestire comportamenti non corretti dell'utente (attaccante).		
Data	Sistema, informazioni utente, registrare attività.		
Stimulus and preconditions	Il sistema non effettua controlli sui dati in ingresso. Inoltre, il codice eseguibile non è protetto. L'attaccante esterno deve precedentemente ottenere la possibilità di eseguire del codice arbitrario sulla macchina del sistema.		
Attack Flow 1	L'attacante, come dato di input richiesto dal sistema, inietta delle risorse.		
Attack Flow 2	L'attaccante cerca di scoprire delle vulnerabilità analizzando il codice sorgente del sistema.		
Response and Postconditions	Il sistema non riesce a gestire questi comportamenti dell'utente (attaccante).		
Non Functional Requirements	Viene violata l'assurance.		
Mitigations	Vengono inseriti dei controlli per sanificare gli ingressi. Inoltre, si applicheranno tecniche di offuscamento al codice eseguibile in maniera tale da renderlo incomprensibile per l'attaccante.		
Comments	Nessuno.		

10 Diagramma dei casi di misuso

Attraverso il diagramma *i*∗ andiamo a rappresentare quali sono le operazioni che un utente sbadato potrebbe compiere. Anche in questo caso, l'utente può impersonarsi in tre tipologie di attore: Fornitore, Trasformatore e Cliente.

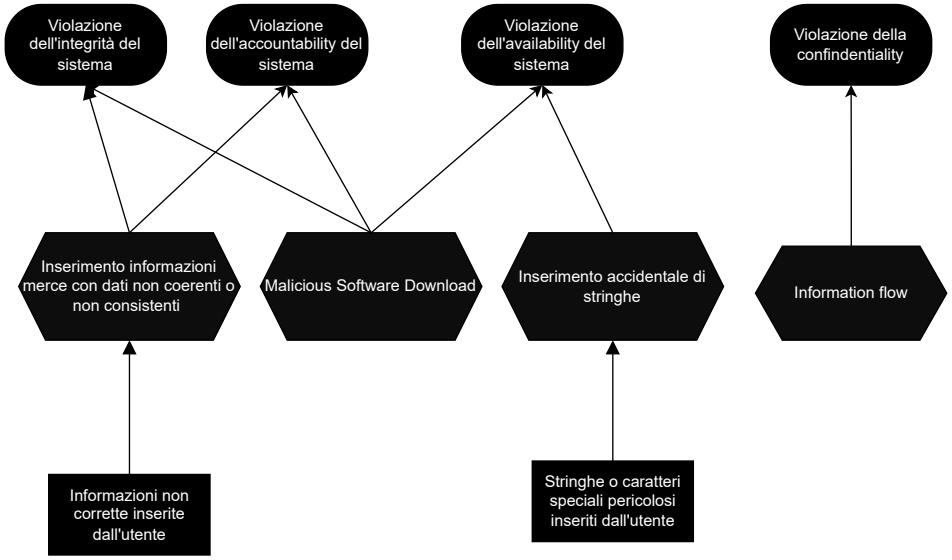
10.1 Utente sbadato

L'utente sbadato è un utente interno al sistema che, nel compiere determinate operazioni, potrebbe compromettere il corretto funzionamento dello stesso. Di seguito viene rappresentato il diagramma *i*∗ relativo all'utente sbadato


Utente Sbadato
Violation of Integrity
Violation of Accountability
Violation of Availability
Sistema
Registrare attivita'
Registrare attività
Calcolo impronta Co2
Acquisto prodotto
Inserimento nuovo prodotto
Registro
Non modificabile [registro]
Certificato
Non modificabile [certificato]
Visualizza informazioni
Informazioni merce
Impronta di Co2
Informazioni utente
Utente
Plays
Plays
Plays
Cliente
Trasformatore
Fornitore

10.2 Attack tree utente sbadato

Con il seguente attack tree vengono rappresentati tutti gli attacchi che possono essere attuati da parte dell'utente sbadato. Tutti gli attacchi riportati sono stati presi dai cataloghi CAPEC e ATTACK.



Andiamo ora a vedere nel dettaglio le operazioni che possono essere compiute da un utente sbadato.

10.3 Inserimento informazioni non corrette da parte dell'utente

In questo caso l'utente sbadato inserisce informazioni non corrette nel sistema, andandone a compromettere il corretto funzionamento.

Case Type	Misuse Case	Case ID	case-01
Case Name	Inserimento informazioni non corrette da parte dell'utente		
Actors	Utente sbadato.		
Description	L'utente inserisce erroneamente delle informazioni non corrette.		
Data	Informazioni inserite dall'utente.		
Stimulus and preconditions	Il sistema non effettua controlli sugli ingressi.		
Attack Flow 1	L'utente non si accorge che sta inserendo erroneamente delle informazioni non corrette.		
Response and	I dati non corretti vengono inviati al sistema e vengono registrati.		
Non Functional Requirements	Viene violata l'integrità, l'accountability e la disponibilità.		
Mitigations	I dati inseriti dall'utente vengono sanificati dal sistema prima di essere registrati.		
Comments	Nessuno.		

10.4 Installazione accidentale di software malevolo

In questo caso l'utente sbadato installa accidentalmente, sulla macchina in cui il software viene eseguito, un software malevolo che compromette il corretto funzionamento del sistema.

Case Type	Misuse Case	Case ID	case-02
Case Name	Installazione accidentale di software malevolo		
Actors	Utente sbadato.		
Description	L'utente accidentalmente installa del software malevolo sulla propria macchina.		
Data	Informazioni inserite dall'utente.		
Stimulus and preconditions	Scarsa conoscenza dell'utente sulla sicurezza informatica.		
Attack Flow 1	L'utente durante il suo lavoro clicca su un qualche esterno e installa un malware.		
Response and Postconditions	Il malware modifica i dati inseriti dall'utente senza che quest'ultimo se ne accorga. Oppure, gli impedisce di interagire con il sistema.		
Non Functional Requirements	Viene violata l'integrità, l'accountability e la disponibilità.		
Mitigations	Viene erogato un corso base per gli utenti per aumentare le loro compotenze in materia di sicurezza informatica.		
Comments	Installare un anti-virus aggiornato.		

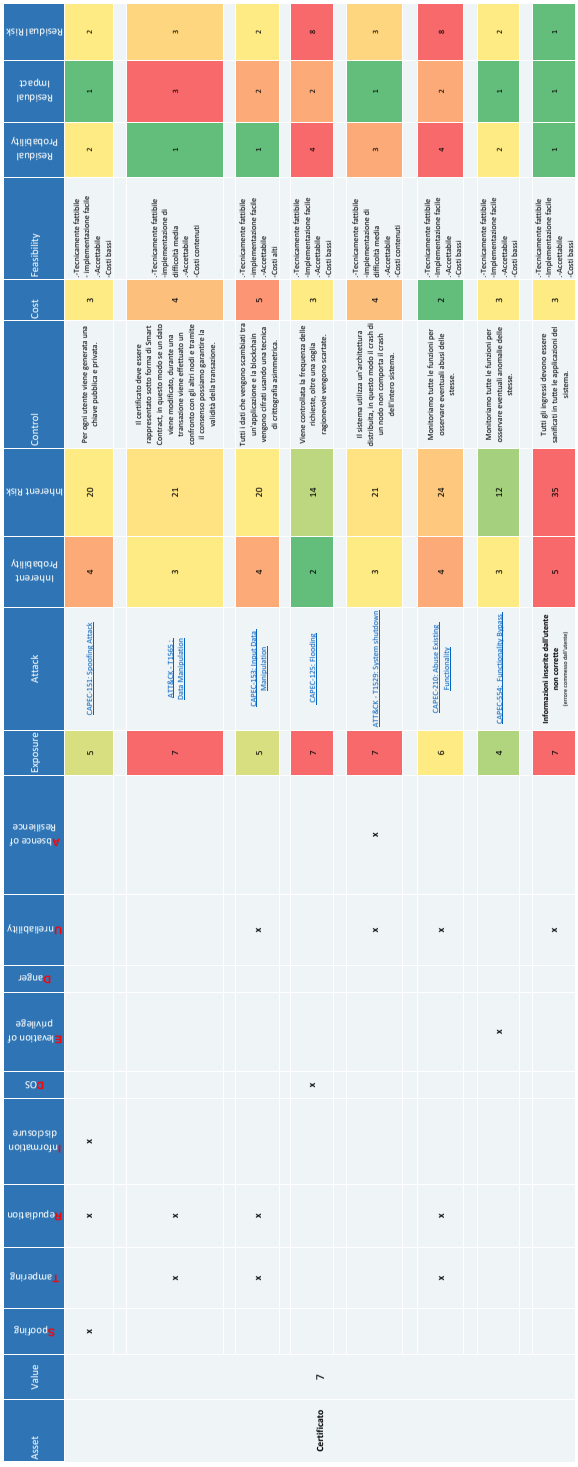
11 Modello STRIDE

Dopo aver elencato tutti i casi d'abuso e misuso relativi agli attaccanti e all'utente sbadato, si utilizza il modello STRIDE per andare ad identificare tutte le possibili minacce che potrebbero attaccare gli asset del nostro sistema. Si effettua quindi una valutazione del rischio per andare poi ad analizzare se vale la pena o meno mitigarlo.
Utilizzando quindi i cataloghi CAPEC e ATTACK siamo andati ad individuare tutti i possibili attacchi per ciascun asset, andando poi a valutare se convenisse o meno mitigarli, considerando la differenza tra il rischio inerente e quello residuo.

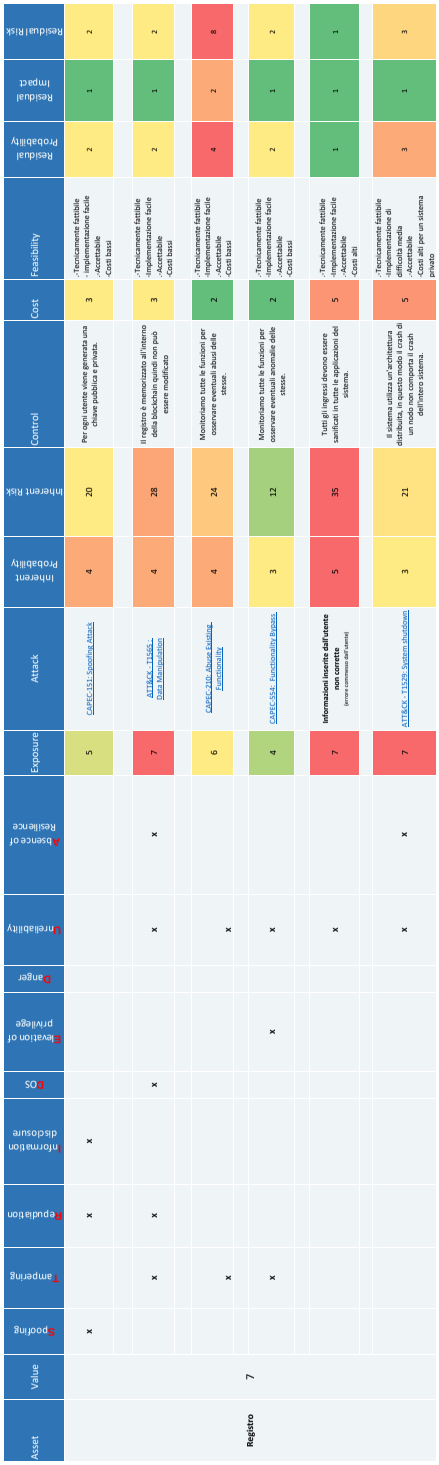
11.1 Scala qualitativa

Per il costo la scala va da 1 a 7:	Per la probabilità va da 1 a 5 dove:	Per il rischio la scala dei valori va da 1 a 35, i valori sono suddivisi nei seguenti intervalli :
1: molto basso	0%-20% : 1	1-10: basso
2: basso	20%-40% : 2	11-14: medio-basso
3: medio-basso	40%-60% : 3	15-22: medio
4: medio	60%-80% : 4	23-26: medio-alto
5: medio-alto	80%-100% : 5	27-35: alto
6: alto		
7: molto alto		

11.2 Certificato

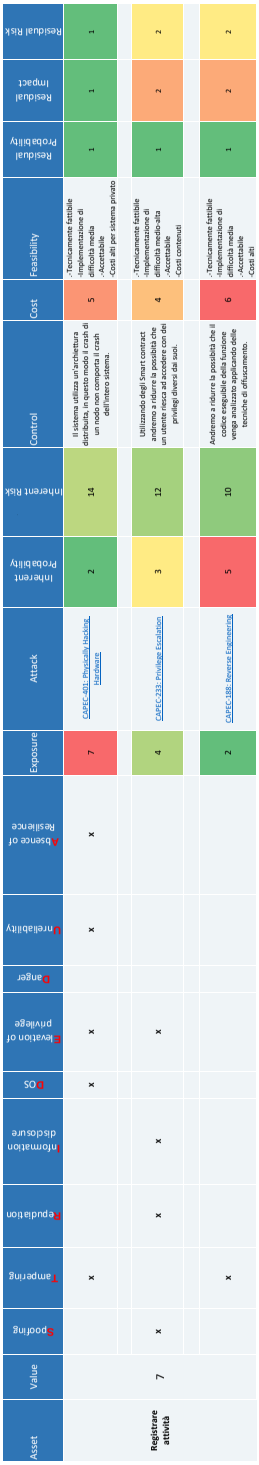


11.3 Registro



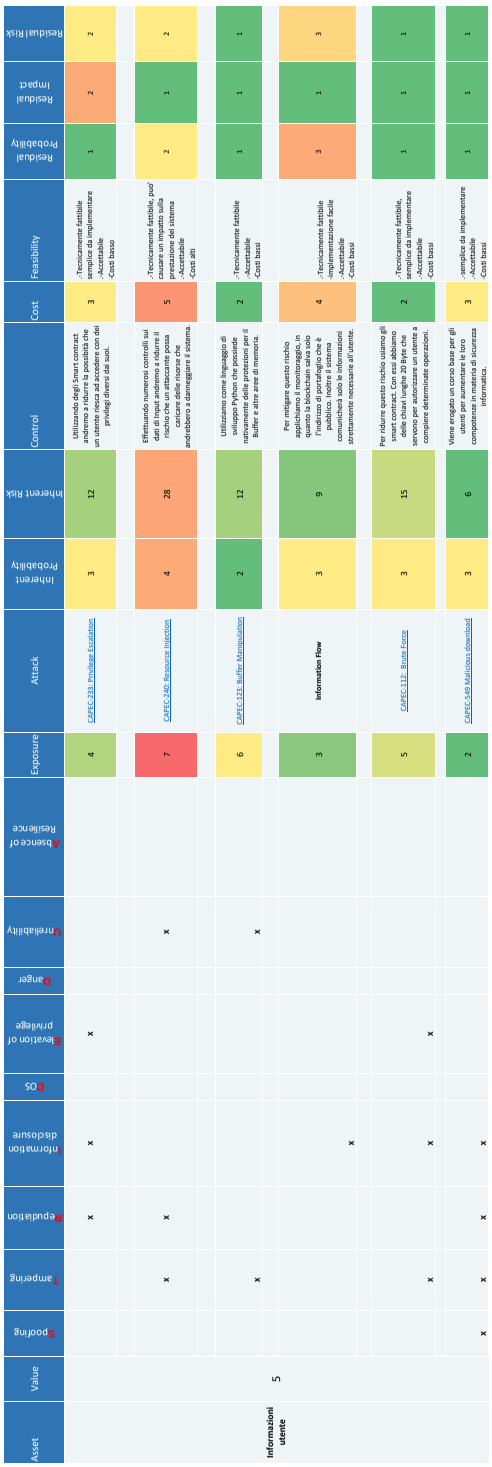
Asset	Value	Spoofing	Tampering	Repudiation	Information disclosure	DOS	Elevation of privilege	Danger	Unreliability	Absence of Resilience	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk
Registro	7	x	x	x	x						5	CAPEC-151: Spoofing Attack	4	20	Per ogni utente viene generata una chiave pubblica e privata.	3	-Tecnicamente fattibile -Implementazione facile -Acceptabile -Costi bassi	2	1	2
			x	x		x				x	7	ATT&CK - T1565 - Data Manipulation	4	28	Il registro è memorizzato all'interno della blockchain e non può essere modificato	3	-Tecnicamente fattibile -Implementazione facile -Acceptabile -Costi bassi	2	1	2
			x						x		6	CAPEC-270: Abuse Existing Functionality	4	24	Monitoriamo tutte le funzioni per osservare eventuali abusi delle stesse	2	-Tecnicamente fattibile -Implementazione facile -Acceptabile -Costi bassi	4	2	8
			x				x		x		4	CAPEC-554: Functionality Bypass	3	12	Monitoriamo tutte le funzioni per osservare eventuali anomalie delle stesse.	2	-Tecnicamente fattibile -Implementazione facile -Acceptabile -Costi bassi	2	1	2
									x		7	**Informazioni inserite dall'utente non corrette** (errori di coerenza dati utente)	5	35	Tutti gli ingressi devono essere sanificati in tutte le applicazioni del sistema.	5	-Tecnicamente fattibile -Implementazione facile -Acceptabile -Costi alti	1	1	1
					x				x	x	7	ATT&CK - T1529: System shutdown	3	21	Il sistema utilizza un'architettura distribuita, in questo modo il crash di un nodo non comporta il crash dell'intero sistema.	5	-Tecnicamente fattibile -Implementazione di difficoltà media -Acceptabile -Costi alti per un sistema privato	3	1	3

11.4 Registrare Attività



Asset	Value	Spoofing	Tampering	Repudiation	Information disclosure	DOS	Elevation of privilege	Danger	Unrelatability	Absence of Resilience	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk
Registrare attività	7		x			x	x		x	x	7	CAPEC-601: Physically Hijacking Hardware	2	14	Il sistema utilizza un'architettura distribuita, in questo modo il crash di un nodo non comporta il crash dell'intero sistema.	5	-Tecnicamente fattibile -Implementazione di difficoltà media -Accettabile -Costi alti per sistema privato	1	1	1
					x		x				4	CAPEC-233: Privilege Escalation	3	12	Utilizzando degli Smart contract andremo ridurre la possibilità che un utente riesca ad accedere con dei privilegi diversi dai suoi.	4	-Tecnicamente fattibile -Implementazione di difficoltà media-alta -Accettabile -Costi contenuti	1	2	2
			x								2	CAPEC-188: Reverse Engineering	5	10	Andremo a ridurre la possibilità che il codice sorgente venga decompilato se venga analizzato applicando delle tecniche di offuscamento.	6	-Tecnicamente fattibile -Implementazione di difficoltà media -Accettabile -Costi alti	1	2	2

11.5 Informazioni Utente



Asset	Value	Spoofing	Tampering	Repudiation	Information disclosure	DOS	Elevation of privilege	Danger	Unreliability	Absence of Resilience	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk
Informazioni utente	5			x	x		x				4	CAPEC-233: Privilege Escalation	3	12	Utilizzando degli Smart contract andremo a creare una regola che un utente riesca ad accedere con dei privilegi diversi dai suoi.	3	-Tecnicamente fattibile semplice da implementare -Accettabile -Costi bassi	1	2	2
			x	x	x				x		7	CAPEC-240: Resource Injection	4	28	Effettuando numerosi controlli sui dati da caricare andremo a ridurre il rischio che un attaccante possa caricare delle risorse che andrebbero a danneggiare il sistema.	5	-Tecnicamente fattibile, puo' causare un impatto sulla prestazione del sistema -Accettabile -Costi alti	2	1	2
			x	x	x				x		6	CAPEC-123: Buffer Manipulation	2	12	Utilizziamo come linguaggio di sviluppo Python che possiede nativamente delle protezioni per il Buffer e altre aree di memoria.	2	-Tecnicamente fattibile -Accettabile -Costi bassi	1	1	1
					x						3	Information Flow	3	9	Per mitigare questo rischio applicheremo il monitoraggio, in maniera tale da bloccare subito l'indirizzo di portafoglio che è pubblico. Inoltre il sistema comunicherà solo le informazioni strettamente necessarie all'utente.	4	-Tecnicamente fattibile -Implementazione facile -Acceptabile -Costi bassi	3	1	3
					x		x				5	CAPEC-112: Brute Force	3	15	Per ridurre questo rischio usiamo gli smart contract. Ci costringono delle chiavi lunghe 20 Byte che servono per autorizzare un utente a compiere determinate operazioni.	2	-Tecnicamente fattibile, semplice da implementare -Acceptable -Costi bassi	1	1	1
		x		x	x						2	CAPEC-549: Malicious Download	3	6	Viene eseguito un corso base per gli utenti per renderli edotti sulle competenze in materia di sicurezza informatica.	3	-semplice da implementare -Acceptable -Costi bassi	1	1	1

Al termine di questa fase, dopo le valutazioni dei rischi e delle tecniche di mitigazione da adottare, procederemo con la fase di implentazione del software vera e propria. Utilizzeremo la blockchain come registro per le transazioni e gli smart contract per lo sviluppo del progetto. Questa è la scelta più conveniente, secondo le nostre valutazioni, per andare a mitigare i rischi che abbiamo individuato, oltre ad applicare i principi di buona programmazione e una funzione di Login.

12 Implementazione

12.1 Linguaggi

Per l'implementazionde del nostro progetto abbiamo utilizzato due linguaggi:

- **Solidity** $\longrightarrow$ per scrivere gli Smart Contract che interagiscono con la blockchain per decidere se registrare o meno una transazione;
- **Python** $\longrightarrow$ per scrivere la parte che permette all'utente di interagire con la blockchain, richiamando le funzioni degli Smart Contract, e fornendogli un'interfaccia per facilitarne la fruizione.

12.2 Front end

Il primo livello, il front end, è l'interfaccia che gli utenti utilizzano per interagire con l'applicazione. Nel nostro progetto, abbiamo utilizzato Python, nello specifico la libreria PySimpleGUI. Lo scopo di questo livello è semplicemente quello di dare all'utente la possibilità di interagire, ricevere e inviare informazioni all'applicazione che sta utilizzando, in maniera semplice e veloce.

Abbiamo deciso di implementare un'interfaccia, affinchè per l'utente, l'utilizzo del nostro programma risulti il più semplice possibile, evitando (o se non altro riducendo) quindi la possibilità che l'utente commetta errori dovuti alla difficoltà dell'interazione con il programma (in quanto interagire con il programma da linea di comando per chi non esperto, potrebbe risultare difficoltoso).

12.2.1 Struttura del front end

Il front end e' suddiviso per ruoli, si ha: il consumatore, il produttore e il trasformatore; questo è stato fatto per facilitare l'adattamento dei vari componenti front end rendendoli indipendenti tra di loro. Le classi hanno alcuni metodi in comune: il costruttore della classe, il costruttore della finestra, CloseWindow per chiudere e terminare tutte le finestre attive e ListenEvent per osservare un'azione eseguita dall'utente.

```
front end
├── produttore.py
│   └── addMP(self)
├── consumatore.py
│   ├── VediP(self)
│   ├── VediLP(self)
│   ├── VediFP(self)
│   ├── infoP(self)
│   ├── acqP(self)
│   └── infoAP(self)
└── trasformatore.py
    ├── tutteMP(self)
    ├── lottiMP(self)
    ├── infoMP(self)
    ├── Acquista(self)
    ├── DettagliMP(self)
    └── AggProd(self)
```

Ogni metodo rappresenta una finestra del front end, la quale contiene un ciclo infinito per osservare le operazioni eseguite dall'utente.

12.2.2 Sanificazione e controllo

```
47 if not values['QUANTMP'].isdigit():
48     self.winAggiungeMat.Element('-Alert-').update("non e' un numero")
49 if not values['FPMP'].isdigit():
50     self.winAggiungeMat.Element('-Alert_1-').update("non e' un numero
          ")
```

Codice 1: esempio di un controllo del tipo di dato inserito nella classe del produttore

Per ogni finestra che richiede un inserimento di informazioni da parte di un utente vengono effettuate operazioni di controllo, da parte del programma, dei tipi di dato inseriti. Quanto appena detto può essere visto nell'esempio. Inoltre nel caso in cui insorga un errore, viene aperta una finestra di avviso riportante le informazioni riguardanti l'errore che possono essere utili per l'utente.
La sanificazione degli ingressi viene anche effettuata all'interno degli Smart Contract; infatti all'interno di tutte le funzioni presenti negli smart contract vengono effettuati controlli sul tipo di dato, sul valore del dato e sulla correttezza.

```
129 ..
130 try:
131     mat_prim = contract.tutti_MP_lotti()
132 except exceptions.SolidityError as error:
133     sg.Popup(str(error).replace('execution reverted:','Si
            verificato il seguente errore:'),keep_on_top=True,
            background_color="#1d8c3b",icon=self.icona)
134 ..
```

Codice 2: esempio della gestione dell'eccezione nella classe trasformatore

Per ogni metodo si ha una richiesta API verso l'esterno e vengono gestite l'eventuali eccezioni, che si possono avere in caso di errore generato esternamente.
Analogamente al controllo del tipo di dato, viene lanciato il messaggio dell'errore.

12.3 Backend

Questo secondo livello fa riferimento alla logica principale dell'applicazione. Il backend è correlato agli Smart Contract che vengono eseguiti sulla blockchain.
L'applicazione fruibile dall'utente, non farà nulla che non sia specificato negli Smart Contract.
Oltre a ciò, il backend è supportato dalle API (Application Programming Interface) e dalle funzionalità della blockchain.

12.3.1 Smart Contract

Mediante gli Smart Contract, si andranno a modellare le azioni che saranno rese possibili sulla blockchain.
(Nota che l'applicazione implementata poi non potrà eseguire nulla che non sia stato specificato negli Smart Contract).

Avevamo inizialmente optato per lo sviluppo di un unico Smart Contract, ma essendo parecchi i metodi che dovevamo implementare ed essendo anche molti gli attributi presenti, eravamo arrivati ad avere un ***"God object"***.
Essendo questo un problema, per la leggibilità e la manutenzione del software, abbiamo deciso poi in un secondo momento di suddividere lo Smart Contract "originario", in tre Smart Contract, uno per ogni ruolo. La difficoltà è stata, in questo caso, di istanziare le funzioni all'interno dei contratti in cui sono presenti le strutture su cui devono essere effettuati dei controlli ed eventualmente delle alterazioni, effettuando un controllo su fatto che chi richiede effettivamente l'esecuzione di tale operazione sia l'indirizzo associato a quel

contratto ed andando quindi a richiamare al loro interno la funzione istanziata nel contratto di interesse.
Si è inoltre deciso di non importare alcuna libreria per lo sviluppo in Solidity degli smart contract e di utilizzare solamente le funzionalità base integrate nel linguaggio, al fine di evitare l'introduzione di vulnerabilità nel software.
Per quanto riguarda il calcolo del footprint totale per un prodotto finito, si è deciso di sommare il footprint di ciascun lotto di materia prima utilizzata con la quantità di footprint prodotta nella produzione di un prodotto finito.

Smart Contract - Consumatore
Mediante questo Smart Contract, si va a creare un utente che come ruolo avrà quello del *CONSUMATORE*.
All'interno di questo Smart Contract definiamo una struttura che rappresenta il magazzino del consumatore, all'interno del quale andremo a salvare tutte le informazioni relative ai prodotti acquistati da parte di quest'ultimo.
Le funzioni implementate all'interno di questo Smart Contract sono:

1. Funzione che permette al consumatore di acquistare un prodotto inserito dal trasformatore. Se il prodotto che viene acquistato è già presente all'interno del magazzino del consumatore, si va semplicemente ad incrementarne la quantità, altrimenti il prodotto viene memorizzato in magazzino;

2. Funzione utilizzata per stampare le informazioni di un prodotto acquistato dal consumatore. L'esecuzione di tale funzione va a buon fine solo se a richederla è effettivamente il consumatore e se il prodotto di cui si vogliono visualizzare le informazioni è effettivamente presente nel magazzino del consumatore.

Smart Contract - Trasformatore
Mediante questo Smart Contract, si va a creare un utente che come ruolo avrà quello del *TRASFORMATORE*.
In questo Smart Contract sono presenti due strutture:

- la prima struttura viene utilizzata per salvare tutte le informazioni riguardanti un nuovo prodotto che viene inserito da parte del trasformatore;

- la seconda struttura viene invece utilizzata per contenere tutti gli acquisti di materie prime da parte del trasformatore.

Le funzioni presenti in questo Smart Contract sono:

1. Funzione che permette al trasformatore di acquistare una materia prima. Se tale materia prima è stata già stata acquistata in precedenza da parte del trasformatore, viene solamente incrementata la quantità di quest'ultima, altrimenti viene inserita nel magazzino del trasformatore;

2. Funzione che permette l'inserimento di prodotti finiti da parte del trasformatore. Tale funzione viene eseguita solo se il lotto di materia prima che viene inserito esiste all'interno del amgazzino del trasformatore, se le materie prime utilizzate effettivamente sono disponibili, nella quantità necessaria, nel magazzino del trasformatore e se la quantità presente in magazzino non è nulla;

3. Funzione che consente al consumatore di vedere tutti i lotti di un prdotto di cui si conosce il nome. La richiesta di esecuzione di tale funzione va a buon fine solo se richiamata dal consumatore;

4. Funzione che consente al consumatore di vedere tutti i lotti di tutti i prodotti. La richiesta di esecuzione di tale funzione va a buon fine solo se richiamata dal consumatore;

5. Funzione che ritorna il numero dei diversi prodotti esistenti;

6. Funzione utilizzata per stampare le informazioni di un prodotto inserito dal trasformatore. L'esecuzione di tale funzione va a buon fine solo se richiesta dal consumatore e solo se il prodotto esiste;

7. Funzione utilizzata per stampare le informazioni di una materia prima acquistata dal trasformatore. L'esecuzione di tale funzione va a buon fine solo se richiesta dal trasformatore e solo se il prodotto esiste;

8. Funzione che permette al consumatore di acquistare un prodotto finito. L'esecuzione di tale funzione va a buon fine solo se richiesta dal consumatore, se la quantità acquistata è un valore positivo e diverso da zero, se il prodotto effettivamente esiste e se la quantità che si desidera acquistare è effettivamente disponibile;

9. Funzione che consente di conoscere il footprint di un prodotto finito. L'esecuzione di tale funzione va a buon fine solo se richiesta dal consumatore.

10. Funzione che ci permette di trasformare un numero in una stringa.

Smart Contract - Produttore
Mediante questo Smart Contract, si va a creare un utente che come ruolo

avrà quello del *PRODUTTORE*.
In questo Smart Contract è presente una sola struttura che viene utilizzata per memorizzare le materie prime inserite dal produttore.
Le funzioni presenti in questo Smart Contract sono:

1. Funzione che consente l'inserimento di materie prime da parte del produttore. L'esecuzione di tale funzione va a buon fine solos se richiesta dal produttore e se sia la quantità che il footprint della materia inserita sono maggiori di zero;

2. Funzione che permette al trasformatore di acquistare materia prima. L'esecuzione di tale funzione va a buon fine solo se richiesta dal trasformatore, se la quantità che si desidera acquistare è maggiore di zero, se la materia prima è effettivamente disponibile e se la materia prima è disponibile nella quantità richiesta.

3. Funzione che consente di vedere tutti i lotti di una materia prima di cui si conosce il nome. L'esecuzione di tale funzione va a buon fine solos se richiesta dal trasformatore.

4. Funzione che consente di vedere tutti i lotti di tutte le materie prime. L'esecuzione di tale funzione va a buon fine solo se richiesta dal trasformatore.

5. Funzione che restituisce il numero di materie prime.

6. Funzione utilizzata per stampare le informazioni di una materia prima inserita da un produttore. L'esecuzione di tale funzione va a buon fine solo se richiesta dal trasformatore e se la materia prima esiste.

7. Funzione che permette al trasformatore di aggiungere un prodotto finito. L'esecuzione di tale funzione va a buon fine solo se richiesta dal trasformatore, se la quantità che si vuole inserire è un numero positivo diverso da zero e se il footprint indicato è un numero positivo diverso da zero.

12.3.2 SMT Checker

In questa sezione parleremo di SMT Checker, uno strumento fondamentale per verificare la correttezza del codice implementato.
Durante lo sviluppo del progetto, una volta terminati gli smart contract abbiamo utilizzato questo strumento per verificare la presenza di errori, sia di natura statica che dinamica. Infatti SMT Checker effettua come prima cosa un'analisi statica del codice, quindi controlla eventuali errori di casting

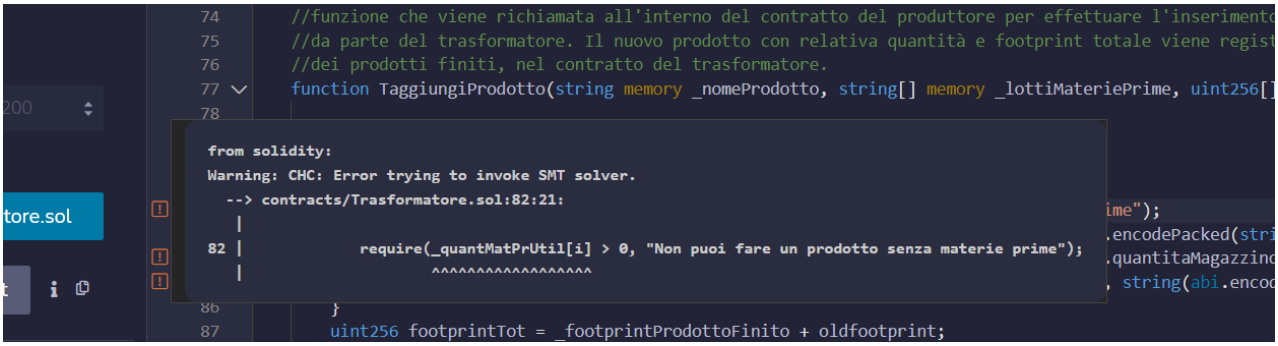
o scrittura, e successivamente effettua dei controlli sulle condizioni definite all'interno dei **Require** e degli **Assert**. Questo strumento utilizza molti teoremi matematici e tecniche di astrazione, quindi, senza scendere troppo nel dettaglio, il suo funzionamento consiste nell'astrarre il problema, per renderlo più generico, e poi viene negato, quindi si cerca una combinazione di valori che rende vero il problema negato, e che quindi non soddisfa il problema originale. Il vantaggio di questo approccio è che non bisogna enumerare tutte le soluzioni giuste, me ne basta una sbagliata per poter dire se le clausole sono corrette, oppure, mi dice se è presente una sequenza di valori delle variabili che può generare delle anomalie.
Abbiamo utilizzato SMT Checker su Remix dopo aver finito di scrivere gli smart contract. Abbiamo utilizzato questo strumento in questo modo.

```
92
93 pragma experimental SMTChecker;
```

Codice 3: Utilizzo di SMT Checker

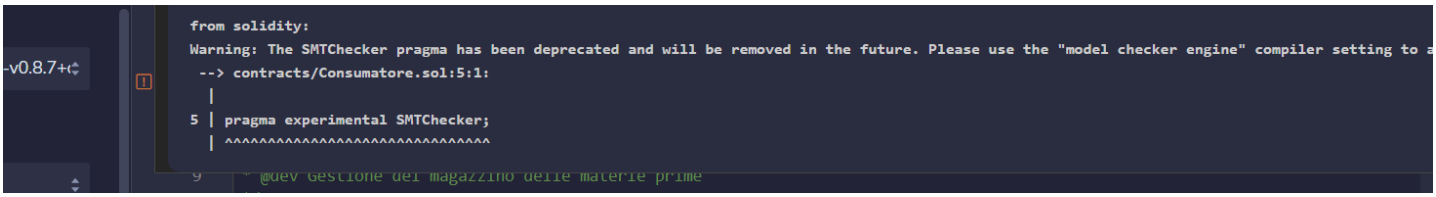
Questa modalità d'uso attualmente è deprecata, ma è ancora valida. Lo abbiamo fatto perchè era il modo più efficiente per utilizzare questo strumento, altri modi per utilizzarlo generavano problemi e anomalie. Questo è un esempio del risultato finale.

12.4 Esempio di falso positivo dovuto all'implementazione deprecata di SMT Checker



In base a quanto letto nella documentazione SMT checker effettua dei controlli sugli array che noi abbiamo implementato in maniera differente, questo genera dei falsi positivi. All'interno degli Smart Contract tutti i parametri che vengono passati alle funzioni sono controllati, non solo i tipi di dato primitivi, ma anche le stringhe e gli array. SMT Checker conferma che il codice degli smart contract è scritto correttamente e tutte le operazioni sulle variabili di stato non generano anomalie.

12.5 Esempio di falso positivo dovuto ad un array passato come argomento ad una funzione



12.5.1 Deploy degli Smart Contract

Uno Smart Contract deve essere distribuito per essere a disposizione degli utenti della rete. Per distribuire uno Smart Contract, si invia una transazione che contiene il codice dello Smart Contract compilato senza specificare alcun destinatario.
Prima di eseguire il depoly, gli Smart Contract devono essere compilati. Per fare ciò utilizziamo la funzione ***compile_source***, di py-solc-x (libreria che ci permette di interagire con il compilatore solc).

```
15 def compile_source_file(file_path):
16     with open(file_path, 'r') as f:
17         source = f.read()
18 
19         return compile_source(source, output_values=['abi', 'bin'])
```

Codice 4: istruzione per compilazione del contratto nel init.py

Non chiamiamo direttamente la funzione, ma questa viene invocata da una altra funzione da noi definita che prende come parametro il file contenente il codice sorgente, come mostrato nelle linee di codice sopra riportate.

```
24 compiled_sol = compile_source_file(data_path)
25 
26 # recupera l'interfaccia del contratto
27 contract_id, contract_interface = compiled_sol.popitem()
28 
29 # prende il bytecode/bin
30 bytecode = contract_interface['bin']
31 
32 # prende l'abi
33 abi = contract_interface['abi']
```

Codice 5: istruzione per recuperazione del bytecode e l'abi del contratto nel init.py

Procediamo quindi, come si vede sopra, compilando lo Smart Contract, ed andando a recuperare (una volta compilato), l'abi e il bytecode dello Smart Contract.

```
78 Consumatore = w3.eth.contract(abi=abi, bytecode=bytecode)
79 
80 tx_hash = Consumatore.constructor(consum).transact({'from': admin})
81 #Aspetta che avvenga la transazione, e prende la "ricevuta" di quest'
       ultima
82 tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
83 
84 consumatore = w3.eth.contract(
85     address=tx_receipt['contractAddress'],
86     abi=abi
```

```
87 )
```

Codice 6: istruzione per deployment final del contratto nel init.py

Nella riga 78 viene creato un oggetto contratto a partire dall'abi e dal bytecode che gli vengono passati come parametri e che altro non sono che l'abi e il bytecode recuperati nel passo prima. Nella riga 80 invece si effettua una transazione, che va a richiamare il metodo costruttore dell'oggetto contratto appena creato, una volta che questa transazione è terminata, come visibile nella riga 84 viene recuperato l'indirizzo del contratto. Infine viene eseguito il deploy.

Nota che qui è mostrata la procedura per il consumatore, ma si procede nella stessa maniera anche per il trasformatore e per il produttore.

12.5.2 API

Abbiamo creato una classe Contract che mappa le funzioni di tutti e tre gli Smart Contract. Nei nostri smart contract abbiamo due tipi di funzioni:

- quelle che devono registrare una transazione sulla Blockchain, come si puo vedere nel codice sotto, nel quale si richiama la funzione dello Smart Contract e si esegue la transazione, si aspetta di avere la ricevuta della transazione e si stampa lo status (se è andata a buon fine(1) o meno(0)), che è anche il valore di ritorno della funzione;

```
59 def inserisci_MP(nomeMP,quantMP,footprint):
60     tx_hash = produttore.functions.aggiungiMateriaPrima(nomeMP,
           quantMP, footprint).transact({'from': current_user})
61     tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
62     print(tx_receipt['status'])
63     return (tx_receipt['status'])
```

Codice 7: interfaccia API per l'inserimento di una materia prima nel contract.py

- quelle che devono non registrare una transazione sulla Blockchain, come si puo vedere nel codice sotto, si richiama la funzione dello Smart Contract direttamente nel return per avere il risultato (lo utiliziamo per le funzioni di visualizzazione).

```
92 def lotti_Prod(nomeP):
93     return trasformatore.functions.vediLottiProdotto(nomeP).call({'
           from': current_user})
```

Codice 8: interfaccia API per recuperare l'informazione del lotto prodotto nel contract.py

Nota che qui è mostrata una funzione per tipo, ma si procede nella stessa maniera anche per tutte le altre funzioni.

13 Testing

Arriviamo adesso a parlare della fase di testing del progetto, una fase molto importante che ne certifica l'efficienza e il funzionamento. Per scrivere i test e' stato utilizzato il framework 'unittest', ispirato a JUnit, che ci ha permesso di scrivere i test utilizzando il linguaggio python. Essi sono stati inseriti all'interno della cartella 'test'; inoltre, sono stati suddivisi in base al tipo di utente che si andava a testare.
Per eseguire i test bisogna prima istanziare il contratto, inoltre non devono essere eseguite altre operazioni sulla blockchain, in quanto cambierebbero i valori restituiti da essa; infatti i test sono stati pensati per lavorare su dati fittizi precedentemente inseriti.
Dopo aver istanziato lo smart contract bisogna eseguire i test in maniera sequenziale, in quanto al loro interno ci sono, come per un DB, dei seeder, cioè dei metodi che popolano la blockchain con dati fittizi.

```
92  class Seeder_consumatore:
93
94      contratto.current_user = contratto.consum
95      contratto.w3.geth.personal.unlock_account(contratto.account[2], '
            consumatore')
96      __acquisto01: int
97      __acquisto02: int
98
99      # Qui vengono chiamati i metodi del contratto che andranno a
            valorizzare gli attributi della classe
100
101     def __init__(self,prod01,prod02,quan01):
102         try:
103             self.__acquisto01 = contratto.acquista_Prod(prod01,quan01
                    )
104         except exceptions.SolidityError as error:
105             self.__acquisto01 = 0
106         try:
107             self.__acquisto02 = contratto.acquista_Prod(prod02,100) #
                    input non valido, verr  generata un'eccezione
108         except exceptions.SolidityError as error:
109             self.__acquisto02 = 0
110
111     def getAcquisto01(self):
112         return self.__acquisto01
```

Codice 9: Esempio di classe seeder utilizzata per implementare i test

I seeder sono tre, uno per ogni tipo di utente; sono implementati come una classe, dove all'interno del costruttore vengono eseguite tutte le operazioni di inserimento e acquisto sulla blockchain; inoltre, i dati restituiti da essa

vengono inseriti all'interno degli attributi della classe.
I test sono cinque, tre di questi test creano un oggetto di tipo seeder, successivamente si utilizza il metodo 'assertEquals', a cui passiamo due parametri, uno è il metodo getter dell'istanza del seeder, il quale restituisce il valore dell'attributo corrispondente, mentre l'altro è il valore atteso. Per avere dei test efficienti sono state effettuate più operazioni dello stesso tipo; inoltre alcuni metodi che eseguono operazioni sulla blockchain sono stati volontariamente sbagliati, per verificare l'affidabilità dei controlli inseriti. Tutte le chiamate alla blockchain vengono eseguite dentro blocchi try-except, in questo modo i metodi con parametri sbagliati non causano il blocco dei test; quando questi metodi vengono eseguiti l'attributo corrispondente assume un valore diverso da quello degli altri, e si controlla nel file di test che abbia assunto il valore giusto.

```
92 seeder = Seeder_produttore('materia01','materia02','materia03','
       materia04','materia05')
93         self.assertEquals(seeder.getMateria01(),1)
94         self.assertEquals(seeder.getMateria02(),1)
95         self.assertEquals(seeder.getMateria03(),1)
96         self.assertEquals(seeder.getMateria04(),0)
97         self.assertEquals(seeder.getMateria05(),1)
```

Codice 10: Classe che implementa i test utilizzando assertEquals

Gli altri due file vanno a controllare tutte le operazioni di visualizzazione, in questi due non sono presenti dei seeder, si interroga direttamente la blockchain e con il metodo assertEquals si controllano i valori restituiti e quelli attesi.
Se si vogliono eseguire i test, dopo la prima volta, sarà necessario istanziare nuovamente lo smart contract, in quanto lo stato del precedente smart contract è già stato modificato, quindi i lotti dei successivi prodotti saranno diversi da quelli specificati dentro i test.

13.1 Esempio di esito positivo