

UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Elettronica
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione



Progetto di Software Security and Blockchain

Professore

Luca Spalazzi

Studenti

Rahmi El Mechri
Rebecca Giuliani
Chiara Gobbi
Alice Moretti
Arianna Ronci

ANNO ACCADEMICO 2022-2023

Indice

1	Introduction	1
1.1	Le Blockchain	2
2	Requirement Engineering	5
2.1	Requirement Analysis	6
2.2	Preliminary Risk Assessment	9
2.3	Risk Identification	9
2.3.1	Asset identification	9
2.4	Risk Analysis	15
2.4.1	Asset Value Assessment & Exposure Assessment	15
2.4.2	Threat Identification	17
2.5	Risk Decomposition	20
2.5.1	Attack Assessment	20
2.6	Risk Reduction	47
2.6.1	Control Identification	47
2.6.2	Feasibility Assessment	47
2.6.3	Security Requirements Definition	47
3	Software Design	53
3.1	Design assets	54
3.2	Architectural design	54
4	Implementation	57
4.1	Local Database	59
4.2	Authentication	60
4.3	Load Balancing	61
4.4	User's Actions Flow	63
4.4.1	Registration and Log in	63
4.4.2	Once Logged In	65
4.5	Testing	68

4.5.1	Dynamic testing	68
4.5.2	Static Testing	68

A Appendice: Evoluzione dell'On-Chain Manager

71

1

Introduction

Indice

1.1 Le Blockchain	2
-----------------------------	---

1.1 Le Blockchain

Oggi si fa un gran parlare della tecnologia Blockchain.

Nell'attuale momento storico questa innovazione sta catalizzando moltissime attenzioni.

La Blockchain (letteralmente "catena di blocchi") sfrutta le caratteristiche di una rete informatica di nodi e consente di gestire e aggiornare, in modo univoco e sicuro, un registro contenente dati e informazioni in maniera aperta, condivisa e distribuita senza la necessità di un'entità centrale di controllo e verifica. Una Blockchain consiste in un registro condiviso e immutabile che serve per registrare le transazioni, tenere traccia degli asset e consolidare un rapporto di fiducia.

I possibili requisiti di una blockchain sono:

1. Decentralizzazione (requisito che si riferisce alla questione di fiducia reciproca);
2. Sicurezza (requisito che si riferisce alla protezione dalla possibilità che qualcuno modifichi le informazioni);
3. Scalabilità.

Le reti decentralizzate possono fornire solo due dei tre vantaggi rispetto a decentralizzazione, sicurezza e scalabilità: si parla di Blockchain Trilemma.

Aldilà di tali attributi è di interesse prendere in considerazione quale sia l'impatto ambientale di una Blockchain: le Blockchain possono avere grossi problemi di consumo energetico.

Per quanto riguarda gli algoritmi del consenso, con il passaggio a Proof of stake, si sta dimostrando come grazie a tale algoritmo sia possibile tornare a dei consumi di energia accettabile, ma resta da provare che passando a questo algoritmo non si perda in termini di sicurezza.

Inoltre, quando si parla di impatto, occorre considerare l'aspetto di archiviazione dei dati: in un registro distribuito le informazioni sono replicate poiché ogni nodo mantiene una copia completa della Blockchain. Quindi, i dati sono ridondati n volte dove n è il numero di full node.

Sono davvero necessarie queste n repliche? Si tratta di migliaia di copie, ognuna delle quali porta con sé un impatto ambientale non indifferente che ne consegue anche in considerazione del fatto che si tratta di dispositivi il cui smaltimento non è banale.

Quindi oltre all'algoritmo del consenso, un ulteriore punto su cui si potrebbe andare a lavorare per aumentare la sostenibilità delle Blockchain è la gestione dello storage.

A tale proposito, il progetto si basa sull'idea di recuperare il concetto di sharding o di partizionamento orizzontale, tipicamente usato nei database: data una tabella questa si può partizionare in due frammenti che possono essere memorizzati uno in un nodo e l'altro in un altro nodo. In questo modo, non è necessario memorizzare l'intera tabella in un solo nodo.

L'applicazione della tecnica di sharding alla tecnologia Blockchain determina che ognuno dei nodi mantenga una copia locale di solo alcuni dei blocchi: il consumo di spazio sarebbe così abbattuto.

2

Requirement Engineering

Indice

2.1 Requirement Analysis	6
2.2 Preliminary Risk Assessment	9
2.3 Risk Identification	9
2.4 Risk Analysis	15
2.5 Risk Decomposition	20
2.6 Risk Reduction	47

2.1 Requirement Analysis

I requisiti di un sistema sono la descrizione dei servizi che il sistema deve fornire e dei suoi vincoli operativi. Sono caratteristiche che riflettono l'esigenza del cliente di avere un sistema che aiuti a risolvere alcuni problemi.

Il processo di ricerca, analisi, documentazione e verifica di questi servizi e vincoli è chiamato Ingegneria dei Requisiti (Requisite Engineering – RE).

In questa fase, l'obiettivo è individuare quali siano i requisiti funzionali, ossia i requisiti che riguardano le funzionalità e i servizi che il sistema deve fornire, e i requisiti non funzionali, ossia i requisiti che si configurano come vincoli sulle funzioni e/o sui servizi offerti, permettendo di descrivere "come" il sistema si comporta, svolge i suoi compiti.

La notazione che in tale progetto è stata adottata per la descrizione dei requisiti è i*; si tratta di un linguaggio di modellazione adatto per la fase iniziale dello sviluppo che permette di ragionare sull'ambiente organizzativo, sui sistemi informativi e sugli attori eterogenei che si inseriscono in tale contesto, esaminando i loro obiettivi, spesso in competizione, al fine di comprendere il dominio del problema.

Gli attori rilevanti sono:

- User;
- Off-Chain Manager;
- On-Chain Manager;
- Shard, che in particolare è stato modellato come Ruolo.

L'attore *User* rappresenta colui che utilizza il sistema e ha come obiettivo principale quello di eseguire delle transazioni.

L'attore *Off-Chain Manager* è quella parte del sistema che viene installato nel dispositivo di ogni utente e che si interfaccia con la parte del sistema che si trova nella Blockchain.

L'attore *On-Chain Manager* è quella parte del sistema che si trova nella Blockchain e si occupa della gestione dello stato degli Shards.

Il ruolo *Shard* è quella parte del sistema che rappresnta un frammento della Blockchain. Congiuntamente tutti gli "individui" che ricorprono tale ruolo costituiscono la Blockchain nella sua interezza.

La prima versione di diagramma i* è quella riportata nella Figura 2.1.

In esso sono stati messi in evidenza i principali obiettivi e task di ogni attore e le dipendenze strategiche

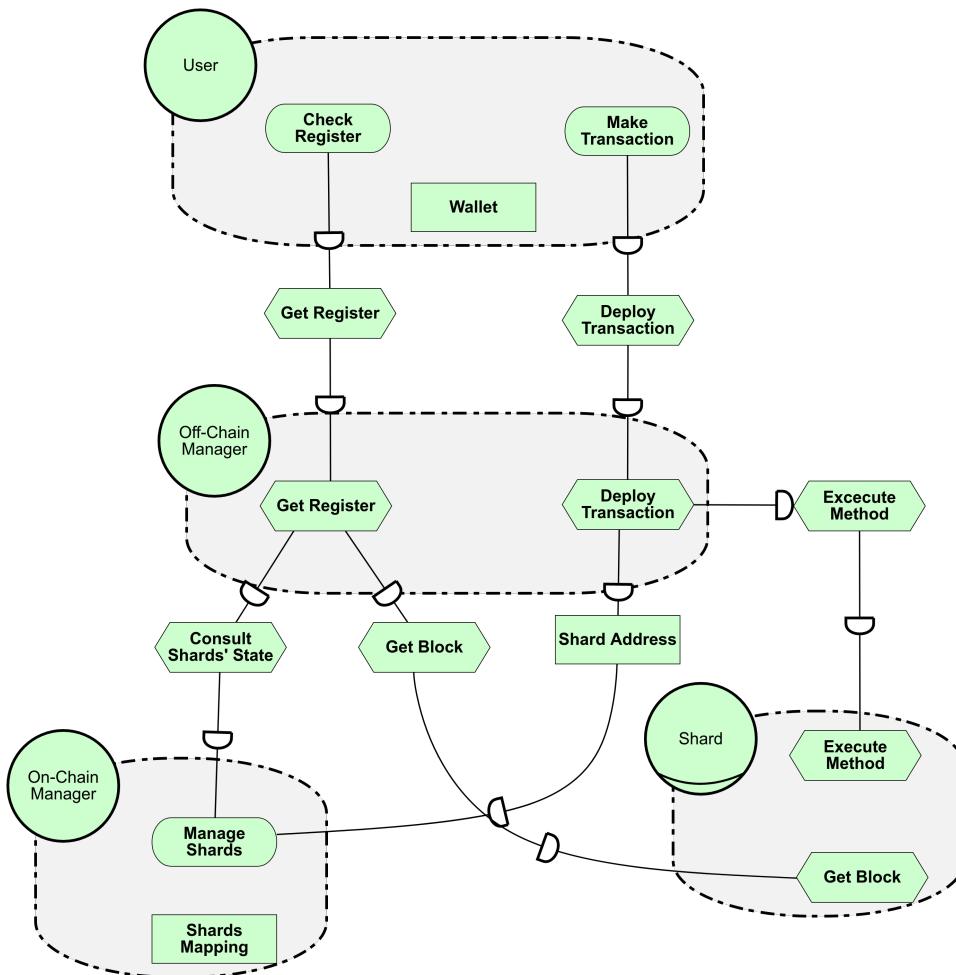
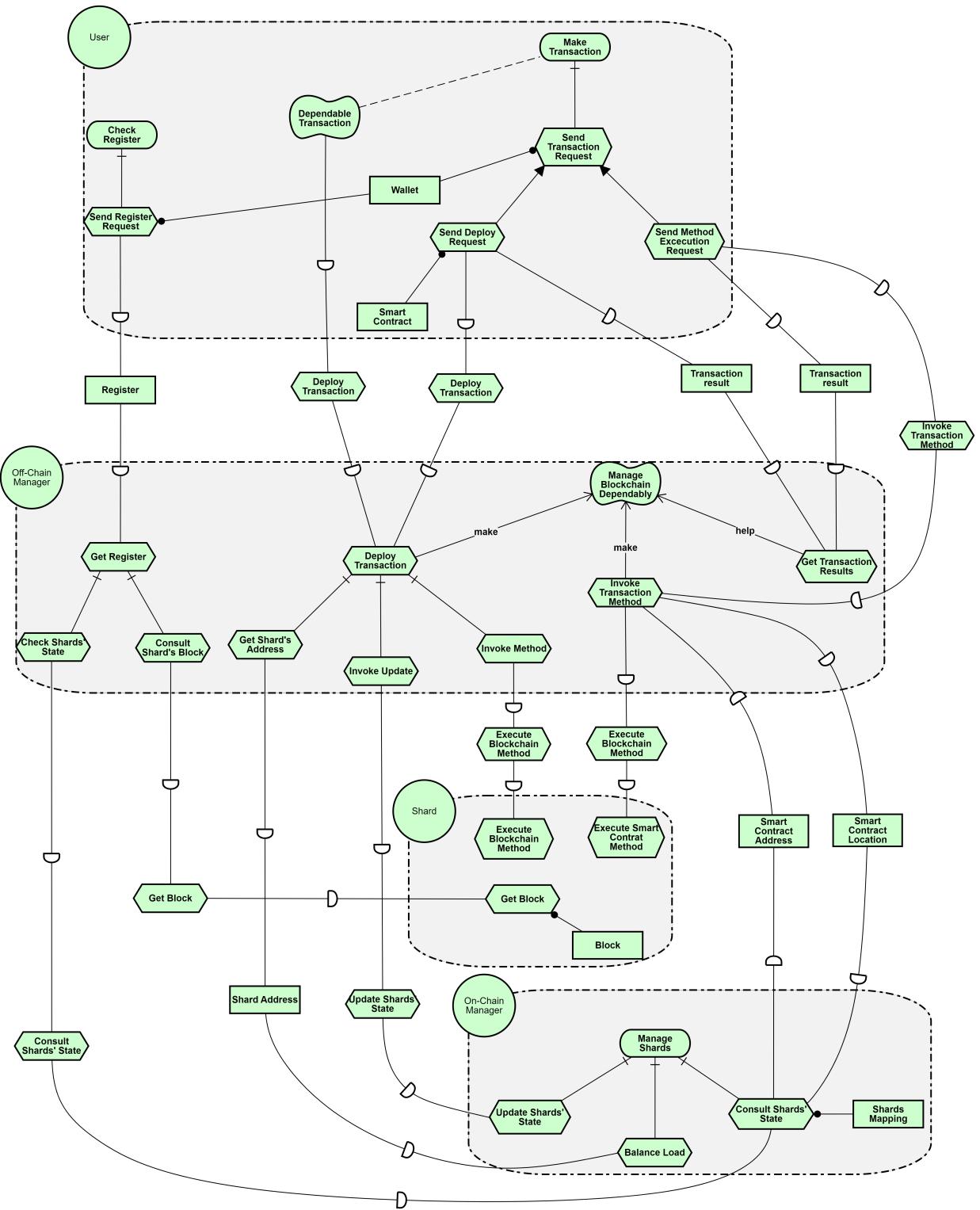


Figura 2.1: Prima versione del diagramma i*

che legano gli uni agli altri.

Il passo successivo è stato quello di decomporre, a partire da tale visione di alto livello, ciascuna entità in sotto-entità, aggiungendo via via un maggiore livello di dettaglio sino alla versione finale del diagramma. In particolare, gli *hardgoal* sono stati decomposti per il tramite di And-Decomposition e Means-end Decomposition e i task sono stati decomposti per il tramite di Task-Decomposition.



2.2 Preliminary Risk Assessment

Sin dalla fase iniziale dell'Ingegneria dei requisiti è importante iniziare a ragionare su quali possano essere i rischi a cui il sistema che si sta sviluppando, potrebbe essere esposto poichè commettere errori nella fase analisi dei requisiti è molto costoso, in quanto tipicamente per essere corretti necessitano di ripercorrere il lavoro eseguito dalla fase in cui è stato scoperto l'errore fino alla fase iniziale di analisi dei requisiti, riprogettare e implementare la situazione non presa in considerazione; questo vale soprattutto per gli errori legati alla sicurezza che sono spesso i più costosi e dannosi.

Adottando quindi l'approccio di specifica dei requisiti basata sulla gestione del rischio, si sono svolte le seguenti attività:

1. Risk Identification
2. Risk Analysis
3. Risk Decomposition
4. Risk Reduction

2.3 Risk Identification

La fase di Risk Identification è una fase fondamentale nel processo di gestione del rischio. Il primo passo è quello di identificare gli asset che potrebbero richiedere una protezione, dato il valore che essi rappresentano. Per ciascuno di essi occorre stabilire gli obiettivi funzionali e in particolare gli obiettivi di sicurezza e la politica di sicurezza da applicare.

2.3.1 Asset identification

Di seguito si riportano gli asset individuati e per ciascuno di essi i rispettivi obiettivi da rispettare e le politiche di utilizzo.

Wallet

L'asset *Wallet* identifica il portafoglio dell'utente ed è ciò che permette all'utente di autenticarsi, gestire le proprie criptovalute ed effettuare transazioni sulla Blockchain.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Authenticity*

- SO2 - *Authorization*
- SO3 - *Accountability*
- SO4 - *Confidentiality*

Le regole di utilizzo dell'asset (policy) sono:

- OP1 - Il wallet deve essere utilizzato dagli utenti per identificarsi univocamente ed eseguire solo ed esclusivamente quelle operazioni sulla Blockchain che gli sono consentite.
- OP2 - Il wallet deve permettere di tenere traccia delle operazioni svolte da ogni utente.
- OP3 - Le informazioni del Wallet sono consultabili solo da parte dell'utente proprietario della risorsa.

Smart Contract

L'asset *Smart Contract* identifica la risorsa che viene "deployata" sulla Blockchain e che permette di effettuare una transazione specifica.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Authenticity*
- SO2 - *Integrity*
- SO3 - *Authorization*

Le regole di utilizzo dell'asset (policy) sono:

- OP1 - Solo l'utente può richiedere il deploy di uno Smart Contract precedentemente da lui compilato.
- OP2 - Una volta deployato, lo Smart Contract non potrà essere modificato o eliminato da nessun utente.
- OP3 - Una volta deployato, solo un utente autorizzato potrà eseguire una transazione che richiede l'invocazione di metodi dello Smart Contract.

Register

L'asset *Register* identifica il registro in cui vengono salvate tutte le transazioni che vengono effettuate sulla Blockchain.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Authenticity*
- SO2 - *Integrity*

Le regole di utilizzo dell'asset (policy) sono:

- OP1 - Il registro non deve poter essere modificato o danneggiato da nessun utente.
- OP2 - Il registro non deve essere difforme in qualche maniera da ciò che è memorizzato sulla Blockchain.

Transaction Result

L'asset *Transaction Result* identifica il responso che viene mostrato all'utente in merito all'esito di una transazione.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Authenticity*
- SO2 - *Integrity*

Le regole di utilizzo dell'asset (policy) sono:

- OP1 - Il risultato di una transazione non deve essere difforme in qualche maniera da quello che è memorizzato sulla Blockchain.
- OP3 - Il risultato di una transazione non deve essere modificato o danneggiato.

Smart Contract Location

L'asset *Smart Contract Location* identifica la locazione e quindi lo Shard in cui è stato salvato lo Smart Contract che contiene il metodo della transazione che si vuole eseguire.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Authenticity*
- SO2 - *Integrity*
- SO3 - *Confidentiality*

Le regole di utilizzo dell'asset (policy) sono:

- OP1 - La locazione di uno Smart Contract sulla Blockchain deve essere disponibile quando l'Off-Chain Manager richiede tale informazione.
- OP2 - La locazione di uno Smart Contract sulla Blockchain non deve essere difforme in qualche maniera dallo stato della Blockchain.
- OP3 - La locazione di uno Smart Contract sulla Blockchain non deve essere modificata o danneggiata in alcun modo da nessuno.

Smart Contract Address

L'asset *Smart Contract Address* identifica l'indirizzo dello Smart Contract in cui è salvato il metodo della transazione che si vuole invocare.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Authenticity*
- SO2 - *Integrity*
- SO3 - *Confidentiality*

Le regole di utilizzo dell'asset (policy) sono:

- OP1 - L'informazione che viene inviata dall'Off-Chain Manager all'On-Chain Manager non deve essere modificata o danneggiata.
- OP2 - L'informazione deve rimanere privata.

Deploy Transaction

L'asset *Deploy Transaction* identifica il task dell'Off-Chain Manager di deployare uno Smart Contract sulla Blockchain. Questo task è scomponibile in tre sotto-task: ottenere l'indirizzo dello Shard su cui effettuare il deploy, invocare il metodo della Blockchain e aggiornare lo stato degli Shard.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Integrity*
- SO2 - *Reliability*
- SO3 - *Confidentiality*

Le regole di utilizzo dell'asset (policy) sono:

- OP1 - L'operazione di deploy di una transazione può essere eseguita solo se l'utente che ha fatto la richiesta è identificabile attraverso un Wallet.
- OP2 - Il deploy di una transazione deve essere eseguito in modo affidabile e corretto secondo la procedura prevista dal sistema.
- OP3 - Il deploy di una transazione deve essere conosciuto solo dall'utente che ha effettuato tale richiesta.

Update Shards' State

Il task dell'On-Chain Manager relativo all'aggiornamento dello stato degli Shard, ogni qualvolta si effetta un nuovo deploy, viene identificato dall'asset *Update Shards' State*.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Availability*
- SO2 - *Integrity*
- SO3 - *Reliability*
- SO4 - *Confidentiality*

Le regole di utilizzo dell'asset (policy) sono:

- OP1 - La richiesta di aggiornamento degli stati degli Shard all'interno dell'On-Chain Manager può essere effettuata solo dall'Off-Chain Manager.
- OP2 - Deve essere sempre possibile eseguire l'aggiornamento degli stati degli Shard.
- OP3 - La richiesta inviata all'On-Chain manager non deve essere alterata.

Shards' mapping

Shard's mapping è l'asset che rappresenta il mapping dello stato degli Shard ed è fondamentale per risalire agli Smart Contract deployati sulla Blockchain.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Authenticity*
- SO2 - *Integrity*

Le regole di utilizzo dell'asset (policy) sono:

- OP1 - Le informazioni del mapping devono essere integre: il mapping può essere modificato (ma le informazioni già presenti non possono mai essere distrutte) solo in seguito ad una richiesta da parte dell'Off-Chain Manager.
- OP2 - Il mapping non deve essere difforme in nessuna maniera dallo stato degli Shards.

Shard's address

L'asset *Shard's address* identifica l'indirizzo dello Shard in cui deve essere deployato il nuovo Smart Contract affinchè il balancing sia verificato.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Authenticity*
- SO2 - *Integrity*

Le regole di utilizzo dell'asset (policy) sono:

- OP1 - L'informazione fornita all'Off-Chain Manager deve essere non modificabile e non difforme in qualche maniera dallo stato della Blockchain.

Block

L'asset *Block* identifica l'insieme di blocchi contenuti in uno Shard. Questa risorsa viene restituita per ricostruire il registro, quando l'utente lo richiede.

Gli obiettivi che l'asset deve rispettare sono:

- SO1 - *Authenticity*

- SO2 - *Integrity*
- SO3 - *Assurance*

Le regole di utilizzo dell'asset (policy) sono :

- OP1 - L'informazione contenuta in un blocco deve essere non modificabile.
- OP2 - IL blocco deve contenere le transazioni che gli utenti hanno indirizzato verso la Blockchain e che i vari nodi hanno precedentemente memorizzato nella propria mempool.
- OP3 - Bisogna avere la garanzia che prima o poi un blocco deve essere finalizzato.

2.4 Risk Analysis

La fase di Risk Analysis è il processo di identificazione e analisi di potenziali eventi futuri che potrebbero avere un impatto negativo sul sistema. Tale analisi viene eseguita per comprendere meglio cosa può accadere, le implicazioni finanziarie del verificarsi di tale evento e quali misure si possono adottare per mitigare o eliminare tale rischio.

2.4.1 Asset Value Assessment & Exposure Assessment

Per ogni asset identificato nella fase precedente viene valutato il valore patrimoniale, ossia il valore che l'asset ha per l'organizzazione (*Asset Value Assessment*).

Inoltre, vengono valutate le potenziali perdite associate a ciascun asset, nel caso in cui un attacco abbia successo. Ciò vuol dire valutare l'esposizione, ossia identificare i danni che l'organizzazione potrebbe subire nel caso in cui gli obiettivi di dependability prestabiliti non venissero soddisfatti (*Exposure Assessment*).

Nel nostro caso tale valutazione è basata su una quantificazione in termini monetari.

Nella seguente tabella sono presenti tre colonne: *Asset* specifica gli asset identificati, *Value* specifica il valore individuato per ciascuno di essi e *Exposure* specifica cosa comporterebbe se uno degli obiettivi non venisse soddisfatto.

Asset	Value	Exposure
Wallet	Il wallet è richiesto perchè necessario all'utente per interagire con la blockchain.	Nel caso di furto di chiavi legate ad un wallet, l'attaccante potrebbe sfruttare il balance legato a quel wallet per spendere Ether compiendo operazioni sulla blockchain con conseguente perdita dell'autenticità delle operazioni compiute. L'anonimato sarebbe comunque garantito perchè non sarebbe possibile risalire alle informazioni dell'utente a partire dal suo wallet.
Smart Contract	Uno Smart Contract permette agli utenti di eseguire più operazioni sulla blockchain tramite l'invocazione dei suoi metodi.	Nel caso di Smart Contract contenenti codice malevolo, un attaccante potrebbe fare il deploy di uno Smart Contract che contiene codice malevolo provocando danno agli utenti che invocano i suoi metodi.
Register, Shards' Mapping, Transaction Result	Poichè ciò che viene restituito all'utente è il risultato di transazioni eseguite a valle di algoritmi di consenso, questi asset garantiscono la fiducia tra gli utenti essendo una terza parte tra essi. Lo Shard's Mapping viene ritornato dall'On-Chain all'Off-Chain in modo tale che quest'ultimo possa, tramite mirate interrogazioni alla blockchain, ricostruire il registro.	Nel caso in cui la comunicazione tra BlockChain e Off-Chain Manager o tra On-Chain e Off-Chain Manager non sia protetta, il risultato dell'interazione potrebbe essere non integro. Nel caso in cui le infomazioni siano integre, ci potrebbero comunque essere problemi di autenticità nel momento in cui, durante il meccanismo di consenso, la maggior parte dei nodi sia stato corrotto da un attaccante. Questo potrebbe portare ad una dimunzione dell'utenza della blockchain.
Smart Contract Location, Smart Contract Method	Tali informazioni servono rispettivamente all'On Chain Manager e all'Off Chain Manager per poter indirizzare una richiesta dell'utente sul corretto shard.	Nel caso in cui l'informazione sul metodo da invocare o sulla posizione dello Smart Contract nello shard sia non autentica, si rischia di far eseguire all'utente un metodo scorretto o non esistente compromettendo l'availability del servizio o una perdita di Ether non voluta sul balance dell'account associato. Questo potrebbe portare ad una dimunzione dell'utenza della blockchain.
Deploy Transaction, Update Shards' State, Shard's Address	Per completare il deploy di una transazione, l'Off-Chain Manager ha bisogno di sapere l'address dello shard su cui effettuare il deploy. Tale informazione viene restituita dall'On-Chain Manager che, sulla base dello Shards' Mapping determina quale degli shard della blockchain è quello su cui effettuare il deploy per garantire il bilanciamento del carico. L'On-Chain Manager deve aggiornare lo Shards' Mapping ogni volta che viene effettuata un'operazione sulla blockchain.	Nel caso in cui alcune di queste informazioni siano compromesse, l'ottimizzazione della memoria verrebbe meno e ci sarebbe una distribuzione non bilanciata delle richieste con conseguente calo delle prestazioni.
Block	Un blocco contiene una serie di transazioni su cui è stato precedentemente raggiunto il consenso	L'integrità del blocco è garantita grazie ad una struttura dati detta Merkle Tree tramite la quale è garantito un meccanismo a prova di compromissione delle transazioni in esso contenute. Nel caso in cui un attaccante possedesse uno stake meggiore del 33\% dello stake degli altri nodi, si potrebbe non avere la garanzia che un blocco prima o poi venga finalizzato.

2.4.2 Threat Identification

L'identificazione delle minacce, di fatto, coincide con la violazione degli obiettivi. **STRIDE** è un diffuso modello di valutazione che permette di identificare minacce e vulnerabilità di un certo prodotto o di una certa applicazione.

Il modello STRIDE comprende le seguenti categorie di minacce: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service e Elevation of Privilege.

Tale modello è strettamente legato agli obiettivi di security, sebbene sia opportuno avere una visione più ampia che comprenda gli obiettivi di dependability. A tal proposito, esiste un modello STRIDE modificato e esteso, che prende il nome di **DUAI-STRIDE** che contiene tutte le minacce del modello precedente, alle quali si aggiungono Danger, Unreliability e Absence of Resilience.

Al termine di questa fase sono state compilate le prime colonne della tabella STRIDE (Figure 2.2 e 2.3). In particolare, per ciascun asset è stata inserita una X ogni volta che è stata individuata una minaccia che comportava la violazione di un requisito di sicurezza.

Asset	Value	Spoofing	Tampering	Repudiation	Information disclosure	DOS	Elevation of privilege	Danger	Unreliability	Absence of Resilience	Exposure
Wallet	200.000-500.000	X			X						10.000-20.000
Shards' Mapping	300.000-500.000	X	X			X					20.000-60.000
Shard's Address	300.000-500.000	X	X			X					100.000-150.000
Smart contract	10.000-50.000	X	X			X					5.000-10.000
Register	50.000-200.000	X	X								20.000-40.000
Transaction Result	5.000-10.000	X	X								1.000-2.500

Figura 2.2: Tabella STRIDE - prima parte

Asset	Value	S poofing	T ampering	R epudiation	I nformation disclosure	D oS	E levation of privilege	D anger	U nreliability	A bsence of Resilience	Exposure
Smart Contract Location	50.000-100.000	X	X			X					10.000-20.000
Smart Contract Address	50.000-100.000	X	X			X					25.000-50.000
Deploy Transaction	200.000-500.000		X	X	X				X		50.000-100.000
						X					10.000-20.000
					X				X		50.000-100.000
Update Shards' State	250.000-500.000						X				80.000-150.000
							X		X		250.000-500.000
Block	150.000-300.000		X	X							100.000-150.000
						X					50.000-100.000

Figura 2.3: Tabella STRIDE - seconda parte

2.5 Risk Decomposition

La fase di Risk Decomposition si riferisce al processo di scomposizione del rischio in più componenti o fattori, al fine di analizzare e comprendere meglio la natura del rischio e di identificare le azioni di mitigazione più efficaci. Questo processo può implicare la valutazione di vari elementi, come il tipo di minaccia, la vulnerabilità del sistema, il potenziale impatto su persone, risorse o processi aziendali e le misure preventive in atto. In sostanza, la Risk Decomposition aiuta a individuare le componenti del rischio e a valutarle separatamente per ottenere una comprensione più approfondita della situazione.

2.5.1 Attack Assessment

Per valutare gli attacchi e quindi il modo in cui una minaccia può realizzarsi, è stato necessario ipotizzare degli scenari e valutare per ognuno di essi ciò che potrebbe accadere. Tali scenari vengono chiamati *casi d'uso*.

Per esaminare gli attacchi è opportuno immaginare degli scenari illegittimi, ovvero degli scenari dove le cose vanno male perché qualcuno sta usando in maniera impropria il sistema, accidentalmente o malintenzionatamente. In questo caso si parla di:

- *casi di misuso* (o casi di uso improprio): quando si indica un'azione sbagliata che viene svolta accidentalmente;
- *casi di abuso*: quando si indica un'azione sbagliata che viene svolta intenzionalmente.

In generale i casi di uso, abuso e misuso vengono descritti tramite delle tabelle che sono chiamate *Schemi di Jacobson*.

Nelle Figure 2.4 e 2.5 si riportano i due scenari impropri relativi alla risorsa Wallet.

Il primo caso di abuso, denominato *Wallet Key Theft*, è relativo al furto di identità, ovvero una situazione nella quale l'attaccante, dopo alcuni tentativi, riesce ad ottenere i dati di un altro utente e utilizza tale identità per agire in maniera fraudolenta.

Anche il secondo caso d'abuso rappresenta uno scenario in cui un utente malevolo riesce ad autenticarsi con un'identità altrui, ma a seguito di azioni come *key logging*, ovvero una tecnica che intercetta e cattura segretamente tutto ciò che viene digitato sulla tastiera senza che l'utente se ne accorga, o *phishing*, ovvero una truffa effettuata su Internet attraverso la quale un malintenzionato cerca di ingannare la vittima a fornire delle informazioni personali. Questo caso di abuso è stato denominato *User's Identity Violation*.

Case Type	Abuse Case	Case ID	AT-01-01
Case Name	Wallet Key Theft		
Actors	User, Attacker		
Description	Dopo un certo numero di tentativi l'attaccante riesce ad indovinare le credenziali di un altro utente che gli permettono di compiere delle azioni illegittime.		
Data	Wallet		
Stimulus and preconditions	Il sistema si basa su un meccanismo di autenticazione a singolo fattore e non prevede un meccanismo di password throttling. L'attaccante ha a disposizione un numero illimitato di tentativi.		
Attack Flow 1	L'attaccante sfruttando la conoscenza di tutti i possibili caratteri utilizzabili esegue n tentativi inserendo una delle possibili combinazioni fino a trovare la chiave che permette l'accesso.		
Response and Postconditions	L'attaccante può accedere al sistema con l'identità di un altro utente.		
Non Functional Requirements			
Mitigation	<ul style="list-style-type: none"> • Limitazione del numero dei tentativi. • Suggerire all'utente di cambiare la password ogni tre mesi. 		
Comments			

Figura 2.4: Wallet Abuse Case - Wallet Key Theft

Case Type	Abuse Case	Case ID	AT-01-02
Case Name	User's Identity Violation		
Actors	User, Attacker		
Description	L'utente tramite il suo wallet deve autenticarsi per poter inviare richieste all'Off-Chain Manager. Il caso di abuso di tale risorsa si riferisce alla possibilità che un attaccante riesca ad ottenere le credenziali di un altro utente per compiere azioni improprie.		
Data	Wallet		
Stimulus and preconditions	L'attaccante riesce a rubare o ottenere per altre vie le credenziali di un altro utente. Ad esempio seguendo l'Attack Flow di Wallet Key Theft.		
Attack Flow 1	L'attaccante ottiene un insieme di credenziali, ad esempio tramite tecniche di key logging o phishing e determina le eventuali policy del sistema di autenticazione per determinare quali delle credenziali rispettano i criteri specificati.		
Response and Postconditions	L'attaccante riesce ad autenticarsi e sfrutta l'identità dell'utente per compiere atti illegittimi.		
Non Functional Requirements			
Mitigation	<ul style="list-style-type: none"> • Autenticazione a due fattori. 		
Comments			

Figura 2.5: Wallet Abuse Case - User's Identity Violation

Il caso di abuso della risorsa *Smart Contract* (Figura 2.6), denominato *Code Interception & Modification*, si riferisce alla possibilità che il codice di uno Smart Contract venga intercettato e modificato, poiché non è stato previsto un efficiente meccanismo di protezione del canale di comunicazione tra l'Off-Chain Manager e uno Shard.

Case Type	Abuse Case	Case ID	AT-02-01
Case Name	Code Interception & Modification		
Actors	User, Attacker, Off-Chain Manager, Shard		
Description	Gli Smart Contract implementati dagli utenti, per essere essere deployati nella Blockchain, vengono spediti dall'utente all'Off-Chain Manager e dall'Off-Chain Manager ad uno degli Shard. Il caso di abuso di tale risorsa si riferisce alla possibilità che il codice dello Smart Contract venga compromesso prima di essere deployato sullo Shard.		
Data	Smart Contract		
Stimulus and preconditions	Affinché l'attacco venga messo in atto, l'attaccante deve possedere i mezzi per modificare il codice di uno Smart Contract. Non è stato previsto un efficiente meccanismo di protezione del canale di comunicazione tra Off-Chain Manager e Shard.		
Attack Flow 1	L'utente implementa uno Smart Contract e ne richiede il deploy all'Off-Chain Manager. L'Off-Chain Manager elabora la richiesta e invoca i metodi dell'interfaccia Web3 per effettuare il deploy dello Smart Contract in uno degli Shard. L'attaccante nel momento in cui lo Smart Contract è trasferito dall'Off-Chain Manager allo Shard, modifica il contenuto dei metodi dello Smart Contract.		
Response and Postconditions	L'utente potrebbe subire dei danni nel momento di interazione con lo Smart Contract il cui codice è stato alterato.		
Non Functional Requirements			
Mitigations	<ul style="list-style-type: none"> • Firma digitale dei dati. • Ridondanza di dati. 		
Comments	Per tale caso d'uso ci si è rifatti alla voce del catalogo CWE-ID 345 (Insufficient Verification of Data Authenticity)		

Figura 2.6: Smart Contract Abuse Case - Code Interception & Modification

Il caso di abuso della risorsa *Register* (Figura 2.7), denominato *Register Interception & Modification*, si riferisce alla possibilità che un attaccante intercetti e modifichi il contenuto del registro richiesto da un utente durante la comunicazione tra Off-Chain Manager e On-Chain Manager e tra l'Off-Chain Manager

e gli Shard.

Case Type	Abuse Case	Case ID	AT-03-01
Case Name	Register Interception & Modification		
Actors	User, Off-Chain Manager, On-Chain Manager, Attacker, Shard		
Description	Quando l'utente vuole ottenere le informazioni del registro distribuito memorizzate all'interno degli Shard della Blockchain, si interfaccia con l'Off-Chain Manager per ottenere tale risorsa. L'Off-Chain Manager, a sua volta, comunica con l'On-Chain Manager per consultare lo Shard's Mapping e quindi avere le conoscenze necessarie per ricostruire il registro. Il caso di abuso che riguarda tale risorsa, si riferisce alla possibilità che l'informazione scambiata tra On-Chain Manager e Off-Chain Manager e/o l'informazione scambiata tra Off-Chain Manager e Shard sia compromessa.		
Data	Register		
Stimulus and preconditions	L'Off-Chain Manager richiede all'On-Chain Manager di visualizzare lo Shard's Mapping e agli Shard il contenuto dei blocchi. L'Off-Chain Manager, sfruttando le informazioni ottenute, ricostruisce il registro. Affinchè l'attacco possa essere messo in atto, l'attaccante deve riuscire ad intercettare la risorsa registro durante la comunicazione tra l'Off-Chain Manager e l'On-Chain Manager e tra l'Off-Chain Manager e gli Shard, inoltre non deve avere previsto un meccanismo di protezione adeguato contro la modifica del contenuto inviato.		
Attack Flow 1	Mentre lo Shard's Mapping è in transito tra l'On-Chain Manager e l'Off-Chain Manager e mentre il contenuto dei blocchi è in transito tra gli Shard e l'Off-Chain Manager l'attaccante ne modifica il contenuto rendendolo diverso da quello originale, così da compromettere la ricostruzione del registro.		
Response and Postconditions	L'utente riceve un registro con contenuto differente rispetto a quello memorizzato sugli Shard.		
Non Functional Requirements			
Mitigations	<ul style="list-style-type: none"> • Firma digitale dei dati. • Ridondanza di dati. 		
Comments	Per tale caso d'uso ci si è rifatti alla voce del catalogo CWE-ID 345 (Insufficient Verification of Data Authenticity)		

Figura 2.7: Register Abuse Case - Register Interception & Modification

È possibile che si verifichi uno scenario illegittimo nel momento in cui un utente richiede una transazione e l'Off-Chain Manager gli restituisce il risultato poiché un attaccante potrebbe intercettare e modificare tale informazione durante la comunicazione tra l'Off-Chain Manager e uno Shard. Tale caso di abuso, denominato *Transaction Result Interception & Modification*, è riportato nella Figura 2.8.

Case Type	Abuse Case	Case ID	AT-04-01
Case Name	Transaction Result Interception & Modification		
Actors	User, Off-Chain Manager, Attacker, Shard		
Description	L'Off-Chain Manager restituisce il risultato di una transazione all'utente che l'ha richiesta dopo aver comunicato con uno degli Shard. Il caso di abuso si riferisce alla possibilità che tale informazione venga compromessa.		
Data	Transaction Result		
Stimulus and preconditions	Non è stato previsto previsto un meccanismo di protezione adeguato contro la possibilità di modificare il contenuto del risultato della transazione durante la comunicazione tra l'Off-Chain Manager e uno Shard. Affinché l'attacco possa essere messo in atto, l'attaccante deve possedere i mezzi per alterare i dati relativi al risultato della transazione.		
Attack Flow 1	L'attaccante intercetta la comunicazione tra Off-Chain Manager e Shard e ne modifica il contenuto del risultato della transazione.		
Response and Postconditions	L'utente riceve come risultato della transazione un risultato differente da quello originale.		
Non Functional Requirements			
Mitigations	<ul style="list-style-type: none"> • Firma digitale dei dati. • Ridondanza di dati. 		
Comments	Per tale caso d'uso ci si è rifatti alla voce del catalogo CWE-ID 345 (Insufficient Verification of Data Authenticity)		

Figura 2.8: Transaction Result Abuse Case - Transaction Result Interception & Modification

Affinché l'Off-Chain Manager possa eseguire una transazione, a seguito dell'invocazione di un metodo dello Smart Contract, deve conoscere la locazione dello Smart Contract. Un attaccante potrebbe intercettare e modificare tale locazione se riuscisse ad intercettare la comunicazione che avviene tra l'On-Chain Manager e l'Off-Chain Manager. Tale caso di abuso, denominato *Smart Contract Location Interception & Modification*, è riportato in Figura 2.9.

Case Type	Abuse Case	Case ID	AT-05-01
Case Name	Smart Contract Location Interception & Modification		
Actors	Off-Chain Manager, On-Chain Manager, Attacker		
Description	L'Off-Chain Manager ha bisogno di conoscere la locazione dello Smart Contract che implementa il metodo da invocare per eseguire una transazione che gli è stata richiesta. Tale risorsa viene restituita dall'On-Chain Manager all'Off-Chain Manager. Il caso di abuso di tale risorsa si riferisce alla possibilità che la locazione dello Smart Contract che l'On-Chain Manager invia venga compromessa da un attaccante.		
Data	Smart Contract Location		
Stimulus and preconditions	L'Off-Chain Manager richiede all'On-Chain Manager la locazione dello Smart Contract. Affinchè l'attacco possa avvenire, l'attaccante deve essere in grado di intercettare la risorsa target che viene trasferita e il target non deve aver previsto una protezione adeguata contro la possibilità di modificare il contenuto inviato.		
Attack Flow 1	Mentre la risorsa in questione viene trasferita dall'On-Chain Manager all'Off-Chain Manager l'attore malintenzionato intercetta e modifica il contenuto rendendolo qualcosa di differente dal contenuto originale che l'On-Chain Manager ha inviato.		
Response and Postconditions	L'Off-Chain Manager riceve una locazione differente dalla locazione dello Smart Contract che effettivamente implementa il metodo che l'Off-Chain Manager vuole invocare. Ciò implica che la richiesta di invocazione del metodo da parte dell'Off-Chain Manager non va a buon fine, poiché la locazione che gli viene restituita non corrisponde a quella in cui si trova effettivamente implementato il metodo da invocare.		
Non Functional Requirements			
Mitigations	<ul style="list-style-type: none"> • Firma digitale dei dati. • Ridondanza di dati. 		
Comments	Per tale caso d'uso ci si è rifatti alla voce del catalogo CWE-ID 345 (Insufficient Verification of Data Authenticity)		

Figura 2.9: Smart Contract Location Abuse Case - Smart Contract Location Interception & Modification

Per invocare un metodo di uno Smart Contract, l'Off-Chain Manager comunica all'On-Chain Manager l'indirizzo dello Smart Contract. Il caso di abuso (Figura 2.10), denominato *Smart Contract Address Interception & Modification*, che riguarda la risorsa Smart Contract Address, si riferisce alla possibilità

che un attaccante riesca ad intercettare e modificare l'informazione che viene scambiata tra l'Off-Chain Manager e l'On-Chain Manager.

Case Type	Abuse Case	Case ID	AT-06-01
Case Name	Smart Contract Address Interception & Modification		
Actors	Off-Chain Manager, On-Chain Manager, Attacker		
Description	L'Off-Chain Manager comunica all'On-Chain Manager l'indirizzo di uno Smart Contract precedentemente deployato, che contiene il metodo che si vuole invocare, al fine di conoscere la locazione per poter inviare la richiesta di esecuzione del metodo allo Shard corretto. Il caso di abuso che riguarda tale risorsa, si riferisce alla possibilità che l'informazione scambiata tra Off-Chain Manager e On-Chain Manager sia compromessa dall'attaccante.		
Data	Smart Contract Address		
Stimulus and preconditions	L'Off-Chain Manager trasmette all'On-Chain Manager l'indirizzo di uno Smart Contract, precedentemente deployato, di cui vuole conoscere la locazione. Affinchè l'attacco possa essere messo in atto, l'attaccante deve riuscire ad intercettare l'indirizzo trasmesso durante la comunicazione tra le due parti e l'Off-Chain Manager non deve avere previsto un meccanismo di protezione adeguato contro la modifica del contenuto inviato.		
Attack Flow 1	Mentre la risorsa è trasferita tra l'Off-Chain Manager e l'On-Chain Manager, l'attaccante intercetta la risorsa e ne modifica il contenuto rendendolo diverso da quello originale.		
Response and Postconditions	L'On-Chain Manager riceve un indirizzo differente rispetto a quello previsto. Se l'informazione alterata corrisponde all'indirizzo di un altro Smart Contract presente, potrebbe essere invocato un metodo non richiesto oppure inesistente.		
Non Functional Requirements			
Mitigations	<ul style="list-style-type: none"> • Firma digitale dei dati. • Ridondanza di dati. 		
Comments	Per tale caso d'uso ci si è rifatti alla voce del catalogo CWE-ID 345 (Insufficient Verification of Data Authenticity)		

Figura 2.10: Smart Contract Address Abuse Case - Smart Contract Address Interception & Modification

Quando viene fatto il deploy di una transazione, un attaccante potrebbe riuscire ad intromettersi nella comunicazione tra Off-Chain Manager e On-Chain Manager o nella comunicazione tra Off-Chain Manager e Shard. Se ciò accadesse, egli potrebbe apportare delle modifiche ai dati da lui osservati. Tale caso di abuso, denominato *Transaction Interception & Modification*, è riportato in Figura 2.11.

Inoltre, un utente che non ha scopi malevoli, ma che è semplicemente disattento, potrebbe inviare più

volte la stessa richiesta non vedendo arrivare una risposta dopo il primo invio. Tale caso di misuso, denominato *Involuntary Request Repetition*, è riportato in Figura 2.12.

Case Type	Abuse Case	Case ID	AT-07-01
Case Name	Transaction Interception & Modification		
Actors	On-Chain Manager, Off-Chain Manager, Shard, Attacker		
Description	L'attaccante si intromette nella comunicazione tra i due componenti con lo scopo di ottenere e modificare i dati trasmessi.		
Data	Deploy Transaction		
Stimulus and preconditions	L'attaccante è in grado di individuare il meccanismo di comunicazione tra le due parti. Non è previsto un meccanismo di mutuo-riconoscimento delle due componenti coinvolte. La comunicazione avviene in chiaro e non è previsto un meccanismo efficace di protezione dei dati scambiati.		
Attack Flow 1	L'attaccante è in grado di determinare il meccanismo di comunicazione tra Off-Chain Manager e On-Chain Manager inserendosi nel canale di comunicazione fingendosi un proxy di routing. L'attaccante filtra, modifica, osserva i dati che fluiscono tra le due parti con l'intento di manipolare le azioni che le due componenti avevano come scopo.		
Attack Flow 2	L'attaccante è in grado di determinare il meccanismo di comunicazione tra Off-Chain Manager e Shard inserendosi nel canale di comunicazione fingendosi un proxy di routing. L'attaccante filtra, modifica, osserva i dati che fluiscono tra le due parti con l'intento di manipolare le azioni che le due componenti avevano come scopo.		
Response and Postconditions	Violazione dell'integrità dei dati qualora l'attaccante modifichi gli stessi prima che vengano inviati al destinatario.		
Non Functional Requirements			
Mitigations	<ul style="list-style-type: none"> • Firma digitale dei dati. • Ridondanza di dati. 		
Comments			

Figura 2.11: Deploy Transaction Abuse Case - Transaction Interception & Modification

Case Type	Misuse Case	Case ID	MA-07-01
Case Name	Involuntary Request Repetition		
Actors	Off-Chain Manager, Clumsy User		
Description	L'utente disattento invia più volte la stessa richiesta, senza accorgersi degli invii precedenti.		
Data	Deploy Transaction		
Stimulus and Precondition	-		
Attack Flow 1	L'utente disattento con la volontà di inviare una richiesta, non vedendo arrivare una risposta, la invia più volte.		
Response and Postconditions	L'utente disattento sperpera inutilmente le sue risorse, ripetendo più volte la stessa richiesta, che più volte va a buon fine.		
Non functional Requirements			
Mitigations	<ul style="list-style-type: none"> • Limitare il numero di richieste che un utente può inviare in un'arco di tempo determinato. 		
Comments			

Figura 2.12: Deploy Transaction Misuse Case - Involuntary Request Repetition

Nelle Figure 2.13-2.15 si riportano i tre scenari impropri relativi a *Update Shard State*.

Nel primo caso d'abuso, denominato *Update Shard's State Request Interception & Modification*, si evidenzia come un attaccante potrebbe modificare i dati prima che essi arrivino al destinatario, se riuscisse ad intromettersi nella comunicazione tra Off-Chain Manager e On-Chain Manager.

Case Type	Abuse Case	Case ID	AT-08-01
Case Name	Update Shard's State Request Interception & Modification		
Actors	On-Chain Manager, Off-Chain Manager, Attacker		
Description	L'attaccante si intromette nella comunicazione tra i due componenti con lo scopo di ottenere e modificare i dati trasmessi.		
Data	Update Shards' State		
Stimulus and preconditions	L'attaccante è in grado di individuare il meccanismo di comunicazione tra le due parti. Non è previsto un meccanismo di mutuo-riconoscimento delle due componenti coinvolte. La comunicazione avviene in chiaro e non è previsto un meccanismo efficace di protezione dei dati scambiati.		
Attack Flow 1	L'attaccante è in grado di determinare il meccanismo di comunicazione tra Off-Chain Manager e On-Chain Manager inserendosi nel canale di comunicazione fingendosi un proxy di routing. L'attaccante filtra, modifica, osserva i dati che fluiscono tra le due parti con l'intento di manipolare le azioni che le due componenti avevano come scopo.		
Response and Postconditions	Violazione dell'integrità dei dati qualora l'attaccante modifichi gli stessi prima che vengano inviati al destinatario.		
Non Functional Requirements			
Mitigations	<ul style="list-style-type: none"> • Firma digitale dei dati. • Ridondanza di dati. 		
Comments			

Figura 2.13: Update Shards' State Abuse Case - Update Shards's State Request Interception & Modification

Il secondo caso d'abuso, denominato *DoS Aimed Request*, rappresenta la possibilità che un attaccante, inviando richieste in maniera ripetuta e continuativa, riesca a rendere alcuni servizi dell'On-Chain Manager non disponibili o riesca ad impedire l'accesso alla risorsa ad altri utenti.

Case Type	Abuse Case	Case ID	AT-08-02
Case Name	DoS Aimed Requests		
Actors	On-Chain Manager, Off-Chain Manager, Attacker		
Description	L'attaccante invia richieste in modo continuativo con lo scopo di occupare l'On-Chain Manager e rendere i suoi servizi non disponibili.		
Data	Update Shards' State		
Stimulus and preconditions	Non sono stati previsti opportuni meccanismi di controllo per prevenire l'attacco.		
Attack Flow 1	L'attaccante, tramite la sua interfaccia Off-Chain Manager, invia un numero di richieste tali da occupare l'On-Chain Manager e rendere alcuni dei suoi servizi non disponibili.		
Attack Flow 2	L'attaccante, tramite la sua interfaccia Off-Chain Manager, invia richieste alla risorsa nel tempo il cui numero non supera il normale carico gestibile, ma pensate in modo tale da mantenere la risorsa impegnata il più a lungo possibile così da negare/limitare l'accesso alla risorsa da parte di altri utenti.		
Response and Postconditions	L'On-Chain Manager non è in grado di processare l'intera mole di richieste ricevute e come conseguenza diretta il mapping non viene aggiornato, per cui non rispecchia lo stato corrente degli shard.		
Non Functional Requirements			
Mitigation	<ul style="list-style-type: none"> • Progettare un algoritmo che non permetta di un lock sostenuto ad un piccolo gruppo di utenti. 		
Comments	Per rendere efficace l'Attack Flow 1 si presuppone che più attaccanti inviano richieste contemporaneamente al fine di occupare la risorsa. La differenza tra i due Attack Flow è che nel secondo il sistema non viene allertato.		

Figura 2.14: Update Shards' State Abuse Case - DoS Aimed Request

Il terzo caso d'abuso, denominato *Unauthorized Communication with On-Chain Manager*, rappresenta uno scenario nel quale un attaccante stabilisca una comunicazione con l'On-Chain Manager, inviando delle richieste di transazione che non sono autorizzate all'Off-Chain Manager.

Case Type	Abuse Case	Case ID	AT-08-03
Case Name	Unauthorized Communication with On-Chain Manager		
Actors	On-Chain Manager, Attacker		
Description	L'attaccante bypassando l'Off-Chain Manager e utilizzando la piattaforma Web3 comunica direttamente con l'On-Chain Manager, inviando la richiesta di una transazione non autorizzata.		
Data	Update Shards' State		
Stimulus and preconditions	Non è prevista una modalità di riconoscimento tra istanze di Off-Chain Manager e On-Chain Manager.		
Attack Flow 1	L'attaccante comunica in modo diretto con l'On-Chain Manager inviando delle richieste di transazione non autorizzate dall'Off-Chain Manager.		
Response and Postconditions	Violazione dell'integrità del Mapping, in quanto vengono salvate delle transazioni non autorizzate.		
Non Functional Requirements			
Mitigations			
Comments			

Figura 2.15: Update Shards' State Abuse Case - Unauthorized Communication with On-Chain Manager

Il caso di abuso della risorsa *Shard's Mapping*, denominato *Shard's Mapping Interception & Modification*, riportato in Figura 2.16, si riferisce alla possibilità che il mapping dello stato degli Shard, mantenuto all'interno dell'On-Chain Manager, venga intercettato e modificato da un attaccante rendendolo non coerente con lo stato reale della Blockchain.

Case Type	Abuse Case	Case ID	AT-09-01
Case Name	Shard's Mapping Interception & Modification		
Actors	On-Chain Manager, Off-Chain Manager, Attacker		
Description	L'On-Chain Manager mantiene al suo interno un mapping dello stato degli Shard sulla Blockchain. Il caso di abuso si riferisce alla possibilità che tale mapping venga intercettato e alterato da un attaccante quando l'Off-Chain Manager lo richiede per ricostruire il Registro.		
Data	Shard's Mapping		
Stimulus and preconditions	L'On-Chain Manager non ha previsto una protezione adeguata contro la possibilità di intercettare e modificare il contenuto del mapping che esso memorizza e scambia con l'Off-Chain Manager. L'attaccante deve essere in grado e avere dei mezzi per intercettare e alterare dati sui quali non ha autorizzazione di lettura e modifica.		
Attack Flow 1	L'attaccante intercetta e modifica il contenuto del mapping degli shard che l'On-Chain Manager invia all'Off-Chain Manager, compromettendo l'autenticità e l'integrità di tale asset.		
Response and Postconditions	Il mapping degli Shard è corrotto e non rispecchia più lo stato reale della Blockchain, rendendo il registro non autentico.		
Non Functional Requirements			
Mitigations	<ul style="list-style-type: none"> • Firma digitale dei dati. • Ridondanza di dati. 		
Comments	Per tale caso d'uso ci si è rifatti alla voce del catalogo CWE-ID 345 (Insufficient Verification of Data Authenticity)		

Figura 2.16: Shard's Mapping Abuse Case - Shard's Mapping Interception & Modification

Il caso di abuso della risorsa *Shard's Address*, denominato *Shard's Address Interception & Modification* e riportato in Figura 2.17, si riferisce alla possibilità che l'indirizzo dello Shard, su cui effettuare il deploy di uno Smart Contract, venga intercettato e modificato da un attaccante.

Case Type	Abuse Case	Case ID	AT-10-01
Case Name	Shard's Address Interception & Modification		
Actors	Off-Chain Manager, On-Chain Manager, Attacker		
Description	L'Off-Chain Manager ha bisogno di conoscere l'indirizzo dello Shard sul quale effettuare il deploy di uno Smart Contract. Tale risorsa viene restituita dall'On-Chain Manager all'Off-Chain Manager. Il caso di abuso di tale risorsa si riferisce alla possibilità che l'indirizzo che l'On-Chain Manager invia venga compromesso.		
Data	Shard's Address		
Stimulus and preconditions	L'Off-Chain Manager richiede all'On-Chain Manager l'indirizzo dello Shard su cui effettuare il deploy di uno Smart Contract. Affinchè l'attacco possa avvenire, l'attaccante deve essere in grado di intercettare la risorsa target che viene trasferita e il target non deve aver previsto una protezione adeguata contro la possibilità di modificare il contenuto inviato.		
Attack Flow 1	Mentre la risorsa in questione viene trasferita dall'On-Chain Manager all'Off-Chain Manager l'attore malintenzionato modifica il contenuto e lo rende qualcosa di differente dal contenuto originale prodotto.		
Response and Postconditions	L'Off-Chain Manager riceve l'indirizzo di uno Shard differente da quello previsto. Ciò implica che, se l'indirizzo ricevuto dall'Off-Chain Manager corrisponde ad uno Shard sulla Blockchain, il deploy va a buon fine ma avviene su uno Shard differente da quello che avrebbe garantito il load balancing; altrimenti, se l'indirizzo ricevuto non corrisponde ad alcuno Shard allora il deploy fallisce.		
Non Functional Requirements			
Mitigations	<ul style="list-style-type: none"> • Firma digitale dei dati. • Ridondanza di dati. 		
Comments	Per tale caso d'uso ci si è rifatti alla voce del catalogo CWE-ID 345 (Insufficient Verification of Data Authenticity)		

Figura 2.17: Shard's Address Abuse Case - Shard's Address Interception & Modification

Per la risorsa *Block* sono stati individuati due casi d'abuso, denominati rispettivamente *Block Modification* e *Block Unjustification* (Figura 2.18 e 2.19).

Il primo scenario si verifica quando l'attaccante, ossia un nodo validatore della Blockchain, altera la propria copia locale andando a modificare uno o più blocchi al suo interno.

Il secondo scenario, invece, si verifica quando l'attaccante, ossia un nodo validatore della Blockchain, possiede il 33%+1 dello stake. In tal caso, l'attaccante non fornendo il suo attestato riesce ad impedire il raggiungimento della soglia minima necessaria per giustificare un blocco.

Case Type	Abuse Case	Case ID	AT-11-01
Case Name	Block Modification		
Actors	Shard, Attacker (dishonest node)		
Description	Il caso di abuso si riferisce allo scenario in cui un nodo (attaccante) altera la sua copia locale della catena, modificando uno o più blocchi nelle transazioni che esso contiene.		
Data	Block		
Stimulus and preconditions	L'attaccante è un nodo validatore della Blockchain.		
Attack Flow 1	L'attaccante modifica la sua copia locale di blocco, alterando una o più transazioni che esso contine.		
Response and Postconditions	Il blocco viene modificato nelle transazioni che esso contiene.		
Non Functional Requirements			
Mitigations	Le transazioni di un blocco sono organizzate in una struttura dati denominata Merkle Tree. Questa struttura determina una catena di hash che crea una struttura dati a prova di modifica. Infatti, andando a modificare una transazione, si andrebbe a modificare l'hash della stessa. È possibile controllare che non ci siano alterazioni all'intero della struttura dati, ricevuto un blocco, andando a ricalcolare tutti gli hash delle transazioni che fanno parte del blocco e verificando che l'hash ottenuto corrisponde all'hash del nodo radice. Se questo accade allora si può essere sicuri dell'assenza di alterazioni. In questo modo un utente è in grado di verificare se una transazione è inclusa o meno in un blocco.		
Comments	Per tale caso d'uso ci si è rifatti alla voce del catalogo CWE-ID 345 (Insufficient Verification of Data Authenticity)		

Figura 2.18: Block Abuse Case - Block Modification

Case Type	Abuse Case	Case ID	AT-11-02
Case Name	Block Unjustification		
Actors	Shard, Attacker (dishonest node)		
Description	In Ethereum, la finalizzazione di un blocco si basa su un meccanismo che prevede una fase precedente che stabilisce la giustificazione di un blocco, quando questo raggiunge il 66%+1 dello stake, in base alla somma di tutti i certificati ottenuti dal blocco. Se un attaccante possiede il 33%+1 dello stake e non fornisce il suo attestato al blocco, questo non può essere giustificato.		
Data	Block		
Stimulus and preconditions	L'attaccante è un nodo validatore della Blockchain e possiede il 33%+1 dello stake.		
Attack Flow	L'attaccante non fornisce il suo attestato al blocco impedendo il raggiungimento della soglia minima per la giustificazione del blocco.		
Response and Postconditions	Il blocco non viene giustificato e questo implica che non possa essere finalizzato, essendo la giustificazione l'anticamera della finalizzazione del blocco.		
Non Functional Requirements			
Mitigations	In Ethereum, per tale attacco alla giustificazione dei blocchi esiste un meccanismo automatico per il quale quando ci sono dei blocchi che dopo più di 4 epoche non sono stati finalizzati, chi non ha ancora votato o chi ha votato contro la maggioranza parziale ottenuta perde parte dei soldi posti sullo stake. Quindi l'attaccante posticipando il suo attestato o votando contro la maggioranza, inizia a perdere dei soldi.		
Comments			

Figura 2.19: Block Abuse Case - Block Unjustification

Gli *attack tree* sono uno strumento di analisi utilizzato per valutare le vulnerabilità di un sistema. Si tratta di una rappresentazione grafica della possibile sequenza di eventi che un attaccante può utilizzare per compromettere la sicurezza di un sistema.

L'attack tree si compone di una radice, ovvero il nodo principale, che rappresenta l'obiettivo dell'attacco, e di nodi figli che rappresentano le fasi necessarie per raggiungere l'obiettivo. Ogni nodo figlio, a sua volta, ha altri nodi figli che descrivono le azioni necessarie per completare quella fase dell'attacco. In questo modo, gli attack tree possono essere utilizzati per identificare le falte nella sicurezza di un sistema e valutare i possibili scenari di attacco.

Nella Figura 2.20 si riporta l'attack tree relativo all'asset *Wallet*. Per tale asset, come introdotto precedentemente, sono stati identificati due casi di abuso: "*Wallet Key Theft*" e "*User's Identity Violation*".

L'attaccante con quest'ultimo attacco riesce ad ottenere le credenziali di un altro utente, sfruttando gli attacchi di *phishing* e *key logging*, precedentemente descritti.

Per quanto riguarda il pattern denominato "*Wallet Key Theft*" sono stati identificati due diversi attack flow. Il primo consiste in un attacco di *Brute Forcing* dove l'attaccante prova tutte le possibili chiavi fino a trovare quella corretta, per poi appropriarsene.

Il secondo, detto *Use of Known Domain Credentials*, può comportare il furto completo delle credenziali o di una parte di esse. Il furto completo può essere eseguito mediante tecniche di phising, social engineering o tramite un furto fisico, mentre il furto di una parte delle credenziali può essere eseguito sfruttando l'*Attacco del Dizionario*.

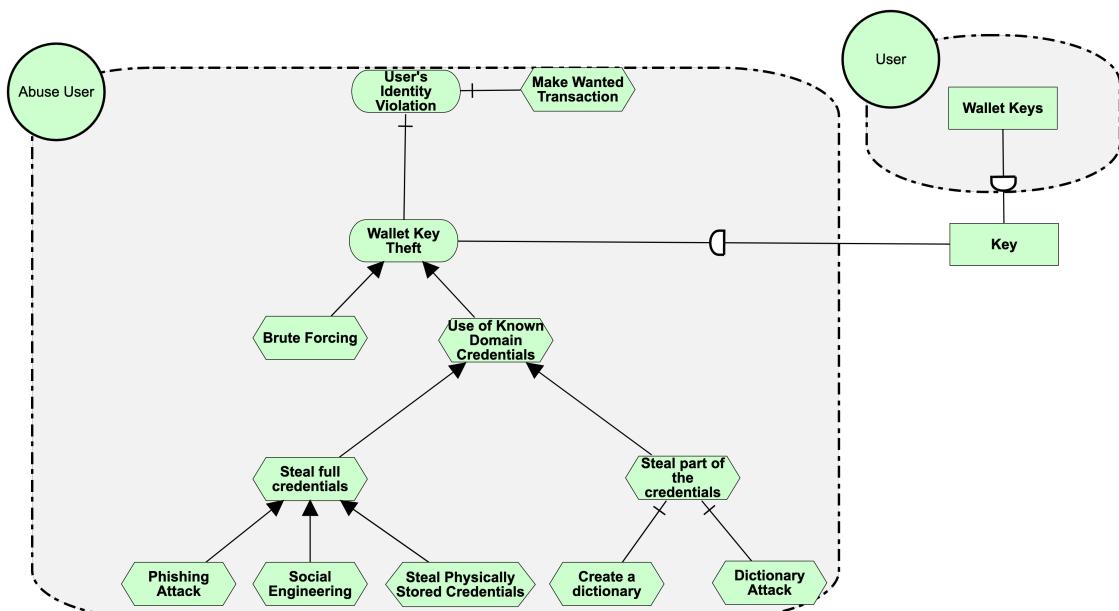


Figura 2.20: Attack Tree - Wallet

Nella Figura 2.21 è raffigurato l'attack tree relativo agli asset *Smart Contract* e *Transaction Result*.

I pattern d'attacco *Code Interception & Modification* per lo Smart Contract e *Transaction Result Interception & Modification* per Transaction Result, precedentemente introdotti, sono entrambi scomponibili negli attacchi di *Adversary In The Middle (AITM)* e *Content Spoofing*, i quali rispettivamente permettono di intercettare e modificare la risorsa in transito, che nel caso dello *Smart Contract* è il codice, mentre nel caso di *Transaction Result* è il risultato stesso della transazione.

La compromissione del codice di uno Smart Contract da parte di un attaccante porta all'esecuzione di un metodo della Blockchain diverso da quello voluto dall'utente. D'altro canto la compromissione del risultato della transazione da parte di un attaccante porta ad avere e a restituire all'utente un risultato non autentico diverso da quello effettivo.

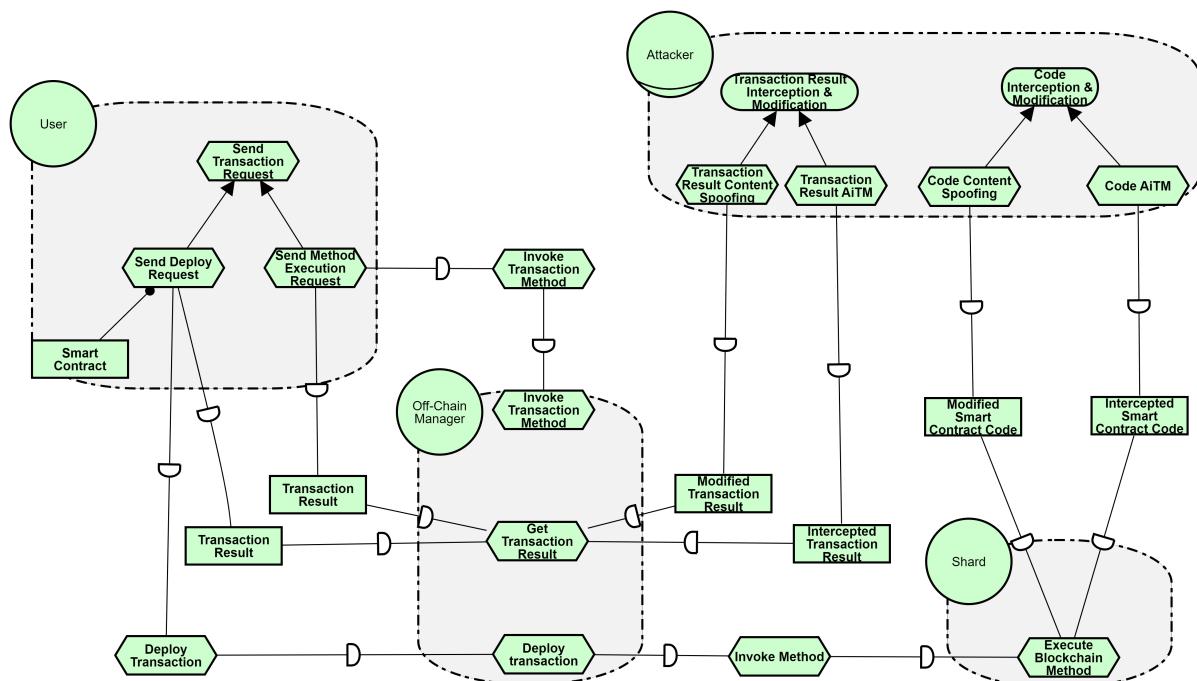


Figura 2.21: Attack Tree - Smart Contract & Transaction Result

Nella Figura 2.22 è riportato l'attack tree degli asset *Shard's Mapping* e *Register*.

I pattern d'attacco *Shard's Mapping Interception & Modification* e *Register Interception & Modification*, precedentemente introdotti, sono entrambi scomponibili negli attacchi di *Adversary In The Middle (AITM)* e *Content Spoofing*, i quali, come già detto, permettono rispettivamente di intercettare e modificare la risorsa in transito, che nel caso dello *Shard's Mapping* è il mapping, mentre nel caso del *Register* è il registro stesso.

La compromissione dell'integrità del mapping degli Shard porta all'invalidazione dello stato attuale della Blockchain, compromettendone il balancing, e alla violazione dell'autenticità del registro che l'utente può visionare.

Mentre la compromissione del registro porta l'utente a visionare un registro che non rappresenta in modo autentico ciò che è contenuto negli Shard.

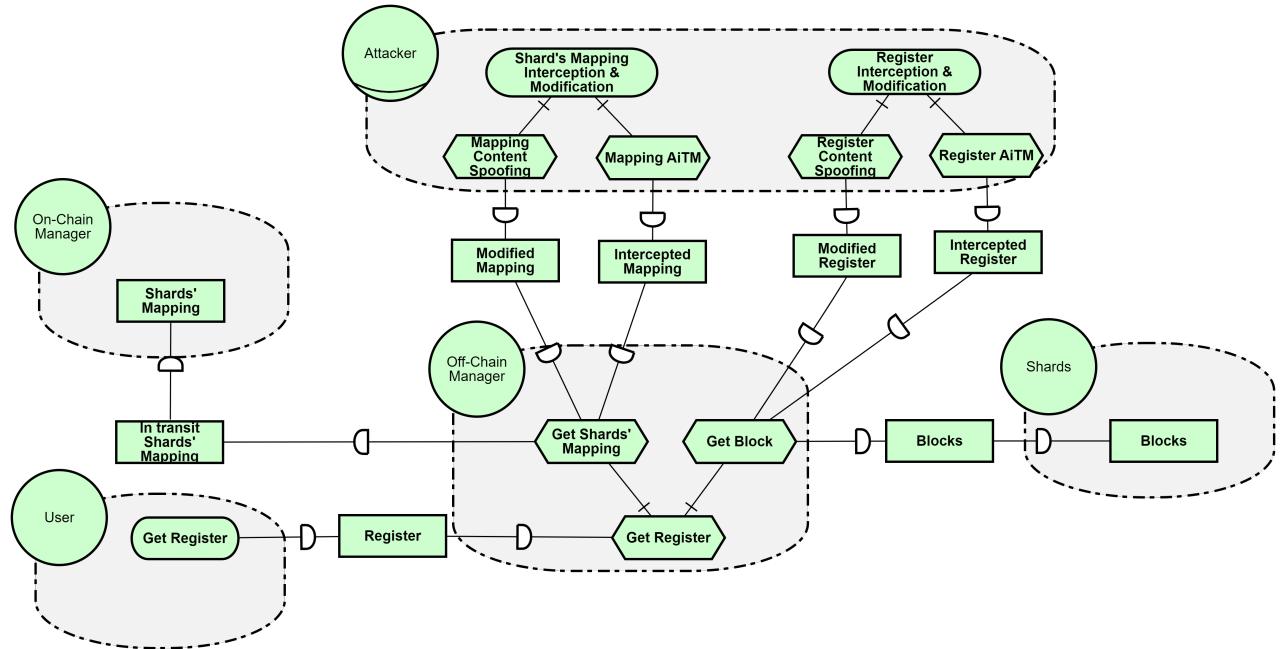


Figura 2.22: Attack Tree - Shard's Mapping & Register

Nella Figura 2.23 sono rappresentati i pattern di attacco agli asset *Smart Contract Address* e *Smart Contract Location* precedentemente introdotti (*Smart Contract Address Interception & Modification* e *Smart Contract Location Interception & Modification*). Sono entrambi scomponibili nei due attacchi di *Content Spoofing* e *Adversary In The Middle (AITM)*.

Nel primo caso l'attaccante altera l'indirizzo dello Smart Contract in cui si trova il metodo della transazione che l'utente vuole invocare. In questo modo l'*On-Chain Manager* riceve un indirizzo diverso da quello atteso. Esistono due possibili conseguenze a questo attacco:

- l'indirizzo inserito dall'attaccante esiste e corrisponde ad un altro Smart Contract, quindi viene invocato un metodo indesiderato o inesistente;
- l'indirizzo inserito dall'attaccante non esiste e l'invocazione del metodo non va a buon fine.

Nel secondo caso l'*Off-Chain Manager* riceve una locazione differente dalla locazione reale dello Smart Contract che implementa il metodo che si vuole invocare. Questo implica che la richiesta di invocazione del metodo da parte dell'Off-Chain Manager non vada a buon fine, poiché la locazione che gli viene restituita non corrisponde a quella attesa.

Nella Figura 2.24 è riportato l'attack tree dell'asset *Shard's Address*.

Anche il pattern d'attacco precedentemente introdotto per questo asset (*Shard's Address Interception*

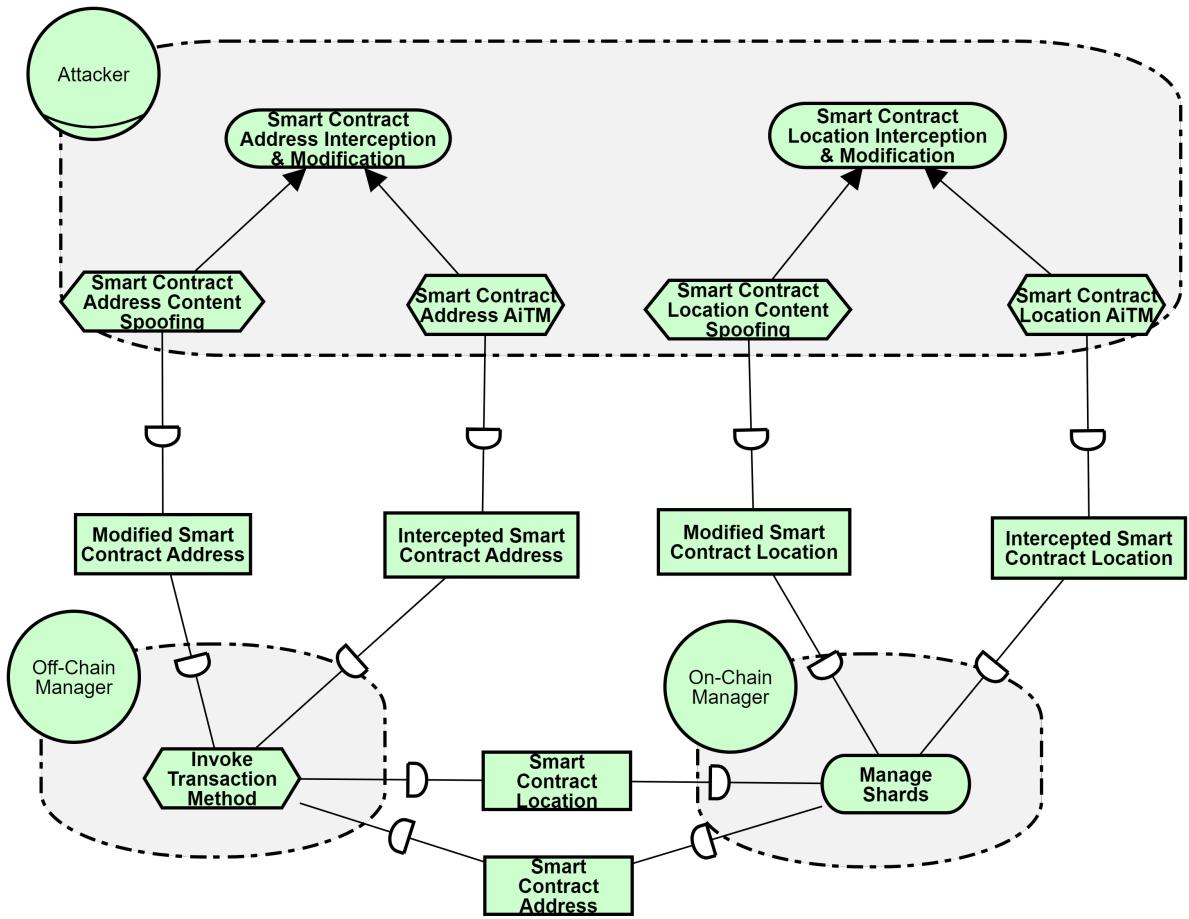


Figura 2.23: Attack Tree - Smart Contract Address & Smart Contract Location

& Modification) è scomponibile negli attacchi di *Content Spoofing* e *AITM*.

L'attaccante, dopo essersi intromesso nella comunicazione tra On-Chain Manager e Off-Chain Manager, compromette l'indirizzo dello Shard su cui si vuole effettuare il deploy e questo due possibili conseguenze:

- l'indirizzo inserito dell'attaccante esiste e corrisponde ad uno Shard diverso da quello in cui si dovrebbe effettuare il deploy in modo tale da mantenere il balancing;
- l'indirizzo inserito dall'attaccante non esiste e quindi il deploy non va a buon fine.

Nella Figura 2.25 è rappresentato l'attack tree dell'asset *Deploy Transaction*.

Il pattern d'attacco *Transaction Interception & Modification*, precedentemente introdotto, prevede due scenari differenti, i quali sono entrambi scomponibili in *Transaction Content Spoofing* e *Transaction AITM*.

La differenza tra i due sta nella comunicazione in cui si inseriscono:

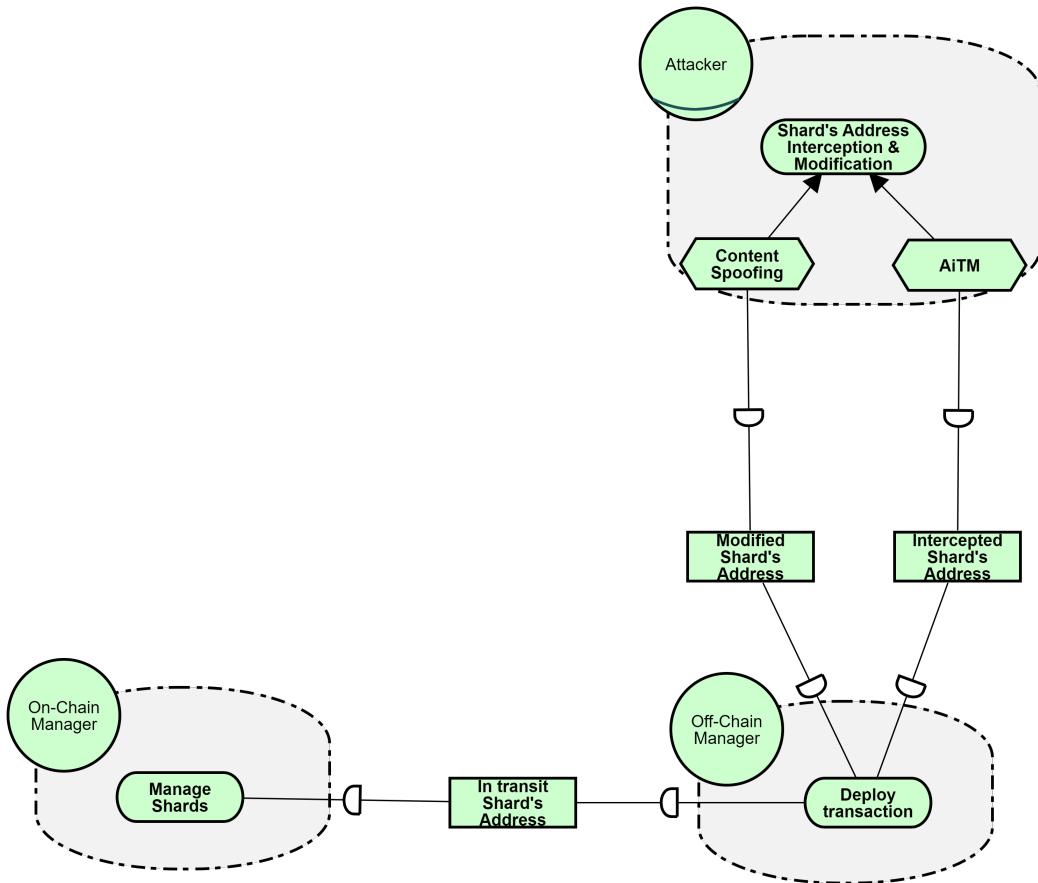


Figura 2.24: Attack Tree - Shard's Address

- nel primo scenario, l'attaccante si intromette nella comunicazione tra On-Chain Manager e Off-Chain Manager fingendosi un proxy di routing e impersonifica uno dei due attori alterando la gestione degli Shard come vuole;
- nel secondo scenario, l'attaccante si intromette nella comunicazione tra Shard e Off-Chain Manager fingendosi un proxy di routing e impersonifica uno dei due attori alterando il metodo della Blockchain da eseguire.

Nell'attack tree è stato rappresentato anche il caso di misuso *Involuntary Request Repetition* in cui l'utente disattento non si accorge di aver ripetuto più volte la richiesta di transazione e sperpera così le sue risorse sovraccaricando l'Off-Chain Manager.

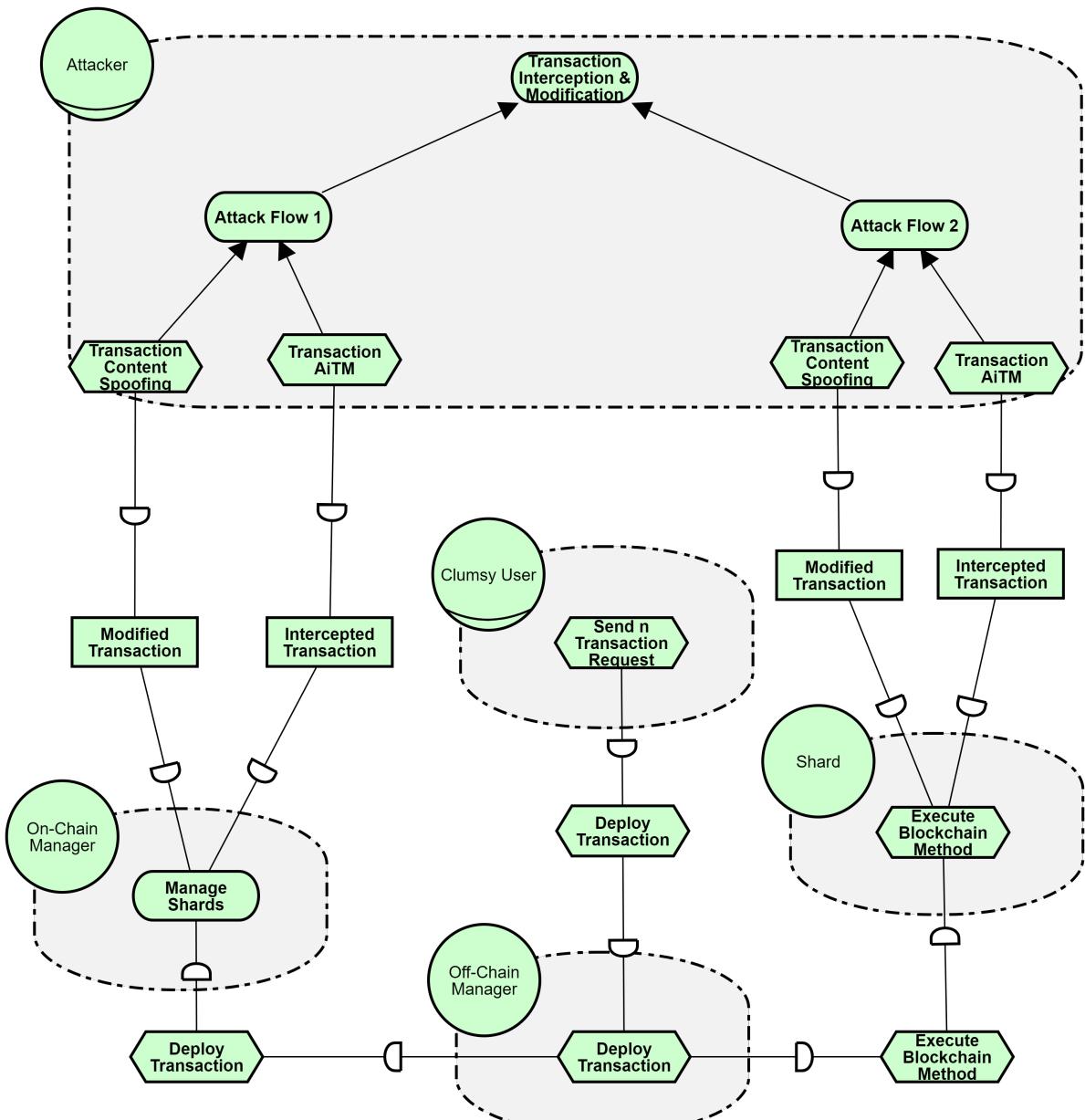


Figura 2.25: Attack Tree - Deploy Transaction

Nella Figura 2.26 sono raffigurati i tre pattern d'attacco precedentemente introdotti per l'asset *Update Shards' State*.

Il pattern *Update Shards' State Request Interception & Modification* è suddivisibile nei due attacchi di *Content Spoofing* e *Adversary In The Middle (AITM)*. In questo caso l'attaccante, dopo essersi intromesso nella comunicazione tra Off-Chain Manager e On-Chain Manager, impersonifica l'Off-Chain Manager, passando all'On-Chain Manager uno stato degli Shard diverso da quello effettivo.

Il pattern *DoS Aimed Request* è suddivisibile invece negli attacchi di *Flooding* e *Request Sustained*

Client Engagement:

- il primo comporta l'invio di n richieste di transazione, in modo tale da sovraccaricare l'On-Chain Manager e impedirgli di lavorare come dovrebbe;
- il secondo comporta l'invio di una sola richiesta che impegna l'On-Chain Manager per un tempo prolungato, impedendogli anche in questo caso di lavorare a dovere.

Infine il pattern *Unauthorized Communication With On-Chain Manager* vede, come già detto in precedenza, l'attaccante intromettersi nella comunicazione tra Off-Chain Manager e On-Chain Manager e, fingendosi l'Off-Chain Manager, invia delle richieste di transazione non autorizzate.

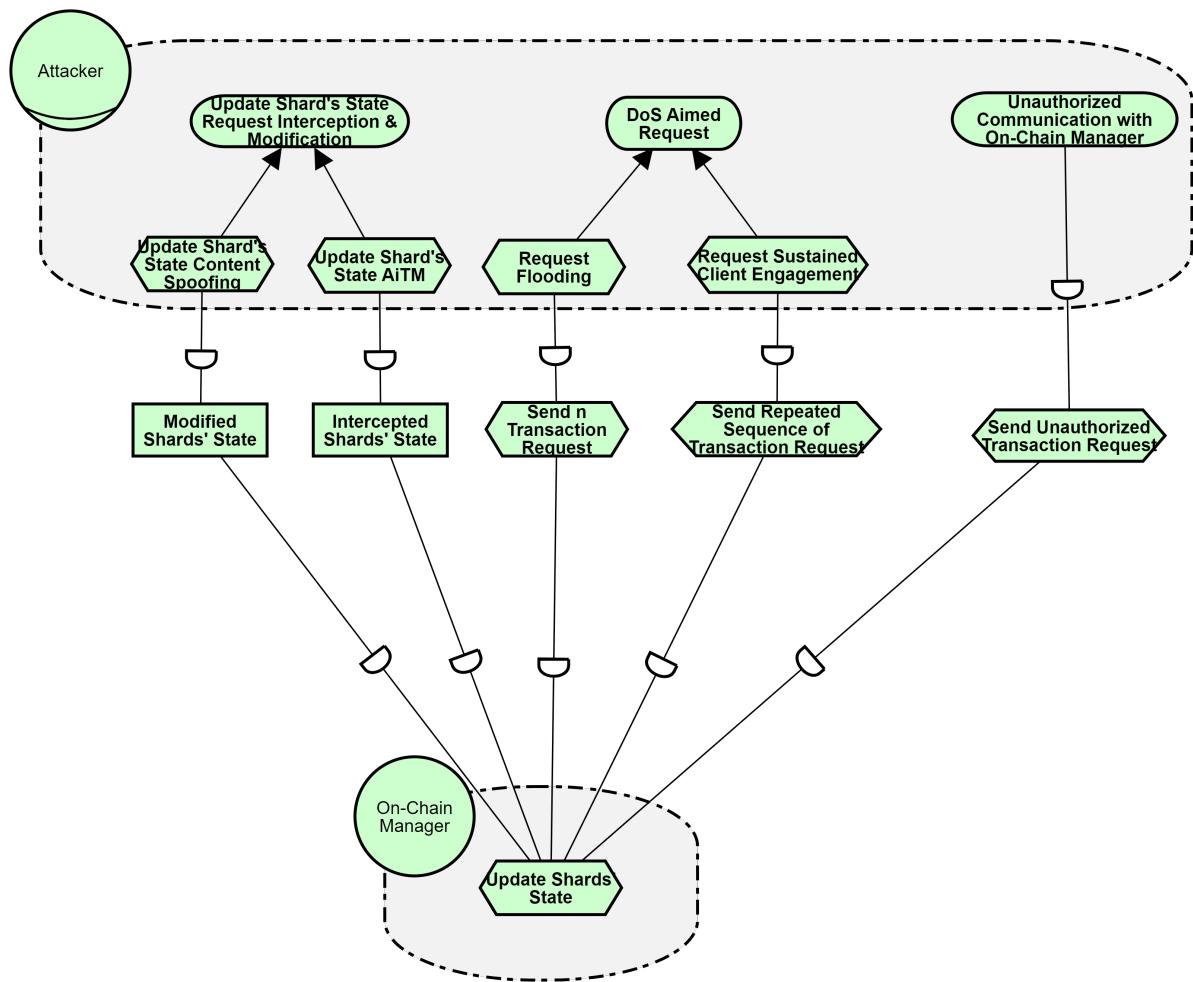


Figura 2.26: Attack Tree - Update Shard's State

Nella Figura 2.27 sono riportati i pattern d'attacco precedentemente introdotti per l'asset *Block* ("Block Modification" e "Block Unjustification").

- Nel primo caso l'attaccante si finge un nodo e modifica in modo malevolo la transazione che è salvata nel blocco.
- Nel secondo caso l'attaccante si finge un nodo e si appropria del 33% + 1 dello stake, impedendo così la giustificazione di un nuovo blocco e ostacolando il raggiungimento del consenso.

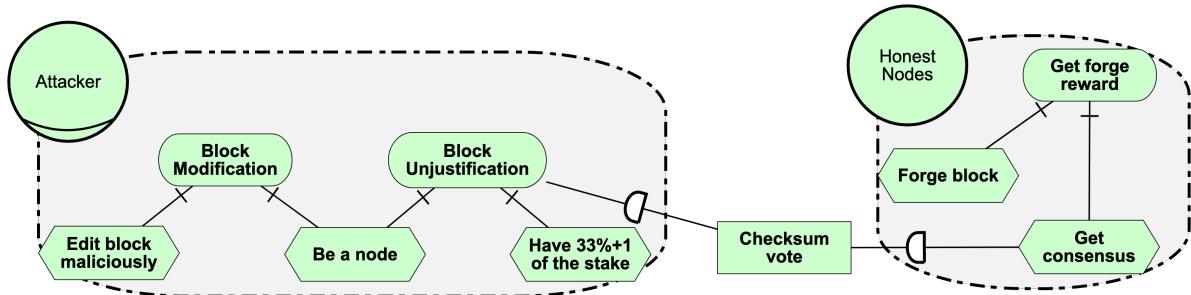


Figura 2.27: Attack Tree - Block

Al termine di questa fase sono state compilate ulteriori colonne della tabella STRIDE (Figure 2.28 e 2.29):

- *Attack*: sulla base delle minacce precedentemente identificate, sono stati individuati gli attacchi a cui ciascun asset è vulnerabile.
- *CAPEC Pattern*: per ogni attacco sono stati riportati i pattern noti presenti nel catalogo CAPEC.
- *Inherent Probability*: stima della probabilità di occorrenza di un attacco.
- *Inherent Risk*: stima del rischio inherente, ossia del rischio che si ha allo stato attuale, senza aver applicato nessuna azione preventiva.

Asset	Attack	CAPEC Pattern	Inherent Probability	Inherent Risk
Wallet	Wallet Key Theft	Brute Force (112)	5%	500-1.000
		Use of known domain credentials (560)	14%	1.400-2.800
	User's Identity Violation	Identity Spoofing (151)	23%	4.600-13.800
Shards' Mapping	Shard's Mapping Interception & Modification	AiTM (94)	19%	19.000-28.500
		Content Spoofing (148)	16%	16.000-24.000
Shard's Address	Shard's Address Interception & Modification	Content Spoofing (148)	19%	19.000-28.500
		AiTM (94)	16%	16.000-24.000
Smart contract	Code Interception & Modification	AiTM (94)	19%	950-1.900
		Content Spoofing (148)	16%	800-1.600
Register	Register Interception & Modification	AiTM (94)	19%	3.800-7.600
		Content Spoofing (148)	16%	3.200-6.400
Transaction Result	Transaction Result Interception & Modification	AiTM (94)	19%	190-475
		Content Spoofing (148)	16%	160-400

Figura 2.28: Tabella STRIDE - prima parte

Asset	Attack	CAPEC Pattern	Inherent Probability	Inherent Risk
Smart Contract Location	Smart Contract Location Interception & Modification	AiTM (94)	19%	1.900-3.800
		Content Spoofing (148)	16%	1.600-3.200
	Smart Contract Address Interception & Modification	AiTM (94)	19%	4.750-9.500
		Content Spoofing (148)	16%	4.000-8.000
	Transaction Interception & Modification	AiTM (94)	19%	9.500-19.000
		Content Spoofing (148)	16%	8.000-16.000
Deploy Transaction				
	Involuntary Request Repetition	/	12%	1.200-2.400
Update Shards' State	Update Shard's State Request Interception & Modification	AiTM (94)	19%	9.500-19.000
		Content Spoofing (148)	16%	8.000-16.000
	DoS Aimed Requests	Flooding (125)	26%	20.800-39.000
		Sustained Client Engagement (227)	22%	17.600-33.000
		Unauthorized Communication with On-Chain Manager	75%	187.500-375.000
Block	Block Modification	Content Spoofing (148)	16%	16.000-24.000
	Block Unjustification (33%+1 Attack)	/	33%	16.500-33.000

Figura 2.29: Tabella STRIDE - seconda parte

2.6 Risk Reduction

La fase di Risk Reduction si riferisce al processo di identificazione e implementazione di misure di sicurezza preventive, volte a ridurre il rischio di vulnerabilità ed evitare eventuali attacchi. L'obiettivo, dunque, è quello di valutare le possibili contromisure di sicurezza capaci di mitigare il rischio.

Ridurre il rischio può voler dire ridurre la probabilità d'attacco oppure ridurre l'impatto; nel migliore dei casi, può voler dire riuscire a ridurre entrambi.

2.6.1 Control Identification

Noti gli attacchi e i rischi che il verificarsi di ciascuna di essi comporta, sono stati identificati dei possibili meccanismi di controllo che cercano di ridurre i danni.

Nell'analisi delle varie soluzioni ammissibili, non è sufficiente enumerare le singole soluzioni poichè va tenuta in considerazione anche la possibilità di avere una combinazione delle soluzioni.

2.6.2 Feasibility Assessment

Per ciascun meccanismo, sono stati esaminati non solo i costi ma anche la fattibilità, ovvero se una soluzione è effettivamente realizzabile e conveniente nel suo utilizzo. Un ulteriore aspetto che è stato tenuto in considerazione è l'influenza del nuovo meccanismo introdotto sull'usabilità del software.

Terminato lo studio di fattibilità, è stato valutato come ridurre il rischio, confrontando diverse opzioni.

Non sempre la soluzione che riduce di più il rischio è quella più conveniente a livello economico, per questo motivo si è deciso di utilizzare un'ulteriore misura, il *Return of Control* che può essere espresso con la seguente formula:

$$\text{Return of Control} = \frac{\text{Reduction of expected loss}}{\text{Control of cost}} - 1 = \frac{\text{Risk}_{\text{inherent}}(R_j) - \text{Risk}_{\text{residual}}(R_j)}{\text{Control of cost}} - 1$$

2.6.3 Security Requirements Definition

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 2.30-2.33). 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;

- *Multiple Control Cost*: stima del costo necessario per implementare la combinazione di più meccanismi di controllo per un singolo attacco;
- *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 Multiple Control Probability*: stima della probabilità di occorrenza residua dopo aver adottato la combinazione di più meccanismi di controllo per un singolo attacco;
- *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;
- *Residual Multiple Control Risk*: stima del rischio residuo dopo aver adottato la combinazione di più meccanismi di controllo per un singolo attacco;
- *RoC*: valore del Return of Control.

Asset	Control	Cost	Multiple Control Cost	Feasibility
Wallet	Limited number of possible tries	1.000	1.500	This control does not have a great impact on the user usability and it is not hard to implement. It consists of a counter that is increased every time the user tries to log in. When the counter reaches a fixed limit of attempts, the user is temporarily blocked out.
	Suggest the user to change the password every three months	500		This control does not have a great impact on the user usability and it is not hard to implement. Every three months, when the user logs in, the system will print out a message suggesting to update the password.
	Two-factor Authentication	8.000		This control has an impact on the user acceptability and it is harder to implement depending on the second factor that is chosen (for example a user could not accept to provide the system with personal information). An example of second factor that could be implemented on the system is the OTP mechanism.
Shards' Mapping				
Shard's Address				
Smart contract	Digital signature and redundant data	10.000		<p>This mechanism does not have impact on the users' acceptability as it does not have a direct impact on their user experience. However, this control can generate overhead for the system and can increase the time of execution as encryption primitives are computationally hard.</p> <p>Plain data are transmitted accompanied by an encrypted hash of the messages that allows to verify the integrity of the information. Another way to mitigate data modification is to provide different copies of the same data. Redundancy causes higher costs, additional overhead and decreases efficiency of the system.</p>
Register				
Transaction Result				
Smart Contract Location				
Smart Contract Address				
Deploy Transaction	Limit User's Request over a time period	8.000		<p>This mechanism could make the user wait for a longer time in case of repetitive erroneous requests of deploy.</p> <p>To implement this control the system could log the user's requests of deploy and check if the number of those requests is higher of a fixed limit over a time period.</p>

Figura 2.30: Tabella STRIDE - prima parte

Asset	Residual Probability	Residual Multiple Control Probability	Residual Impact	Residual Risk	Residual Multiple Control Risk	RoC
Wallet	3%	11%	10.000-20.000	300-600	900-1.800	-0,6
	10%			1.000-2.000		0,6
	15%	/	20.000-60.000	3.000-9.000	/	-0,4
Shards' Mapping	11%	/	10.000-15.000	1.100-1.650	/	1,7
	9%		10.000-15.000	900-1.350		1,3
	11%	/	10.000-15.000	1.100-1.650	/	1,7
Shard's Address	9%		100.000-150.000	900-1.350		1,3
	11%	/	5.000-10.000	550-1.100	/	-0,9
	9%		5.000-10.000	450-900		-0,9
Register	11%	/	2.000-4.000	220-440	/	-0,3
	9%		2.000-4.000	180-360		-0,4
Transaction Result	11%	/	1.000-2.500	110-275	/	-1
	9%		1.000-2.500	90-225		-1
Smart Contract Location	11%	/	1.000-2.000	110-220	/	-0,6
	9%		1.000-2.000	90-180		-0,7
Smart Contract Address	11%	/	25.000-50.000	2.750-5.500	/	-0,6
	9%		25.000-50.000	2.250-4.500		-0,7
Deploy Transaction	11%	/	50.000-100.000	5.500-11.000	/	-0,2
	9%		50.000-100.000	4.500-9.000		-0,3
	7%	/	10.000-20.000	700-1.400	/	-0,9

Figura 2.31: Tabella STRIDE - seconda parte

Asset	Control	Cost	Multiple Control Cost	Feasibility
Update Shards' State	Digital signature and redundant data	10.000		<p>This mechanism does not have impact on the users' acceptability as it does not have a direct impact on their user experience. However, this control can generate overhead for the system and can increase the time of execution as encryption primitives are computationally hard.</p> <p>Plain data are transmitted accompanied by an encrypted hash of the messages that allows to verify the integrity of the information. Another way to mitigate data modification is to provide different copies of the same data. Redundancy causes higher costs, additional overhead and decreases efficiency of the system.</p>
	Limit User's Request over a time period	8.000		<p>This mechanism could make the user wait for a longer time in case of repetitive requests of deploy.</p> <p>To implement this control the system could log the user's requests of deploy and check if the number of those requests is higher of a fixed limit over a time period.</p>
	Design algorithms that don't allow a sustained lock by a small group of users	10.000		Avoid mechanism that require a lock. In case this is needed, a possible control mechanism to avoid an abuse is limiting users' requests by logging their accesses to the resource.
	-	-		
Block	Merkle tree	/		The block transactions are organized in a data structure which name is Merkle tree. This structure determines a hash chain that is editing-resistant. Editing a transaction would edit the hash tree so that, if the transactions hash tree is computed, it is possible to verify that the transactions saved in a block are not altered.
	The attacker could partially lose their stake	/		After four epochs, if a block is still not justified, there is an automatic control that checks who voted against the justification and those validator nodes could partially lose their stake. This control mechanism does not affect the user experience. The implementation is not difficult because this control is already provided by Ethereum.

Figura 2.32: Tabella STRIDE - terza parte

Asset	Residual Probability	Residual [Multiple Control] Probability	Residual Impact	Residual Risk	Residual Multiple Control Risk:	RoC
Update Shards' State	11%	/	50.000-100.000	5.500-11.000	/	-0,2
	9%	/	50.000-100.000	4.500-9.000	/	-0,3
	15%	/	80.000-150.000	12.000-22.500	/	1,1
	12%	/	80.000-150.000	9.600-18.000	/	0,5
Block	3%	/	10.000-15.000	300-450	/	/
	17%	/	50.000-100.000	8.500-17.000	/	/

Figura 2.33: Tabella STRIDE - quarta parte

3

Software Design

Indice

3.1 Design assets	54
3.2 Architectural design	54

Nella fase di analisi dei requisiti sono state definite delle specifiche che però non sono sufficienti per la definizione di un software.

Pertanto, è fondamentale definire l'architettura del software e l'interazione tra i suoi componenti.

L'utilizzo di una determinata architettura e le scelte tecnologiche fatte hanno un proprio impatto sulla sicurezza e sul raggiungimento degli obiettivi di dependability.

Di conseguenza, è importante valutare le opzioni disponibili al fine di garantire il rispetto degli obiettivi di sicurezza imposti.

Tutte le decisioni prese in fase di design sono conformi alle policy di sicurezza individuate nella fase di Risk Analysis, in modo tale da atternersi al principio *Base decisions on an explicit security policy* riportato nel *Catalogo di Sommerville*.

Inoltre, si è posta una particolare attenzione sul principio di *Open Design* riportato nel *Catalogo di Saltzer e Schroeder*, che corrisponde al principio *Avoid Security by Obscurity* riportato nel *Catalogo OWASP*. Questi sottolineano l'importanza di sviluppare il sistema attraverso l'utilizzo di informazione progettuale condivisa pubblicamente.

3.1 Design assets

Negli anni sono state definite delle linee guida, delle buone pratiche da seguire per svolgere correttamente la fase di progettazione. Queste, in generale, hanno l'obiettivo di rendere consapevoli su quali possono essere i problemi di sicurezza a seconda delle scelte che vengono fatte.

Sono stati proposti diversi cataloghi che contengono alcune linee guida comuni. Infatti, i cataloghi successivi a quello di Saltzer e Schroeder del 1975, riprendono alcune linee guida in esso presenti e le completano con ulteriori aspetti legati alle nuove innovazioni tecnologiche.

3.2 Architectural design

La ridondanza è un aspetto fondamentale nella progettazione di un software sicuro che prevede la duplicazione di componenti o funzionalità critiche per garantire la continuità del servizio. Infatti, se una copia smettesse di funzionare a causa di un guasto hardware o di un attacco, le altre copie continuerebbero a funzionare, garantendo così la corretta erogazione del servizio.

Nella scelta dell'architettura da adottare ha avuto una notevole influenza la struttura della Blockchain. Essa rappresenta una tecnologia che permette di creare un grande registro distribuito per la gestione di

transazioni condivisibili tra più nodi di una rete. Tutti i nodi hanno la stessa copia della Blockchain e tutti i nodi svolgono le medesime azioni: prendono una transazione, la controllano e se è corretta propongono di aggiungerla al registro.

Inoltre, i dati una volta scritti in un blocco non possono essere retroattivamente alterati senza che vengano modificati anche tutti i blocchi successivi ad esso e ciò, per la natura del protocollo di validazione, necessiterebbe del consenso della maggioranza della rete. Per questo motivo si dice che le transazioni su una Blockchain sono immodificabili.

La Blockchain è dunque un registro immutabile che di per sé implementa il concetto di ridondanza delle copie e di distribuzione sui vari nodi.

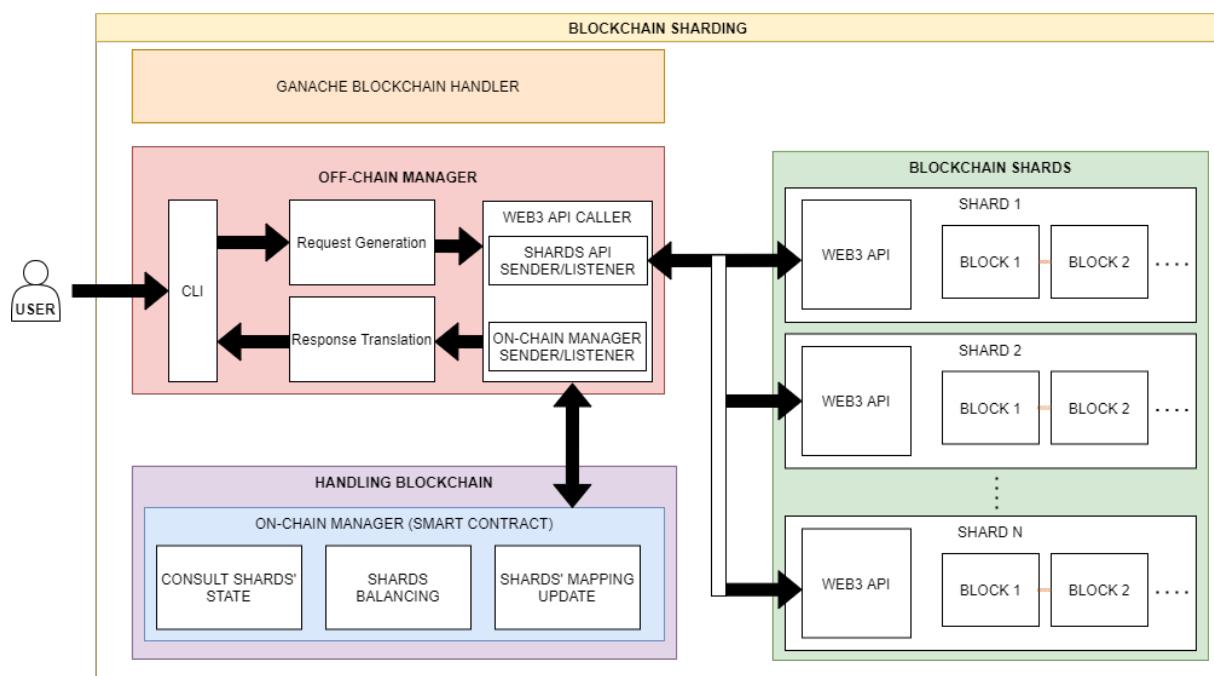


Figura 3.1: Architettura

Nella Figura 3.1 è rappresentata l'architettura utilizzata.

L'architettura è stata pensata in modo da limitare i punti di accesso, così da attuare il principio di sicurezza *Minimize Attack Surface* definito nel catalogo di OWASP.

Sono stati considerati quattro attori:

- *User*;
- *Off-Chain Manager*;
- *Handling Blockchain*, ossia una Blockchain di Gestione;
- *Blockchain Shards*, ossia una Blockchain che serve da database per gli Shard.

L'utente può interagire tramite linea di comando esclusivamente con l'*Off-Chain Manager* e solo dopo essersi autenticato potrà generare delle richieste di transazione. Si è deciso quindi di limitare al minimo le informazioni sul sistema di cui dispone l'utente, attendendosi così al principio *Least Privilege*, espresso nei cataloghi di Saltzer e Schroeder e OWASP.

Sarà poi l'*Off-Chain Manager* a gestire tali richieste e ad entrare in contatto con l'*On-Chain Manager* per conoscere lo stato degli Shard e gestire il deploy degli Smart Contract in maniera bilanciata; e con gli *Shard*, tramite l'interfaccia di Web3, per conoscerne il contenuto.

4

Implementation

Indice

4.1 Local Database	59
4.2 Authentication	60
4.3 Load Balancing	61
4.4 User's Actions Flow	63
4.5 Testing	68

L'implementazione è un aspetto tutt'altro che trascurabile. Infatti, è possibile che, nonostante il software sia stato progettato correttamente, l'implementazione contenga degli errori.

I possibili errori che si potrebbero commettere in fase di implementazione non si limitano ad una mancata conformità del codice rispetto al progetto: spesso il codice è scritto in modo apparentemente conforme al progetto, ma a causa di alcune scelte fatte si può rendere ugualmente vulnerabile il software.

Le vulnerabilità sono spesso specifiche della semantica operazionale del linguaggio che si sta utilizzando. Esistono comunque diversi cataloghi che contengono delle linee guida generali per la buona programmazione.

In tal caso si è tenuto conto delle linee guida riportate da Sommerville, le quali si riferiscono sia alla sicurezza che alla dependability.

In particolare, le linee guida che sono state seguite sono:

- *Limitare la visibilità delle informazioni di un programma*, linea guida che si ricollega al principio di buona progettazione di scorporare le componenti e fare accedere le componenti soltanto alle unità software minimamente necessarie.
- *Verificare la validità di tutti gli input*, linea guida ormai nota da decenni, la quale sottolinea l'importanza di verificare la validità degli input per evitare che il programma si comporti in modo imprevedibile quando vengono presentati input insoliti, cosa che a volte, si configura come una minaccia alla sicurezza del sistema. I controlli implementati sono controlli di intervallo, controlli di dimensione, controlli di rappresentazione.
- *Gestire le eccezioni*.
- *Ridurre al minimo l'uso di costrutti che possono facilmente indurre a errori*.
- *Controllare i limiti dell'array*.

I linguaggi utilizzati per il implementare il sistema sono:

- Solidity, per ciò che riguarda la parte On-Chain e i vari smart contract di esempio;
- Python, per ciò che riguarda la parte Off-Chain.

La decisione di utilizzare Python è stata presa alla luce del fatto che tale linguaggio offre una vasta gamma di funzioni di libreria (web3.py) che permettono di gestire l'interazione con una blockchain Ethereum e l'intero ecosistema Ethereum, evitando quindi di dover scrivere codice da zero in molti casi.

4.1 Local Database

Al fine di poter gestire al meglio i dati di ciascun utente, si è deciso di appoggiarsi su un database locale. In particolare, si è scelto di utilizzare SQLite3 poiché efficiente, multiplattforma ed open source. Inoltre, essendo le sue API nativamente supportate da Python, risulta particolarmente comodo ed agevole l'utilizzo di tale tecnologia.

Le informazioni personali associate all'utente che vengono salvate includono l'username, il digest della password e chiavi privata e pubblica del wallet associato.

Il digest della password corrisponde all'output ottenuto applicando la funzione di hashing SHA256 sulla password dell'utente.

Essendo la funzione di hashing una funzione one-way, se un attaccante ottenessse il digest non riuscirebbe a risalire alla password.

Per controllare che una password data sia corretta, a partire da quest'ultima verrà generato un digest utilizzando la medesima funzione di hashing, e controllando che questo corrisponda al digest memorizzato nel database.

Similmente si è agito a riguardo del salvataggio della chiave privata di utente.

Questa chiave, a differenza della chiave pubblica, deve essere conosciuta solo dal legittimo proprietario, poiché consente l'accesso ai fondi e garantisce la privacy dell'utente stesso.

Per questo motivo, si è deciso di crittografare la chiave privata tramite l'algoritmo di cifratura simmetrica AES, utilizzando la password dell'utente come chiave segreta.

Non cifrare questa informazione potrebbe consentire a un utente malintenzionato di avere accesso alla chiave privata, copiarla e utilizzarla per svolgere azioni illegittime.

La scelta di rendere di dominio pubblico la primitiva crittografica impiegata si rifà al principio *Avoid Security by Obscurity* definito nel Catalogo OWASP.

Inoltre, al fine di garantire l'integrità dei dati, è stato ritenuto necessario cifrare il timestamp relativo all'ultima modifica della password, con lo stesso meccanismo utilizzato per la chiave privata.

Così facendo, è stato aggiunto un ulteriore livello di sicurezza: se un attaccante dovesse accedere al sistema e tentare di modificare il timestamp, il testo in chiaro ottenuto decifrando non corrisponderebbe più ad un timestamp valido. Ciò permette di evitare che vi siano dei tentativi di manipolazione dei dati, volti a ritardare il momento in cui ad un utente viene suggerito dal sistema di cambiare la sua password.

Per ciascun utente sono state salvate anche le informazioni relative agli Smart Contract deployati. In particolare, nel database vengono memorizzati il nome, l'indirizzo dello smart contract e l'indirizzo dello

Shard.

4.2 Authentication

Per inviare delle richieste all'Off-Chain Manager, ogni utente deve autenticarsi tramite il proprio wallet. A tal proposito, si è pensato di utilizzare come credenziali per l'autenticazione, oltre alla coppia di chiavi pubblica e privata associate al wallet dell'utente, un username e una password scelti dall'utente. In particolare, per garantire la minima robustezza della password è richiesto all'utente che essa sia lunga almeno 10 caratteri, che contenga almeno una lettera maiuscola ed una minuscola e almeno un numero.

Sempre per questioni di sicurezza della password, si è ritenuto importante implementare un controllo per verificare da quanto tempo non viene cambiata la password di un utente. Infatti, se una password non viene cambiata per un lungo periodo di tempo, aumenta la possibilità che diventi vulnerabile ad attacchi informatici. Per questo motivo, dopo tre mesi dall'ultima modifica della password, viene mostrato un messaggio all'utente che consiglia di scegliere una nuova password.

Il cambio di password regolare è consigliato per proteggere gli account da malintenzionati e intrusi che mettono a rischio la privacy dell'utente.

Dunque, quando un utente effettua il login, o la registrazione qualora sia la prima volta che egli utilizza il software, gli viene richiesto, innanzitutto di inserire la propria chiave pubblica e la propria chiave privata e poi di scegliere un username e una password. Quest'ultima è vincolata da una *Regex*, ossia una regola di sicurezza che specifica come essa deve essere strutturata.

Dopo aver effettuato il login, se è trascorso un tempo superiore ai 3 mesi dall'ultima volta che la password è stata aggiornata, all'utente viene suggerito il cambio password.

Detto ciò, va comunque ricordato che il database in cui sono memorizzate le credenziali è locale per cui non si può considerare tale meccanismo una vera e propria autenticazione a due fattori, questo perchè appunto per un malintenzionato è sufficiente scoprire la sola chiave privata di un wallet per appropriarsene, senza necessariamente indovinare anche la password.

Infatti, avendo a disposizione la coppia di chiavi pubblica, che è nota in quanto tale, e privata, l'attaccante potrà utilizzare in modo improprio il wallet di un utente semplicemente registrandosi nuovamente da un dispositivo diverso da quello che utilizza l'utente, in cui nel database locale la coppia di chiavi non è associata alla password scelta dall'utente.

Tale scelta quindi non è tanto rivolta al tentativo di implementare un'autenticazione a due fattori, ma piuttosto è stato pensato per evitare che l'utente ad ogni transazione debba inserire ogni volta le sue chiavi, che così sarebbero maggiormente esposte.

Grazie al database locale in cui la coppia di chiavi e username e password sono memorizzati, è sufficiente che l'utente specifichi correttamente la sua password e poi sarà il sistema stesso che andando ad interrogare il database otterrà le chiavi del wallet dell'utente.

Inoltre, l'attuazione di queste misure in parte semplifica l'utilizzo del sistema da parte dell'utente, che piuttosto di dover inserire svariate volte la sua chiave privata, che presumibilmente non ricorda a memoria, deve semplicemente inserire la sua password.

In questo caso tutte le decisioni prese sono state fatte in virtù del principio *Balance Security and Usability* definito nel *Catalogo di Sommerville*.

Questo porta a cercare un "trade off" tra sicurezza e usabilità del sistema.

4.3 Load Balancing

Nelle Blockchain c'è da sempre un trilemma per tentare di soddisfare contemporaneamente decentralizzazione, scalabilità e security (noto come Blockchain trilemma).

Uno dei meccanismi che prova a trovare un compromesso tra questi tre elementi è lo sharding che si ispira alla struttura dei database nei quali i dati vengono partizionati per migliorare il processing delle transazioni.

La sicurezza di una Blockchain si basa sulla memorizzazione di ogni transazione in tutti i nodi ma questo a volte può rallentare considerevolmente il processing di ogni transazione. Lo sharding aiuta a distribuire il carico sulla rete di nodi e a diminuire la quantità di memoria utilizzata per ridondare i blocchi.

Sicuramente lo sharding pone l'accento sulla scalabilità aumentando la parallelizzazione delle operazioni in base al numero di shard presenti. Sono, invece, ancora in corso ricerche riguardo la sicurezza di questo meccanismo.

Infatti, in una Blockchain decentralizzata, ogni nodo ha il compito di verificare le transazioni e di aggiornare il registro distribuito. Tuttavia, alcuni nodi potrebbero avere più risorse o capacità di elaborazione rispetto ad altri. Se questi nodi avessero la possibilità di confermare la maggior parte delle transazioni, ciò potrebbe portare ad una centralizzazione del controllo e ad un potere decisionale limitato nelle mani di pochi nodi.

Quello che è certo è che lo sharding favorisce l'implementazione di Blockchain *suitable* poiché permette di ridurre il problema dello spazio di memorizzazione necessario per i blocchi che per natura di una

Blockchain è destinato a crescere infinitamente.

Il fulcro della bontà di un meccanismo di sharding si basa sull'implementazione dell'algoritmo di balancing per distribuire il carico. Per decidere l'algoritmo più adatto occorre tenere conto che le transazioni in Ethereum possono essere invocazioni di metodi degli smart contract o deployment degli stessi.

Proprio per questa ragione, poichè alcuni smart contract potrebbero essere invocati più frequentemente di altri, si è deciso di non affidarsi ad un algoritmo *Round Robin* (versione 1 di *On-Chain-Manager.sol* riportata in A.1) che avrebbe tenuto conto solo del deployment degli smart contract.

Per questo, l'algoritmo implementato effettua il deployment di uno smart contract sullo shard che ha forgiato il minor numero di blocchi.

Questo approccio non è in grado di garantire un balancing perfetto poichè in presenza di un numero elevato di utenti, il numero di transazioni effettuate potrebbe essere molto elevato.

Nonostante ciò, questo approccio consente di bilanciare il carico tra gli shard in modo sufficientemente equilibrato, implementando la iperproprietà di liveness "*Prima o poi gli shard saranno bilanciati*". L'algoritmo è stato implementato in Python in quanto l'informazione sul numero di blocchi forgiati da ogni shard è facilmente ottenibile tramite la libreria *web3.py*.

Inizialmente, si è optato per una definizione "hard coded" del numero di shard, il che significa che il valore era stato fissato direttamente nel codice sorgente.

Tuttavia, per migliorare la scalabilità del programma e rendere il sistema più flessibile, si è deciso di permettere all'amministratore di sistema di settare il parametro che specifica il numero degli shard da utilizzare tramite un file di configurazione.

4.4 User's Actions Flow

L'utente può interagire con il programma mediante una CLI (Command Line Interface), un'interfaccia a riga di comando che permette all'utente di interagire con il programma tramite dei comandi testuali.

4.4.1 Registration and Log in

All'avvio del programma viene mostrato un primo menù che presenta tre opzioni e richiede all'utente di sceglierne una (Figura 4.1).

```
1 -- Register.  
2 -- Log in.  
3 -- Exit.  
Enter your choice: □
```

Figura 4.1: Schermata iniziale

1. *Register*

Se l'utente non ha mai utilizzato l'applicazione, deve innanzitutto registrarsi. Per far questo, l'utente deve inserire le informazioni relative al proprio wallet e all'account. In particolare, l'utente deve compilare i seguenti campi: *Public Key*, *Private Key*, *Confirm Private Key*, *Username*, *Password* e *Confirm Password*.

Questi campi sono sottoposti a controlli per garantire la correttezza delle informazioni inserite. Per quanto riguarda le informazioni del wallet, se la chiave pubblica e la chiave privata inserite non corrispondono a nessun account, viene mostrato un messaggio di errore e all'utente viene richiesto di inserirle nuovamente.

Per quanto riguarda le informazioni relative all'account, viene mostrato un messaggio di errore se l'username inserito dall'utente è già utilizzato da un altro utente. Inoltre, se la password scelta dall'utente non rispetta la regex imposta, viene visualizzato un ulteriore messaggio di errore che specifica come deve essere composta la password e viene richiesto di inserirla nuovamente.

Se l'utente compila tutti i campi correttamente, la registrazione andrà a buon fine, come mostrato in Figura 4.2, e potrà procedere con il log in.

```
Handle option 'Option 1: Register'
Enter your wallet information.
Public Key: 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e
Private Key: 0xb0057716d5917badaf911b193b12b910811c1497b5bada8d7711f758981c3773
Confirm Private Key: 0xb0057716d5917badaf911b193b12b910811c1497b5bada8d7711f758981c3773
Enter your personal account information.
(in this way every time you log in or want to perform a transaction it will not be necessary
to provide your private key, but the username and password that you will specify below)
Username: alicemoretti
Password:
Confirm Password:
Registration was successful!
```

Figura 4.2: Registrazione

2. Log in

Dopo aver effettuato la registrazione, l'utente può effettuare il log in. Per autenticarsi deve inserire le credenziali specificate al momento della registrazione, ovvero *Public Key*, *Private Key*, *Username* e *Password*.

Se l'utente compila tutti i campi correttamente, comparirà un nuovo menù che specifica le operazioni che può svolgere una volta autenticato (Figura 4.3).

```
Handle option 'Option 2: Log in'
Public Key: 0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e
Private Key: 0xb0057716d5917badaf911b193b12b910811c1497b5bada8d7711f758981c3773
Username: alicemoretti
Password:

You are logged in.

1 -- Deploy Smart Contract.
2 -- Invoke Smart Contract's Method.
3 -- Consult your Smart Contract in your local database.
4 -- Delete Smart Contract from your local database.
5 -- Change password.
6 -- Logout.
Enter your choice: □
```

Figura 4.3: Log in

3. Exit

L'utente può decidere di uscire dal programma scegliendo l'opzione *Exit* come si riporta nella Figura 4.4.

```

1 -- Register.
2 -- Log in.
3 -- Exit.

Enter your choice: 3

Handle option 'Option 3: Exit'

Process finished with exit code 0

```

Figura 4.4: Uscita

4.4.2 Once Logged In

Dopo che l'utente si è autenticato, gli viene mostrato un menù con sei opzioni, come indicato in Figura 4.3.

1. Deploy Smart Contract

L'utente può decidere di deployare uno Smart Contract di cui è in possesso. Per garantire un adeguato livello di sicurezza, l'utente deve confermare nuovamente la propria identità inserendo la password prima di poter procedere.

Per proseguire viene richiesto di inserire il percorso in cui è salvato lo Smart Contract che si vuole deployare. Se il percorso è corretto, sarà stampata a video la frase "*We found your file: Smart Contract loaded correctly!*". Viene poi chiesto di inserire i valori del gas price e del gas limit e prima di completare il deploy dello smart contract viene effettuato un calcolo per preventivare il costo effettivo dell'azione di deploy e si richiede all'utente se è disposto a spendere tale quantità di gas. Se l'utente risponde in maniera affermativa, lo Smart Contract viene deployato (Figura 4.5).

```

Handle option 'Option 1: Deploy Smart Contract.'
Before proceeding with the deployment of Smart Contract it is necessary to enter the password.
Password:
Enter the path of your file: C:\Users\moret\OneDrive - Università Politecnica delle Marche\Primo Anno\Primo semestre\Software Security and Blockchain\swsb-project\LIBRACHAIN\soliditycontracts\OnChainManager.sol
We found your file: Smart Contract loaded correctly!

Smart Contract Name: OnChain
Gas limit (in Gwei): 100000000
Gas price (in Gwei): 20000
The estimated cost to deploy the smart contract is 745892.

Do you want proceed with the deploy (Y/N)?
y
Deployment was successful

Contract deployed at address: 0x07a457d878BF363E0Bb5aa0B096092f941e19962.

Shard address: http://localhost:8547.

```

Figura 4.5: Deploy Smart Contract

2. Invoke Smart Contract's Method

Dopo avere effettuato il deployment di uno o più Smart Contract, l'utente può decidere di invocare i metodi presenti in essi. Prima di procedere, l'utente deve confermare la sua identità inserendo la password.

Successivamente, all'utente viene richiesto di inserire il percorso di memoria in cui è salvato lo Smart Contract contenente il metodo che si desidera invocare. Se il percorso inserito è corretto, sarà stampata a video la frase "*We found your file: Smart Contract loaded correctly!*". Viene poi richiesto all'utente di inserire lo shard address e lo smart contract address a cui l'utente può, eventualmente, risalire digitando l'opzione 3 e consultando i dati dello Smart Contract di suo interesse.

A questo punto, viene mostrato a schermo l'elenco dei metodi presenti nello Smart Contract selezionato, evidenziati in modo differente a seconda che modifichino o meno lo stato. In particolare, i metodi colorati in rosso sono quelli che modificano lo stato, mentre i metodi colorati in blu sono quelli che non lo modificano.

Infine, l'utente deve decidere quale metodo selezionare tra quelli presenti nell'elenco, come mostrato in Figura 4.6.

```
Handle option 'Option 2: Invoke Smart Contract' Method.'  
Before proceeding with the invocation of a method of a smart contract it is necessary to enter the password.  
Password:  
Enter the path of your file (press 0 to go back): C:\Users\moret\OneDrive - Università Politecnica delle Marche\Primo Anno\Primo semestre\Software Security and Blockchain\swsb-project\\Librachain\solidity\contracts\OnChainManager.sol  
We found your file: Smart Contract loaded correctly!  
  
Enter smart contact address:0x07a457d878BF363E0Bb5aa0B896092f941e19962  
Enter shard address:http://localhost:8547  
1) addToDictionary(uint256,address)  
2) isValidAddress(uint256,address)  
3) shardsMapping(uint256,address)  
Which of these methods do you want to invoke (press 0 to exit)?
```

Figura 4.6: Invoke Smart Contract's Method

3. Consult your Smart Contract in your local database

Per semplificare l'utilizzo dell'applicazione, è stato introdotto un meccanismo che permette all'utente di consultare facilmente la lista degli Smart Contract presenti nel proprio database. Nel caso in cui non sia stato effettuato il deploy di alcun Smart Contract, verrà visualizzato un messaggio che segnala l'assenza di contratti disponibili. In caso contrario, l'elenco completo degli Smart Contract disponibili sarà restituito all'utente. Per ciascuno di questi viene specificato il *Contract Name*, il *Contract Address* e lo *Shard Address* (Figura 4.7).

```
Handle option 'Option 3: Consult your Smart Contract in your local database.'  
  
1 -----  
Contract name: OnChain  
Contract address: 0x07a457d878BF363E0Bb5aa0B096092f941e19962  
Shard address: http://localhost:8547  
  
2 -----  
Contract name: Owner  
Contract address: 0x07a457d878BF363E0Bb5aa0B096092f941e19962  
Shard address: http://localhost:8548
```

Figura 4.7: Consult your Smart Contract in your local database

4. Delete a Smart Contract from your local database

L'utente può decidere di eliminare dal suo database locale uno Smart Contract che non utilizza da tempo (Figura 4.8).

```
Handle option 'Option 4: Delete Smart Contract from your local database.'  
  
1 -----  
Contract name: OnChain  
Contract address: 0x07a457d878BF363E0Bb5aa0B096092f941e19962  
Shard address: http://localhost:8547  
  
2 -----  
Contract name: Owner  
Contract address: 0x07a457d878BF363E0Bb5aa0B096092f941e19962  
Shard address: http://localhost:8548  
  
Which one do you want to delete from your local database (press 0 to exit)? 2  
Smart Contract selected:  
Contract name: Owner  
Contract address: 0x07a457d878BF363E0Bb5aa0B096092f941e19962  
Shard address: http://localhost:8548  
  
Do you want proceed with the deletion (Y/N)?  
y  
Successful deletion.
```

Figura 4.8: Delete a Smart Contract from your local database

5. Change password

L'utente ha la possibilità di cambiare la propria password corrente. Per farlo è necessario inserire prima la password attuale e poi la nuova password desiderata, come si riporta in Figura 4.9. Questo processo di verifica della password attuale serve a garantire che solo l'utente legittimo sia in grado di modificare la propria password e impedire l'accesso non autorizzato da parte di altri utenti.

```
Handle option 'Option 5: Change password'

Do you want to change your password (Y/N)?
y
Old password:
New password:
Confirm new password:
```

Figura 4.9: Change password

6. Logout

L'utente può, infine, scegliere di effettuare il log out (Figura 4.10) e viene riportato alla schermata iniziale illustrata nella sezione precedente ("Registration and Log in").

```
Handle option 'Option 6: Logout.'

1 -- Register.
2 -- Log in.
3 -- Exit.
Enter your choice: □
```

Figura 4.10: Logout

4.5 Testing

4.5.1 Dynamic testing

Nella fase di testing dinamico sono state identificate e gestite le varie eccezioni e problematiche cercando sempre di tenere fede al principio *Fix security issues correctly* definito nel catalogo OWASP.

4.5.2 Static Testing

Sebbene il testing di tipo dinamico sia fondamentale per capire il comportamento del programma a runtime, questo non rappresenta l'unica tipologia di Software Testing che occorre considerare.

In particolare, è stato effettuato il testing statico del codice dell'On-Chain Manager implementato nel linguaggio di programmazione Solidity (versione 2 di On-Chain-Manager.sol riportata in A.2).

Questo tipo di testing è essenziale al fine di individuare vulnerabilità sulla sicurezza del codice legate alla sintassi e alla logica implementata e prevenire attacchi.

A tal proposito si è deciso di utilizzare *Solhint*, una libreria per l'analisi statica di codice scritto in Solidity. In particolare, il testing è stato effettuato tramite l'IDE Remix spuntando la casella "Select all" che permette di tenere in conto di tutti i consigli forniti da Solhint.

Il tool ha individuato warning relativi all metodo *findString(address[] memory array, address_string)* il quale utilizzava un ciclo for per scorrere la lista degli smart contract.

Tale lista è implementata tramite un array dinamico la cui lunghezza dipende dal numero degli smart contract deployati su un determinato shard.

Come suggerisce Solhint, i loops che non hanno un numero fisso di iterazioni, devono essere utilizzati con cautela. Infatti, a causa del gas limit dei blocchi, le transazioni possono consumare solo una certa quantità di gas e, nel caso di operazioni su array dinamici, il numero di iterazioni potrebbe crescere al di sopra di questo limite.

Per tale ragione, si è scelto di eliminare il ciclo for implementando un mapping che non utilizza come tipo del valore una lista ma un altro mapping.

Questa terza implementazione è riportata in A.3 e permette di effettuare il controllo di validità della copia shard-indirizzo senza utilizzare un loop ma sfruttando l'ottimizzazione della struttura mapping che funziona come un dizionario.

In definitiva, il mapping implementato prende come chiave l'indirizzo di uno shard e come valore un altro mapping che a sua volta ha come chiavi gli indirizzi degli smart contract deployati sui relativi shard.

I valori del secondo mapping sono dei valori booleani che servono unicamente per supportare questa struttura di mapping dentro ad un altro mapping.

Un altro warning suggerito da Solhint, sconsiglia l'utilizzo delle stringhe, le quali, si comportano ancora una volta come array dinamici.

Questo significa che all'interno di un tipo stringa è possibile memorizzare dati di lunghezza arbitraria dove elemento di una variabile di tipo stringa è un carattere della stringa.

Per questo è stato deciso di sostituire i tipi stringa del mapping con degli interi che, invece di indicare l'indirizzo completo di uno shard, memorizzano solo le porte (si trascura quindi "<http://localhost:>" che è uguale per tutti gli shard).

Questa modifica è riportata in A.4 e rappresenta la versione definitiva dell'On-Chain Manager.

A

Appendice: Evoluzione dell'On-Chain Manager

```

1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.4.22 <0.9.0;
3
4 contract OnChainManager {
5     mapping(string => address[]) public shardsMapping;
6     mapping (string => int) public shardsBalance;
7     event AddedToDict(bool result);
8
9     //add element to dictionary
10    function addToDictionary(string memory shardAddress, address
11        contractAddress) public{
12        shardsMapping[shardAddress].push(contractAddress);
13        shardsBalance[shardAddress]++;
14        emit AddedToDict(true);
15    }
16
17    //get list of contract addresses of a given shard
18    function getAddressList(string memory shardAddress) public view
19        returns(address[] memory){
20        return shardsMapping[shardAddress];
21    }
22
23    //get balance of a given shard
24    function getBalance(string memory shardAddress) public view
25        returns (int){
26        return shardsBalance[shardAddress];
27    }
28
29    function compareStrings(address a, address b) public pure returns
30        (bool) {
31        return (keccak256(abi.encodePacked((a))) == keccak256(abi.
32            encodePacked((b))));
33    }
34
35    //find string inside list
36    function findString(address[] memory array,address _string)
37        internal pure returns (bool){
38        for (uint i = 0; i < array.length; i++) {
39            address stringToFind = array[i];
40            bool exists = compareStrings(stringToFind, _string);
41            if(exists == true) {
42                return true;
43            }
44        }
45        return false;
46    }
47
48    //check if couple shard-address_of_contract exists
49    function isValidAddress(string memory shard, address
50        smartContractAddress) public view returns (bool){
51        for(uint i=0; i<2; i++){
52            if (findString(shardsMapping[shard], smartContractAddress
53                )==true)
54                return true;
55        }
56        return false;
57    }
58 }
```

```

1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.4.0.0;
3 pragma experimental SMTChecker;
4
5 contract OnChainManager {
6     mapping(uint => mapping(address => bool)) public shardsMapping;
7     event AddedToDict(bool result);
8
9     //add element to dictionary
10    function addToDictionary(uint shardAddress, address
11        contractAddress) public {
12        shardsMapping[shardAddress][contractAddress] = true;
13        emit AddedToDict(true);
14    }
15
16    function compareStrings(address a, address b) public pure returns
17        (bool) {
18        return (keccak256(abi.encodePacked((a))) == keccak256(abi.
19            encodePacked((b))));
20    }
21
22    function findString(address[] memory array, address _string)
23        internal pure returns (bool){
24        for (uint i = 0; i < array.length; i++) {
25            address stringToFind = array[i];
26            bool exists = compareStrings(stringToFind, _string);
27            if(exists == true) {
28                return true;
29            }
30        }
31        return false;
32    }
33
34    function isValidAddress(string memory shard, address
35        smartContractAddress) public view returns (bool){
36        for(uint i=0; i<2; i++){
37            if (findString(shardsMapping[shard], smartContractAddress
38                )==true)
39                return true;
40            }
41            return false;
42        }
43    }

```

Figura A.2: Seconda versione di *On-Chain-Manager.sol*

```

1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.4.0.0;
3
4 contract OnChainManager {
5     mapping(string => mapping(address => bool)) public shardsMapping;
6     event AddedToDict(bool result);
7
8     //add element to dictionary
9     function addToDictionary(string memory shardAddress, address
10         contractAddress) public {
11         shardsMapping[shardAddress][contractAddress] = true;
12         emit AddedToDict(true);
13     }
14
15     //check if smart contract is deployed inside
16     function isValidAddress(string memory shardAddress, address
17         contractAddress) public view returns (bool){
18         if(shardsMapping[shardAddress][contractAddress]==true)
19             return true;
20         else
21             return false;
22     }
23 }
```

Figura A.3: Terza versione di *On-Chain-Manager.sol* con mapping modificato

```

1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity >=0.4.0;
3
4 contract OnChainManager {
5     mapping(uint => mapping(address => bool)) public shardsMapping;
6     event AddedToDict(bool result);
7
8     //add element to dictionary
9     function addToDictionary(uint shardAddress, address
10         contractAddress) public {
11         shardsMapping[shardAddress][contractAddress] = true;
12         emit AddedToDict(true);
13     }
14
15     //check if smart contract is deployed inside
16     function isValidAddress(uint shardAddress, address
17         contractAddress) public view returns (bool){
18         if(shardsMapping[shardAddress][contractAddress]==true)
19             return true;
20         else
21             return false;
22     }
23 }
```

Figura A.4: Quarta versione di *On-Chain-Manager.sol* con tipi stringa sostituiti