BRIGHTNODE

Supporto alla scelta della blockchain più adatta agli scopi di Unblended

Durante l'esecuzione dell'attività sono state spese circa 4 ore in call e riunioni con il cliente per acquisire tutte le informazioni necessarie per fornire gli elementi decisionali richiesti. In seguito all'acquisizione di tali informazioni sono state proposte due soluzioni architetturali. Nella prima soluzione è stato suggerito un approccio fortemente basato su Merkle Tree in cui tutti gli elementi da notarizzare potevano essere "compressi" crittograficamente in un'unica Merkle Root da notarizzarsi su una blockchain pubblica con cadenza giornaliera o con altra periodicità.
La soluzione presenta l'enorme vantaggio di comprimere la parte on-chain dei dati in modo radicale lasciando ai database off.chain il compito di contenere le proof ed i dati, questo avrebbe permesso di utilizzare direttamente una blockchain maggiore come Bitcoin o Ethereum per la notarizzazione degli eventi di filiera. Tuttavia, alla luce di ulteriori conversazioni con il cliente sono emerse delle difficoltà nell'implementare la parte off-chain di tale soluzione e si è preferito optare per una diversa soluzione basata su smart contract e sono state calcolate le spese di gas necessarie per ogni transazione, sulla base delle quali si sono individuate le blockchain Ethereum equivalenti Polygon e Fantom come possibili candidate.

Best practice sviluppo smart contract

Nel corso di diverse riunioni e call si è accompagnato team del cliente allo sviluppo dello smart contract ed all'adozione delle best practice tra cui a titolo esemplificativo e non esaustivo:

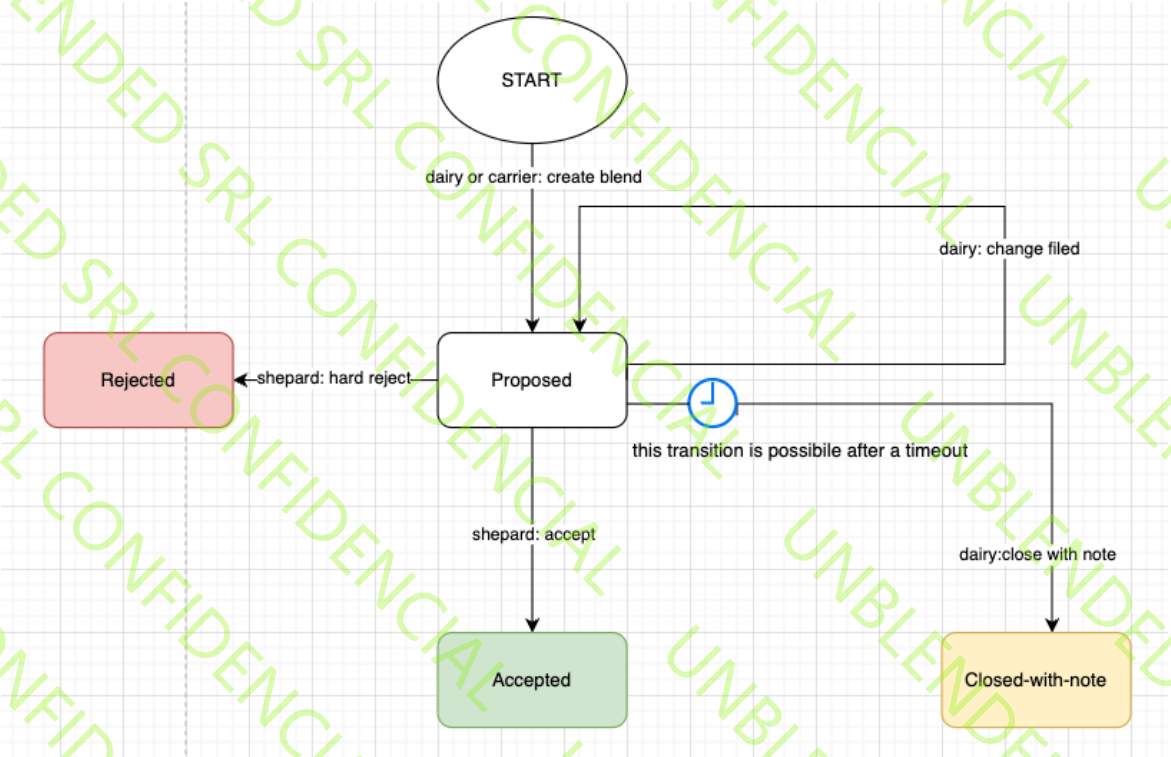
- Predisposizione di un progetto Hardhat
- Predisposizione di una suite di test automatici basati su libreria javascript chai
- Scrittura di alcuni testcase di esempio
- Analisi delle spese di gas per ogni transazione e decisioni basate sulle quantità
- In generale preferire la semplicità per evitare errori:
    - Ensure the contract logic is simple
    - Modularize code to keep contracts and functions small

- Use already-written tools or code where possible (eg. don't roll your own random number generator)
- Prefer clarity to performance whenever possible
- Only use the blockchain for the parts of your system that require decentralization

Consulenza e assistenza nella progettazione dello smart contract

Si è assistito il team del cliente nella progettazione dello smart contract ed in una prima iterazione insieme al team si sono ottenute le seguenti specifiche successivamente migliorate in funzione dei vari progressi durante lo sviluppo.

Diagramma Stati/Transizioni per un Blend



Transactions

Il contratto deve consentire l'aggiunta in modalità append-only di nuovi record di tipo Blend

appendBlend(Parents, Quantity, Unit, TotalQty, TrantisionType, TransitionParamsKeys, TransitionParamsValues)

Il campo **timestamp** viene aggiunto in modo automatico secondo il tempo di blocco. Il blend si pone nello stato **Proposed**. Nel caso in esame **l'Owner** è il firmatario di questa transazione.

A partire dallo stato Proposed sono possibili:

- **acceptBlend**(id)

  Il blend se si trova nello stato Proposed deve essere approvato dalle parti che possiedono il blend parent. Se abbiamo solo un blend.Parent è facile identificare chi deve firmare acceptBlend. Se i parent sono 2 (cosa possibile) normalmente uno dei parent blend appartiene al pastore mentre l'altro è sempre chi ha creato il blend corrente. In tal caso acceptBlend va firmato dal pastore.

- **rejectBlend**(id)

  Come per acceptBlend, questa può essere azionata dall'owner di un parent del blend in questione. Secondo i flussi analizzati sarebbe sempre un pastore che rifiuta di accettare un blend.

A partire dallo stato proposed, l'owner del blend può modificarne dati e attributi in qualsiasi momento

updateBlend(id, payload)

Se il timeout di attesa nello stato proposed scade, l'owner del blend può editarlo e chiuderlo con una nota

closeWithNote(id, note, payload)

Eventi

Grazie all'emissione e successiva analisi degli eventi sarà possibile a posteriore ricostruire tutta l'evoluzione degli stati di un Blend, dalla sua creazione fino alla sua accettazione o chiusura.

- **BlendCreated**, quando un blend viene creato
- **BlendAccepted**, quando un blend viene accettato
- **BlendRejected**, quando un blend viene rifiutato
- **BlendUpdated**, quando un blend viene aggiornato
- **BlendClosedWithNote**, quando un blend viene chiuso con una nota dopo il timeout

Tutti gli eventi avvengono alla timestamp del blocco in cui vengono emessi

Tutti gli eventi portano come minimo le seguenti informazioni

- Id del blend
- Address di chi ha scatenato la transazione

Query

Il contratto deve consentire di conoscere il blend sulla cima ed inoltre deve consentire di ricavare un blend a partire dal suo ID. Deve anche restituire la lunghezza del counter di blend che viene incrementato ad ogni nuovo inserimento. La funzione blendById permette di esplorare all'indietro il grafo dei blend a partire da un qualunque blend id di partenza.

lastBlend( ) -> Blend

blendById(uint id) -> Blend

getCounter( ) -> uint

Quality assurance sulla progettazione dello smart contract per versione beta

Nel corso dell'attività si sono analizzati i codici sorgenti Blends.sol e IdentityRegistry.sol con uno static analyzer e poi si sono effettuate le seguenti verifiche:

Category	Description	Results
Reentrancy	Intra- and inter-function reentrancy attacks and potentially faulty solutions to them.	No cases of reentrancy found
Oracle Manipulation	Manipulation of external data providers and potential solutions to oracle security issues.	No action required but read below considerations. Blends.sol strongly relies on external sources of data because it is a supply chain tracking contract. That being said the contract assumes the good faith of declaration done by the participants and there is no way to mitigate such malicious behavior other than what is described in the business logic. In particular the assertions are always required to be signed by the relevant parties. Another point of vulnerability is the presence of the Identity

		Registry as single point of authority to assign or map public keys to real identities. Compromising the Registry would have catastrophic consequences but it is accepted as a trade off of the whole system that the Registry is managed by a trusted party with full liability of their operations.
Frontrunning	A definition and taxonomy around frontrunning and related attacks.	Not applicable as only authorized identities can write on the Blends storage and in any case there is no incentive to insert blends records before other participants
Timestamp Dependence	Attacks relating to the timing of a transaction.	Blocktime is used to timestamp records. Timestamping of records is essential in a supply chain solution. The incentive for a miner to forge these data is practically zero while on the other hand the use of an off-chain source of time proof would be more problematic and prone to manipulation.
Insecure Arithmetic	Integer overflows and underflows.	Not applicable, no math is involved in these contracts

Denial of Service	Denial of service attacks through unexpected reverts and the block gas limit.	Not applicable
Griefing	Attacks relating to bad faith players around a smart contract system.	Not applicable
Force Feeding	Forcing Ether to be sent to smart contracts to manipulate balance checks.	Not applicable
Deprecated/Historical	Attacks that are part of Ethereum's history and vulnerabilities that have been fixes on a (Solidity) compiler level.	Action SUGGESTED. upgrade solc to latest version available

Nessuna azione risulta richiesta in modo mandatorio, solo un'azione risulta SUGGERITA rispetto alle vulnerabilità analizzate sopra.

L'attività di code review è poi proseguita con il suggerimento ed implementazione di numerosi testcase per la verifica della correttezza dell'implementazione rispetto ai requisiti funzionali. Tali testcase sono disponibili nella cartella /test del progetto. I testcase sono 56 per un totale di oltre 700 LOC per testare circa 270 LOC di codice degli smart contract. Si riporta tabella di copertura percentuale generata con tool di code coverage e si SUGGERISCE di agire per portare il code coverage al 100% (attualmente 96%).

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	96.43	80.43	94.12	96.43	
Blends.sol	94.44	75	90.91	94.44	178,179
IdentityRegistry.sol	100	88.89	100	100	
All files	96.43	80.43	94.12	96.43	

Disclaimer

Gli smart contract vengono analizzati in modo proporzionale alla complessità del codice ed alle possibili perdite economiche di un eventuale malfunzionamento. E' molto importante considerare che questo code review ha come scopo la verifica del codice degli smart contract per una versione beta e non per una versione da rilasciare in produzione per la quale si raccomanda in ogni caso la verifica e l'audit da parte di un ulteriore sviluppatore terzo o agenzia specializzata in smart contract security.

Sono state analizzate le più note e importanti vulnerabilità secondo le best practice correnti tuttavia è importante precisare che il code review ed il testing non "dimostra" mai l'assenza di bug o malfunzionamenti ma al contrario ne può solo dimostrare la presenza quando un testcase fallisce o un'evidenza di errore viene trovata, quindi in ogni caso il presente code review non deve essere considerato una garanzia di corretto funzionamento dello smart contract.