FundETH
Smart Contract System and Security Design Document

Introduzione

Contesto e Panoramica

La piattaforma **FundETH** è un sistema di crowdfunding decentralizzato basato su blockchain per la creazione, il finanziamento e la gestione di campagne di raccolta fondi. L'obiettivo principale è garantire la trasparenza dei flussi finanziari, la sicurezza dei fondi (tramite meccanismi di escrow) e la rimozione degli intermediari tradizionali, permettendo ai creatori di interagire direttamente con i sostenitori.

L'uso della tecnologia blockchain consente di superare i limiti delle piattaforme di crowdfunding tradizionali (come le commissioni elevate e la gestione centralizzata dei fondi), che sono spesso esposte a politiche arbitrarie o rischi di blocco dei capitali. Con FundETH, ogni campagna è gestita da uno **Smart Contract** che automatizza la raccolta delle donazioni in Ether e regola rigidamente le condizioni di prelievo (solo al raggiungimento del target) o di rimborso (in caso di mancato raggiungimento), mentre i metadati descrittivi e le immagini sono archiviati in modo distribuito su **IPFS** (InterPlanetary File System) per garantire efficienza e decentralizzazione.

Obiettivo del Documento

Questo documento analizza le scelte di progettazione, l'architettura tecnica e i meccanismi di sicurezza utilizzati nello sviluppo degli smart contract di FundETH. Verranno descritti i contratti implementati, i **Design Pattern** adottati per garantire la sicurezza delle transazioni finanziarie (come la prevenzione della reentrancy), le interazioni tra i componenti del sistema e le strategie di mitigazione dei rischi per proteggere i fondi degli utenti.

Smart Contract

Panoramica

Il cuore della piattaforma FundETH risiede in un unico smart contract monolitico denominato `CrowdFunding.sol`. A differenza di architetture modulari complesse che dividono logica e dati su più contratti, per FundETH è stata scelta una struttura consolidata e diretta per ottimizzare i costi di gas (deployment e interazione) e semplificare l'audit di sicurezza. Il contratto è sviluppato in Solidity (versione ^0.8.24) e gestisce l'intero ciclo di vita finanziario delle campagne.

CrowdFunding.sol

Ruolo: Gestione del ciclo di vita della campagna, custodia dei fondi (escrow), logica di contribuzione e distribuzione.

Il contratto funge da "cassaforte programmabile". Non possiede privilegi amministrativi centralizzati (Admin keys) che possano censurare o bloccare fondi arbitrariamente; la logica è interamente dettata dalle regole matematiche definite nel codice.

Funzioni Principali:

- **`createCampaign`:** Permette a qualsiasi utente di inizializzare una raccolta fondi. La funzione configura la `struct Campaign`, impostando il target finanziario, la scadenza temporale e i metadati (incluso il riferimento IPFS per l'immagine). Restituisce l'ID univoco della campagna.
- **`donateToCampaign`:** Funzione `payable` che accetta Ether dai sostenitori. Aggiorna il saldo della campagna (`amountCollected`) e registra il donatore e l'importo nelle strutture dati interne (`donators` e `donations`) per garantire la tracciabilità e il diritto all'eventuale rimborso.
- **`withdrawFunds`:** Permette al creatore della campagna (Owner) di prelevare i fondi. Questa funzione è eseguibile solo se il target è stato raggiunto entro la scadenza e i fondi non sono già stati riscossi.
- **`refund`:** Funzione di sicurezza per i donatori. Se la campagna scade senza aver raggiunto il target, questa funzione permette ai sostenitori di recuperare l'importo esatto donato, prevenendo il blocco dei capitali nel contratto.
- **`getCampaigns` / `getDonators`:** Funzioni di tipo `view` (lettura) che permettono al frontend di recuperare lo stato delle campagne e la lista dei sostenitori senza costi di gas.

Design Pattern

L'analisi del codice di `CrowdFunding.sol` evidenzia l'adozione di pattern di sicurezza standard del settore per mitigare vulnerabilità note e garantire la robustezza logica.

Guard Check Pattern

- **Contesto:** `createCampaign`, `donateToCampaign`, `withdrawFunds`, `refund`.
- **Descrizione:** Il contratto utilizza clausole `require()` all'inizio delle funzioni per validare gli input e lo stato del contratto prima di eseguire qualsiasi logica critica.
    - In `createCampaign`, si verifica che la scadenza sia nel futuro.
    - In `donateToCampaign`, si verifica che la campagna non sia scaduta.
    - In `withdrawFunds`, si verificano tre condizioni: identità del richiedente (Owner), successo della campagna (Target raggiunto) e stato dei fondi (Non ancora prelevati).
- **Vantaggi:** Garantisce che le transazioni falliscano rapidamente (fail-fast) risparmiando gas all'utente in caso di condizioni non valide e protegge l'integrità logica del sistema.

Checks-Effects-Interactions Pattern

- **Contesto:** `withdrawFunds`, `refund`.

- **Descrizione:** Questo è il pattern di sicurezza più critico implementato in FundETH per prevenire attacchi di *Reentrancy*. L'ordine delle operazioni nel codice è rigoroso:
    1. **Checks:** Vengono verificati i requisiti (es. `require(!campaign.claimed)`).
    2. **Effects:** Viene aggiornato lo stato del contratto *prima* di inviare Ether.
        - In `withdrawFunds`: `campaign.claimed = true` viene impostato prima del trasferimento.
        - In `refund`: `campaign.donations[i] = 0` (azzeramento del saldo) avviene prima del trasferimento.
    3. **Interactions:** Solo alla fine viene eseguita la chiamata esterna `.call{value: ...}` per inviare i fondi.
- **Vantaggi:** Se un attaccante tentasse di rientrare nella funzione durante la chiamata di trasferimento, troverebbe lo stato già aggiornato (es. saldo azzerato o campagna già riscossa), rendendo inefficace l'attacco.

Pull over Push (Withdrawal Pattern)

- **Contesto:** `refund`.
- **Descrizione:** Invece di inviare automaticamente i rimborsi a tutti i donatori nel momento in cui una campagna scade (operazione *Push*), il contratto attende che sia il singolo donatore a richiedere il proprio rimborso (operazione *Pull*).
- **Vantaggi:**
    - **Prevenzione DoS (Denial of Service):** Se il contratto tentasse di rimborsare tutti i donatori in un ciclo unico, la transazione potrebbe fallire superando il limite di gas del blocco (Block Gas Limit), bloccando i fondi per sempre.
    - **Sicurezza:** Sposta il costo del gas sull'utente beneficiario e isola i fallimenti (se il trasferimento a un utente fallisce, non blocca gli altri).

Interazioni

Poiché FundETH opera su un singolo contratto, le interazioni avvengono principalmente tra attori esterni (EOA - Externally Owned Accounts) e il contratto stesso.

1. **Creazione:** L'utente invia una transazione a `createCampaign`. Il contratto stanzia lo spazio di archiviazione nella blockchain.
2. **Finanziamento:** Il donatore invia Ether a `donateToCampaign`. Il contratto incapsula i fondi nel proprio saldo (address(this).balance) e aggiorna i registri interni.
3. **Risoluzione (Successo):** L'Owner chiama `withdrawFunds`. Il contratto verifica lo stato e trasferisce il saldo accumulato all'Owner.
4. **Risoluzione (Fallimento):** Il Donatore chiama `refund`. Il contratto cerca il saldo del donatore, lo azzera e restituisce gli Ether al chiamante.

Scelte Architetturali

Storage on-chain vs off-chain Per bilanciare decentralizzazione ed efficienza, è stata fatta una scelta architetturale ibrida.

- **On-chain (Solidity):** Solo i dati critici per la logica finanziaria (Owner, Target, Deadline, Saldi, Lista Donatori) sono salvati nella `struct Campaign`.
- **Off-chain (IPFS):** Le informazioni descrittive pesanti, specificamente l'immagine, non sono salvate on-chain. Il contratto memorizza solo una stringa `image` contenente l'URL o l'Hash IPFS.
- **Motivazione:** Questa scelta riduce drasticamente i costi di storage sulla Ethereum Virtual Machine (EVM), rendendo la creazione delle campagne economicamente accessibile.

Gestione degli Ether: `call` vs `transfer` Nel codice è stata utilizzata la funzione di basso livello `.call{value: ...}("")` per i trasferimenti di Ether, accompagnata da un controllo sul valore booleano di ritorno (`require(sent, ...)`).

- **Motivazione:** Questa è la pratica raccomandata nelle versioni moderne di Solidity rispetto a `transfer()` o `send()`, poiché protegge il contratto da futuri cambiamenti nei costi del gas delle operazioni (opcode) di Ethereum e permette l'invio di Ether anche a smart contract wallet complessi.

Sicurezza e Mitigazione dei Rischi

Panoramica

La sicurezza è l'elemento cardine nella progettazione dello smart contract di **FundETH**, poiché la piattaforma gestisce transazioni finanziarie reali in criptovaluta. Una singola vulnerabilità nel codice potrebbe portare alla perdita irreversibile dei fondi dei donatori o al blocco dei capitali raccolti. Di seguito vengono analizzate le principali minacce identificate e le contromisure tecniche adottate nel codice `CrowdFunding.sol`.

Protezione dell'Accesso e Gestione delle Autorizzazioni

Per evitare accessi non autorizzati e furti di fondi, la piattaforma implementa un rigido controllo degli accessi basato sull'identità dell'indirizzo che interagisce con il contratto.

A differenza di sistemi complessi con ruoli amministrativi centralizzati (come `Admin` o `Issuer` descritti nel documento di riferimento), FundETH adotta un approccio **decentralizzato owner-centric**:

- **Ruolo Owner:** Ogni campagna ha un unico proprietario (`campaign.owner`), definito immutabilmente al momento della creazione tramite la funzione `createCampaign`.

- **Controllo dei Privilegi:** Nella funzione `withdrawFunds`, il contratto verifica esplicitamente che `msg.sender` (chi sta chiamando la funzione) coincida con `campaign.owner`.
    - *Codice:* `require(msg.sender == campaign.owner, "Solo il creatore puo prelevare");`.
- **Isolamento:** Questo meccanismo garantisce che solo il creatore legittimo possa accedere ai fondi raccolti, e che nessun altro utente (incluso chi ha deployato il contratto originale) possa dirottare il prelievo.

Prevenzione di Attacchi ai Contratti Smart

Gli smart contract Solidity sono esposti a vettori di attacco specifici. FundETH mitiga i rischi più comuni attraverso pattern architetturali e controlli logici.

Mitigazione Reentrancy Attack L'attacco di rientranza è una delle minacce più gravi, dove un contratto malevolo chiama ricorsivamente una funzione di prelievo prima che il saldo venga aggiornato.

- **Soluzione:** FundETH applica rigorosamente il pattern **Checks-Effects-Interactions**.
- **Implementazione:** Nelle funzioni `withdrawFunds` e `refund`, lo stato della campagna viene modificato (es. `campaign.claimed = true` o `campaign.donations[i] = 0`) *prima* di eseguire la chiamata esterna `.call` per trasferire gli Ether. Se un attaccante tentasse di rientrare nella funzione, troverebbe lo stato già alterato (fondi già prelevati o saldo azzerato) e la transazione fallirebbe.

Mitigazione Denial of Service (DoS) con Gas Limit Un rischio comune nei contratti di crowdfunding è tentare di rimborsare tutti i donatori in un unico ciclo `for` automatico. Se il numero di donatori è elevato, il costo del gas supererebbe il limite del blocco, bloccando i fondi per sempre.

- **Soluzione:** Adozione del pattern **Pull over Push**.
- **Implementazione:** La funzione `refund` non invia denaro automaticamente. È il singolo donatore che deve chiamare la funzione per prelevare la propria quota. Questo isola le transazioni: il fallimento o l'alto costo di gas di un utente non influenza gli altri, garantendo la scalabilità del sistema.

Protezione da Overflow/Underflow Aritmetici Errori matematici potrebbero permettere a un attaccante di prelevare più fondi del dovuto.

- **Soluzione:** L'utilizzo della versione di Solidity `^0.8.24` integra nativamente i controlli sugli overflow aritmetici. Non è necessario utilizzare librerie esterne come `SafeMath`, poiché il compilatore restituisce automaticamente errore se un'operazione supera i limiti del tipo `uint256`.

Sviluppi e Miglioramenti Futuri

Evoluzione della Piattaforma

FundETH è stata progettata con un'architettura essenziale per garantire la massima sicurezza nella sua prima versione (MVP). Tuttavia, per favorire un'adozione di massa e aumentare la flessibilità finanziaria, sono stati identificati diversi sviluppi futuri strategici.

Supporto Multi-Token (ERC-20) Attualmente, la piattaforma accetta solo la criptovaluta nativa (Ether), che è soggetta a forte volatilità. Un miglioramento critico sarà l'integrazione di token ERC-20, in particolare **Stablecoins** (come USDC o USDT). Questo permetterebbe ai creatori di campagne di ricevere fondi con valore stabile, proteggendo il capitale raccolto dalle fluttuazioni di mercato durante la durata della campagna.

Scalabilità e Riduzione Costi (Layer 2) Il costo del gas sulla mainnet di Ethereum può rappresentare una barriera all'ingresso per i piccoli donatori. Una futura migrazione o deploy parallelo su soluzioni di **Scaling Layer 2** (come Polygon, Arbitrum o Optimism) ridurrebbe drasticamente le commissioni di transazione, rendendo la piattaforma accessibile anche per micro-donazioni.

Smart Contract Aggiornabili (Proxy Pattern) Poiché il codice sulla blockchain è immutabile, correggere eventuali bug o aggiungere funzionalità in futuro richiederebbe il deploy di un nuovo contratto e la migrazione dei dati. L'adozione del **Proxy Pattern** (standard OpenZeppelin) permetterebbe di separare la logica dal deposito dei dati, consentendo aggiornamenti trasparenti della logica del contratto senza dover spostare i fondi o cambiare l'indirizzo della piattaforma.