UNIVERSITÀ
DI PARMA

Sicurezza negli smart contract cross-chain: sfide e possibili soluzioni

Seminario del corso di Linguaggi, Interpreti e Compilatori (a.a. 2024/25)

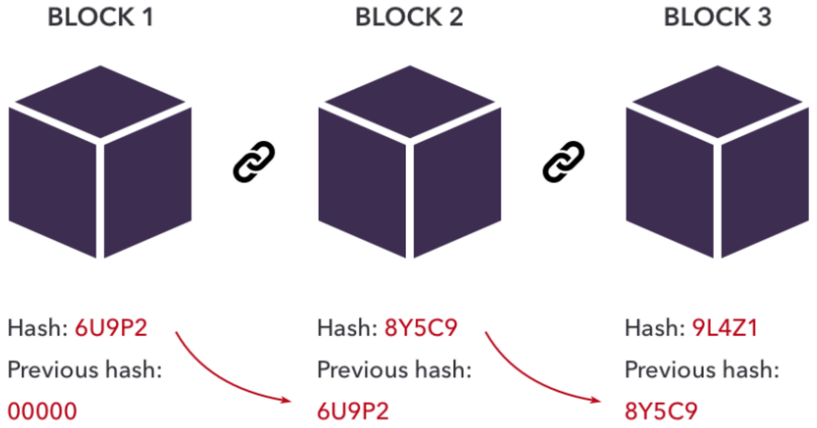
Merenda Saverio Mattia

Obiettivi del seminario

- ***Capire*** il funzionamento della tecnologia blockchain
- ***Esplorare*** le vulnerabilità cross-chain
- ***Scoprire*** come l'analisi statica può migliorare la sicurezza

Introduzione alla blockchain

- ***Registro digitale decentralizzato***
- ***Trasparente e immutabile***
- ***Esempi reali:***
  - Bitcoin e le transazioni monetarie [1]
  - Ethereum e gli smart contract [2]

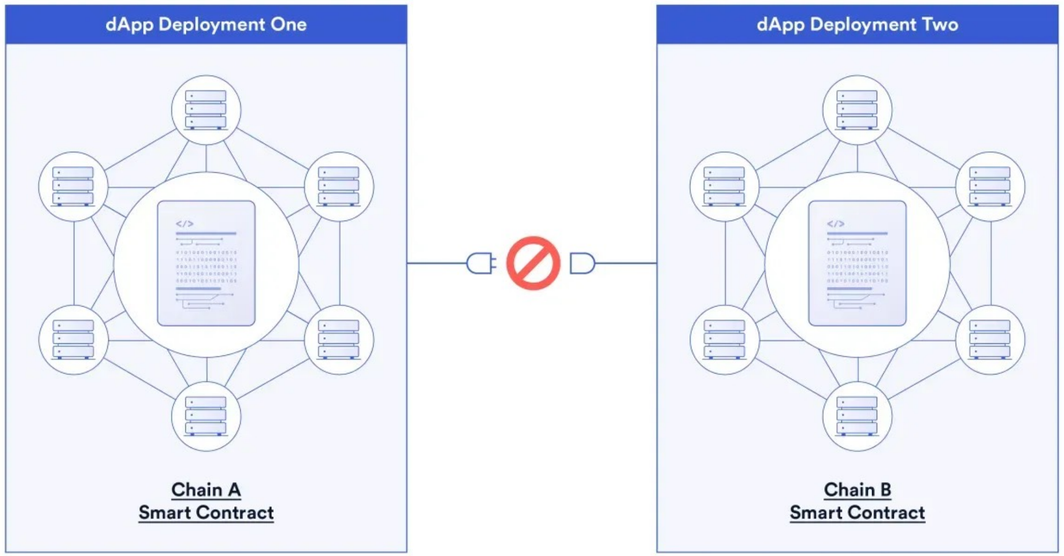


Che cosa sono gli smart contract?

- ***Programmi*** memorizzati su blockchain
- ***Esecuzione automatica*** quando vengono soddisfatte determinate condizioni
- ***Immutabilità*** e ***trasparenza***

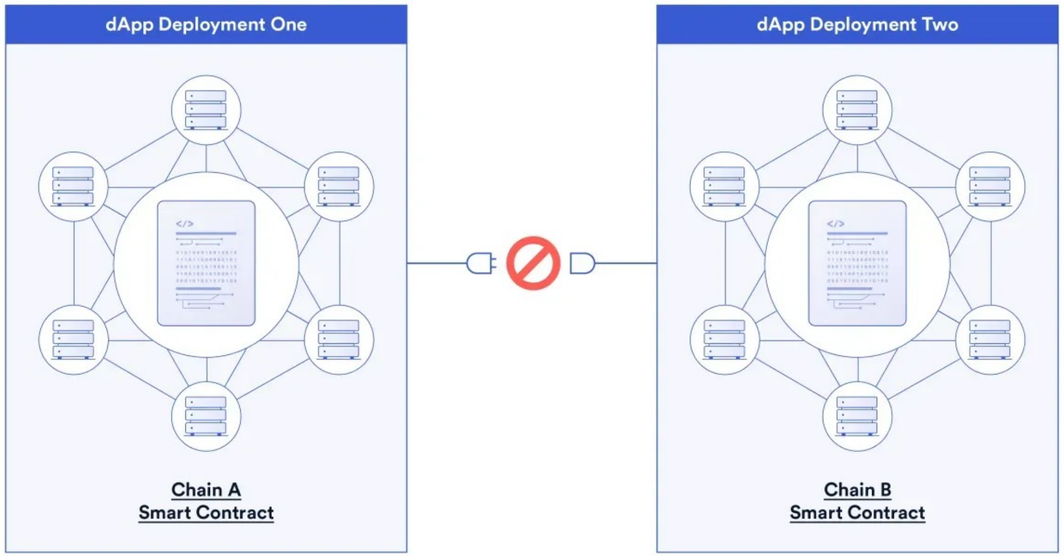
Sfida dell'interoperabilità

- Le blockchain non possono ***interagire direttamente***



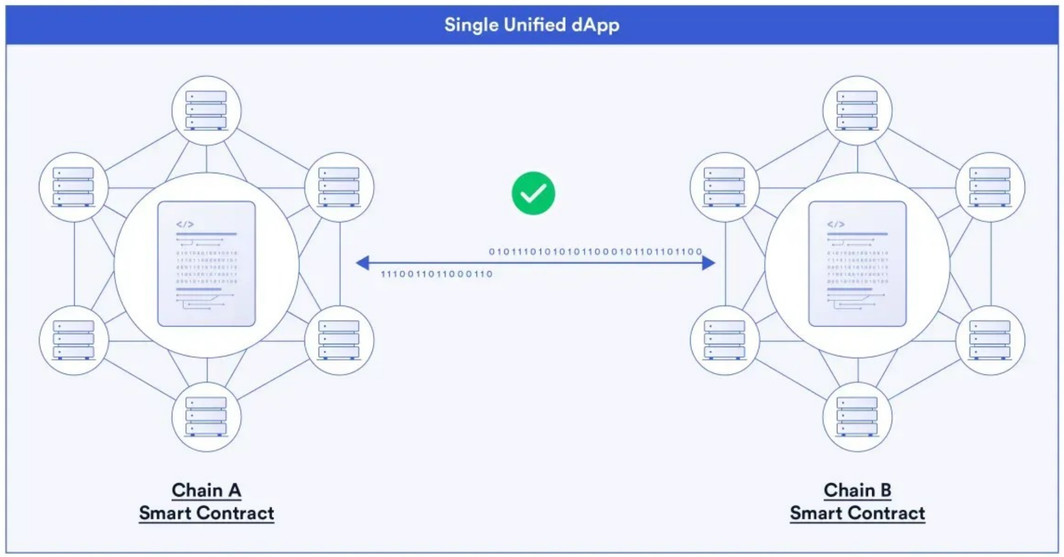
Sfida dell'interoperabilità

- Le blockchain non possono ***interagire direttamente***
- Soluzione: ***smart contract cross-chain (bridge)***



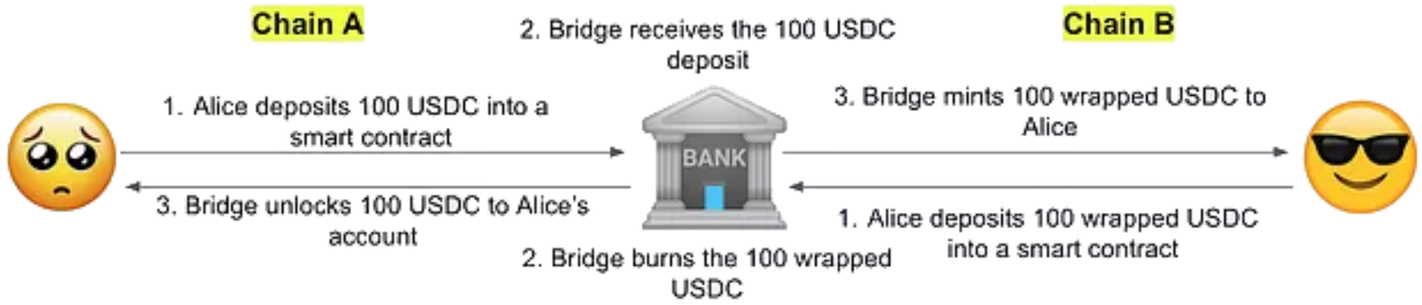
Sfida dell'interoperabilità

- Le blockchain non possono ***interagire direttamente***
- Soluzione: ***smart contract cross-chain (bridge)***



Funzionamento dei bridge

- ***Source chain***: blocca l'asset
- ***Relayer***: comunica l'operazione
- ***Destination chain***: crea una copia dell'asset



Superfici di attacco dei bridge

- ***Lato server***
  - Front-end phishing
  - Mishandling events
- ***Lato smart contract***
  - Problematic mint & fake burn
  - Vulnerabilità nel codice

Vulnerabilità nel codice

- ***Problemi logici***
  - Variabile inizializzata in un modo errato
- ***Reentrancy attack***

```
contract ReentrantContract {
  mapping (address => uint) private balances;

  function withdraw (uint amount) public {
    require(amount <= balances [msg.sender]);

    if(msg.sender.call.value(amount)( ))
      balances [msg.sender] -= amount;
  }
}
```

```
contract MaliciousContract {
  ReentrantContract reentrantContract;

  function attack() public {
    reentrantContract.withdraw(100);
  }

  function () public {
    reentrantContract.withdraw(100);
  }
}
```

SmartAxe[3]: una prima soluzione

- ***Rileva*** vulnerabilità cross-chain
  - Controllo degli accessi incompleto
  - Inconsistenza semantica tra le chain
- Effettua un'***analisi statica*** del bytecode degli smart contract

Come funziona SmartAxe

1. Analisi del flusso di controllo
2. Rilevamento problemi di accesso
3. Allineamento semantico
4. Analisi delle tracce vulnerabili

Come funziona SmartAxe

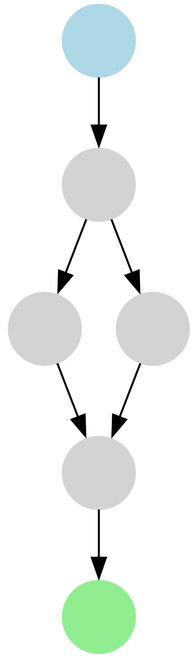
Analisi del flusso di controllo

1. Viene utilizzato ***SmartDagger*** [4]

2. ***Costruzione del CFG*** per i contratti della *source* e *destination chain*

   - I nodi rappresentano le istruzioni

   - Gli archi rappresentano l'execution flow



Esempio di Control-flow Graph

Come funziona SmartAxe

1. Analisi del flusso di controllo
2. **Rilevamento problemi di accesso**
3. Allineamento semantico
4. Analisi delle tracce vulnerabili

Come funziona SmartAxe

Rilevamento problemi di accesso

1. ***Estrazione*** dei vincoli di controllo degli accessi dal CFG
   - Esempio: `require`, `assert`, `if`, ecc.
2. ***Analisi*** delle risorse coinvolte
   - Esempio: lettura/scrittura variabili di stato, metodi interni, chiamate esterne, ecc.
3. Utilizzate ***tecniche probabilistiche*** per inferire i legami tra risorse e controlli

Come funziona SmartAxe

Rilevamento problemi di accesso

Tecniche probabilistiche

1. Per ogni risorsa, vengono calcolati tutti i ***percorsi raggiungibili*** dal punto di ingresso del contratto

2. Per ogni percorso, viene effettuata un'***associazione probabilistica*** basata sulla probabilità che un controllo di sicurezza *protegga la risorsa* [Tab. I]

   - Esempio: solo utenti autorizzati possono accedere ad una risorsa, le operazioni su una risorsa vengono eseguite in modo sicuro e corretto, ecc.

Come funziona SmartAxe

Rilevamento problemi di accesso

1. ***Estrazione*** dei vincoli di controllo degli accessi dal CFG
   - Esempio: `require`, `assert`, `if`, ecc.
2. ***Analisi*** delle risorse coinvolte
   - Esempio: lettura/scrittura variabili di stato, metodi interni, chiamate esterne, ecc.
3. Utilizzate ***tecniche probabilistiche*** per inferire i legami tra risorse e controlli
4. Costruzione del ***Data-flow Graph*** (DFG)

Come funziona SmartAxe

1. Analisi del flusso di controllo
2. Rilevamento problemi di accesso
3. **Allineamento semantico**
4. Analisi delle tracce vulnerabili

Come funziona SmartAxe

Allineamento semantico

1. Costruzione del ***cross-chain CFG*** (xCFG)

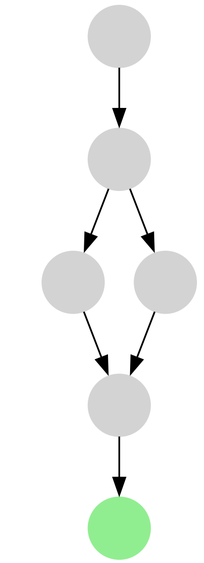
   - Identificati i punti di uscita della source chain e i punti di entrata della destination chain

   - Collegamento dei punti per creare un xCFG

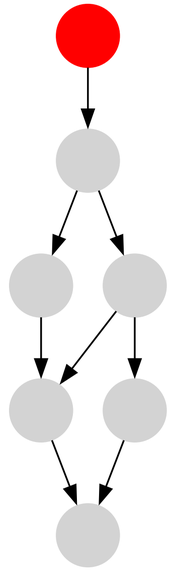
Come funziona SmartAxe

Allineamento semantico

1. Costruzione del ***cross-chain CFG*** (xCFG)



CFG della source chain

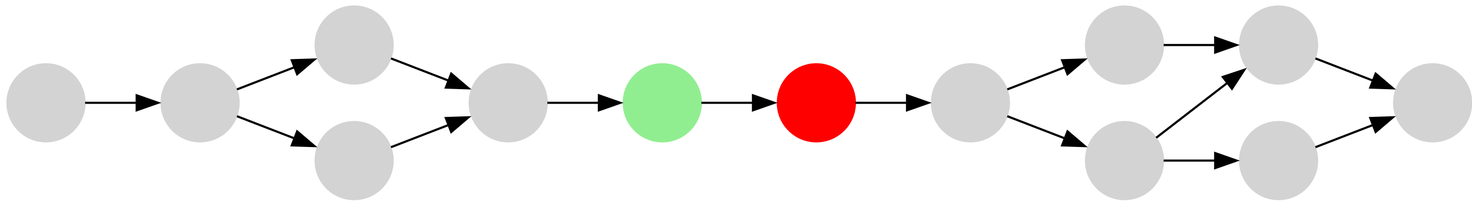


CFG della destination chain

Come funziona SmartAxe

Allineamento semantico

1. Costruzione del ***cross-chain CFG*** (xCFG)



Cross-chain CFG (xCFG)

Come funziona SmartAxe

Allineamento semantico

1. Costruzione del ***cross-chain CFG*** (xCFG)
   - Identificati i punti di uscita della source chain e i punti di entrata della destination chain
   - Collegamento dei punti per creare un xCFG
2. Costruzione del ***cross-chain DFG*** (xDFG)
3. ***Segnalazione*** di funzioni vulnerabili

Come funziona SmartAxe

1. Analisi del flusso di controllo
2. Rilevamento problemi di accesso
3. Allineamento semantico
4. **Analisi delle tracce vulnerabili**

Come funziona SmartAxe

Analisi delle tracce vulnerabili

1. Effettuata una ***taint analysis***

2. ***Tracciate le interazioni*** tra funzioni vulnerabili e variabili di stato globali

Valutazione sperimentale di SmartAxe

- Effettuato ***benchmark*** su due dataset
  - 223 contratti con bug inseriti manualmente
  - 1703 contratti reali presi da Chainspot [5]

$$\text{Precision (P)} = \frac{|T_P|}{|T_P| + |F_P|} = 84{,}9\,\%$$

$$\text{Recall (R)} = \frac{|T_P|}{|T_P| + |F_N|} = 89{,}7\,\%$$

Conclusioni

- ***Problemi***
  - Falsi positivi: limiti strutturali di SmartDagger
  - Falsi negativi: mancanza di analisi on-chain (e.g., accedere alle variabili di stato)
- ***Limiti***
  - Non sono supportate blockchain non EVM
  - Utilizzato metodo probabilistico per l'allocazione delle risorse
  - Il codice non è open source

Q&A

Sicurezza negli smart contract cross-chain: sfide e possibili soluzioni

Seminario del corso di Linguaggi, Interpreti e Compilatori
(a.a. 2024/25)

Merenda Saverio Mattia

UNIVERSITÀ
DI PARMA

Bibliografia

1. *Bitcoin:* https://www.bitcoinpaper.info/bitcoinpaper-html/
2. *Ethereum:* https://cryptodeep.ru/doc/paper.pdf
3. *SmartAxe:* https://dl.acm.org/doi/abs/10.1145/3643738
4. *SmartDagger:* https://dl.acm.org/doi/abs/10.1145/3533767.3534222
5. *ChainSpot:* https://chainspot.io/

Tabella I

Pattern	Condition	Probabilistic assignment
P1	$ControlFlowDependency(c, \{r\})$	$Association(c, p, r) = true(0.95)$
P2	$ControlFlowDependency(c, R) \vee r \in R$	$Association(c, p, r) = true(0.60)$
P3	$SameBlock(r_1, r_2)$	$Association(c, p_1, r_2) \xrightarrow{0.60} Association(c, p_1, r_1)$
P4	$SemanticCorrelation(r_1, r_2)$	$Association(c, p_2, r_2) \xrightarrow{0.70} Association(c, p_1, r_1)$
P5	$DataFlowDependency(r_1, r_2)$	$Association(c, p_2, r_2) \xrightarrow{0.80} Association(c, p_1, r_1)$