Capitolo 1

Dataset

Il dataset [**?**] utilizzato in questo progetto è un dataset disponibile pubblicamente sulla piattaforma HuggingFace una delle più importanti piattaforme per il Natural Language Processing. HF è un'infrastruttura open-source che fornisce accesso a una vasta gamma di modelli di deep learning pre-addestrati, tra cui alcuni dei più avanzati nel campo del NLP. Questo dataset contiene 106.474 smart contracts scritti in linguaggio Solidity un linguaggio di programmazione Touring completo divenuto lo standard per la scrittura di smart contracts su Ethereum. Ogni elemento nel dataset è composto da quattro elementi:

- **Address**: l'indirizzo del contratto
- **SourceCode**: il codice sorgente del contratto
- **ByteCode**: il codice bytecode del contratto
- **Slither**: il risultato dell'analisi statica del contratto con Slither, è un array che contiene valori che vanno da 1 a 5.

1.0.1 Vulnerabilità

In questo lavoro sono state prese in considerazione cinque classi di vulnerabilità diverse:

- Access-Control
- Arithmetic
- Other
- Reentrancy
- Safe
- Unchecked-Calls

Access-Control

Arithmetic

Other

Reentrancy

La Reentrancy è una classe di vulnerabilità presente negli SmartContracts che permette ad un malintenzionato di rientrare nel contratto in modo inaspettato durante l'esecuzione della funzione originale. Questa vulnerabilità può essere utilizzata per rubare fondi e rappresenta la vulnerabilità più impattante dal punto di vista di perdita di fondi a seguito di attacchi. Il caso più famoso di questo attacco che lo ha anche reso noto è il caso di The DAO, un contratto che ha subito un attacco di reentrancy che ha portato alla perdita circa sessanta milioni di dollari in Ether, circa il 14% di tutti gli Ether in circolazione all'epoca. Nonostante dal 2016 ad oggi siano stati fatti numerosi progressi nelle tecnologie e nelle misure di sicurezza questa vulnerabilità rimane comunque una delle minacce più pericolose per gli SmartContracts, poichè negli anni questo tipo di attacchi si è ripresentato notevole frequenza [**?**]. Un attacco di reentrancy può essere classificato in tre classi differenti:

- **Mono-Function**: la funzione vulnerabile è la stesa che viene chiamata più volte dall'attaccante, prima del completamento delle sue invocazioni precedenti. Questo è il caso più semplice di attacco reentrancy e di conseguenza il più facile da individuare.
- **Cross-Function**: questo caso è molto simile al caso di mono-function Reentrancy, ma in questo caso la funzione che viene chiamata dall'attaccante non è la stessa che fa la chiamata esterna. Questo tipo di attacco è possibile solo quando una funzione vulnerabile condivide il suo stato con un'altra funzione, risultando in un una situazione fortemente vantaggiosa per l'attaccante.
- **Cross-Contract**: questo tipo di attacco prende piede quando lo stato di un contratto è invocato in un altro contratto prima che viene correttamente aggiornato. Avviene solitamente quando più contratti condividono una variabile di stato comune e uno di loro la aggiorna in modo non sicuro.

Mostreremo adesso alcuni esempi di contratti vulnerabili a questo tipo di attacco.

```
// UNSECURE
function withdraw() external {
    uint256 amount = balances[msg.sender];
    (bool success,) = msg.sender.call{value: balances[msg.sender
       ]}("");
    require(success);
```

```
    balances[msg.sender] = 0;
}
```

In questo caso, il balance dell'utente viene aggiornato solo dopo che la chiamata esterna è stata completata. Questo permette all'attaccante di chiamare la funzione withdraw più volte prima che il balance venga settato a zero, permettendo all'attaccante di rubare fondi allo smart contract. Una versione più complessa dello stesso processo è il caso cross function, di cui mostriamo un esempio:

```
    // UNSECURE
function transfer(address to, uint amount) external {
  if (balances[msg.sender] >= amount) {
    balances[to] += amount;
    balances[msg.sender] -= amount;
  }
}

function withdraw() external {
  uint256 amount = balances[msg.sender];
  (bool success,) = msg.sender.call{value: balances[msg.sender]}(
     "");
  require(success);
  balances[msg.sender] = 0;
}
```

In questo esempio, l'attaccante può effettuare un attacco di tipo reentrancy avendo una avendo una funzione che chiama `transfer()` per trasferire fondi spesi prima che il bilancio sia settato a zero dalla funzione `withdraw()`.

Unchecked-Calls