KENNESAW STATE UNIVERSITY

COLLEGE OF COMPUTING AND SOFTWARE ENGINEERING

CS 4850-01 | FALL 2023 | SP-24 RED | SECURITY IN BLOCKCHAIN

TRANSACTION ORDER DEPENDENCE

Huy Le

INSTRUCTOR: SHARON PERRY

INTRODUCTION

Blockchain technology is a revolutionary innovation that allows for the creation and exchange of digital assets without the need for intermediaries or centralized authorities. Blockchain is based on a distributed ledger that records transactions in a secure and transparent way, using cryptography to ensure the integrity and immutability of the data. However, blockchain technology is not without its challenges and limitations. One of the potential vulnerabilities that can affect smart contracts, which are self-executing agreements that run on blockchain platforms, is the transaction-ordering dependence attack.

Transaction-ordering dependence, often referred to as TOD, is a vulnerability that can arise in smart contracts that are dependent on the sequence in which transactions are processed. This attack occurs when a smart contract's behavior depends on the order in which transactions are processed by the network, and an attacker can manipulate this order to gain an unfair advantage or cause harm to other parties.

This paper will analyze the technical aspects of transaction-ordering dependence (TOD) in smart contracts that rely on the sequence of transactions for their logic. By examining several real-life cases of TOD exploitation in smart contracts, this paper will demonstrate how attackers can manipulate the order of transactions using techniques such as front-running, back-running, and sandwich attacks. This paper will also propose some prevention strategies and recommendations for smart contract developers to mitigate the risk of TOD and ensure the security and fairness of their contracts.

TECHNICAL ASPECTS

Understanding Transaction-Ordering Dependence Attacks

A Transaction-Ordering Attack (TOD) is considered a race condition and front-running attack in blockchain due to the nature of how transactions are processed in a blockchain network. A race condition occurs when multiple transactions attempt to modify the same state variable at the same time. When this happens, it is unclear which transaction will be processed first, and the outcome of the contract becomes unpredictable. Moreover, when a transaction is initiated on a blockchain, it is broadcast across the network and finds its way into each node's mempool.

In a TOD attack, an attacker can observe a transaction that's beneficial to them and then quickly broadcast their own transaction with a higher gas price. This transaction also enters the mempool. Miners, who play a crucial role in maintaining the blockchain network, rely on the mempool to select and prioritize transactions for processing. The higher the fee associated with the transaction, the greater the incentive for miners to promptly prioritize and process that particular transaction. This means that if an attacker sends a transaction with a higher gas fee, it is likely to be processed before the original one. This allows the attacker to potentially profit at the expense of other users.

The Role of Mempool in TOD

The nodes that produce and verify blocks by adding a group of transactions are known as validators (or miners) in the Ethereum platform. All transactions sent to Ethereum contracts are initially stored in a memory pool called 'mempool' before a validator selects them to execute in the next block.

The order of transactions is not determined by the order in which they are sent to the mempool, but by the gas price that is attached to each transaction. The higher the gas price, the more likely the transaction is to be mined first. Therefore, an attacker can manipulate the order of transactions by sending a transaction with a higher gas price than the user's transaction. Moreover, an attacker can also be a miner, as the miner can choose the order in which the transactions are mined in a block.

This creates a problem in Smart Contracts that rely on the state of storage variables to remain at a certain value according to the order of transactions. A TOD attack can alter the state of these variables and cause unexpected outcomes for the users.

For example, if a user purchases an item at the price advertised, they expect to pay that price. However, a transaction-ordering attack can change the price during the processing of the user's transaction because someone else (the contract owner, miner, or another user) has sent a transaction modifying the price before the user's transaction is complete. This can result in the user paying more or less than they intended, or even failing to buy the item at all.

Data Races in Transactions

The order of transaction execution can affect the state of smart contracts if they share global variables that are accessed and modified by different transactions. TOD can cause data races that compromise the correctness and security of the contracts.

The blockchain platform uses a consensus mechanism to agree on the next block that will update the state of the blockchain. Ethereum has recently switched from a Proof of Work (POW) mechanism, where nodes compete by solving a hard puzzle, to a Proof of Stake (POS) mechanism, where a random node with a high stake is chosen to create and validate a new block.

Ethereum requires a fee called "gas" to run transactions and rewards nodes for prioritizing transactions with a higher gas price. Also, Ethereum limits the amount of gas that each block can use, allowing only a finite number of transactions in a block. This means that nodes can maximize their profit by selecting and ordering certain transactions in a block.

Ethereum's switch to POS introduces new risks such as more centralization of nodes. Nodes are not the only ones who can influence the gas price of transactions; attackers can also manipulate the gas price of their transactions, for example, by offering more gas than the victim contract's transaction and thus gaining some control over the transaction execution order.

Example

The contract below is designed to function as an intermediary for the provision of a digital asset at a specified price.

```
pragma solidity ^0.4.18;



contract TransactionOrdering {

uint256 price;

address owner;

event Purchase(address _buyer, uint256 _price);
event PriceChange(address _owner, uint256 _price);

modifier ownerOnly() {

require(msg.sender == owner);

_;

}



function TransactionOrdering() {

// constructor

owner = msg.sender;

price = 100;

}



function buy() returns (uint256) {

Purchase(msg.sender, price);

return price;

}



function setPrice(uint256 _price) ownerOnly() {

price = _price;

PriceChange(owner, price);

}

}
```

This price is stored in the contract's storage variable, designated as uint256 price. The owner of the contract has the ability to alter this price by invoking the setPrice(uint256 _price) function. This allows for dynamic pricing of the digital asset, providing flexibility and control to the contract owner.

This contract could potentially be vulnerable to a transaction-ordering dependence (TOD) attack. In Ethereum, miners choose the order of transactions in a block, and an attacker could potentially manipulate this to their advantage.

For instance, if an attacker sees a transaction from the contract owner to setPrice, they could front-run this transaction with their own buy transaction at the old price.

- The owner of the contract calls the setPrice function to lower the price of the digital asset.
- This transaction is broadcasted to the Ethereum network but is not yet included in a block.
- An attacker, who is constantly watching the transaction pool, sees this transaction before it's mined.
- The attacker quickly creates a buy transaction at the old price and specifies a higher gas price to incentivize miners to include their transaction in the next block before the owner's setPrice transaction.
- If successful, the attacker's buy transaction gets included in a block before the setPrice transaction, allowing them to purchase the digital asset at the old price.

This kind of attack is possible because Ethereum miners have discretion over the order of transactions within a block.

SOLUTION

Detecting and Preventing TOD Vulnerability in Smart Contracts

It is important to look for functions that change the state of the smart contract. Smart contracts maintain a state, which is altered by functions within the contract. If these functions are not properly implemented, they could be exploited by attackers.

Examine if the transactions order result has an impact on the output of the function. The order in which transactions are mined and included in the blockchain can affect the outcome of these functions. If the output of a function varies based on the order of transactions, it could be a sign of a TOD vulnerability.

Transaction-ordering dependence attack is a serious threat to the security and fairness of blockchain applications, especially those that rely on smart contracts. To prevent TOD attack, one of the common practices is to implement locking mechanism in Smart Contract to prevent race conditions.

Consider the code below:

```
pragma solidity ^0.4.18;


contract TransactionOrdering {

uint256 price;

address owner;

bool locked;

event Purchase(address _buyer, uint256 _price);

event PriceChange(address _owner, uint256 _price);

modifier ownerOnly() {

require(msg.sender == owner);

_;

}




modifier notLocked() {

require(!locked);

_;

}
function TransactionOrdering() {

// constructor

owner = msg.sender;

price = 100;

locked = false;

}



function buy() notLocked returns (uint256) {

Purchase(msg.sender, price);

return price;

}



function setPrice(uint256 _price) ownerOnly() {

locked = true;

price = _price;

PriceChange(owner, price);

locked = false;

}

}
```

In the Solidity code above, the locking mechanism works by temporarily blocking certain actions while a critical operation is being performed. When the setPrice function is called by the owner, the locked variable is set to true. This indicates that a price change operation is in progress. While locked is true, any attempts to call the buy function will fail because of the notLocked modifier. This modifier requires that locked be false for the function to execute. Once the price change operation in setPrice is complete, locked is set back to false. This signals that the critical operation has finished, and normal contract interactions can resume.

By doing this, it ensures that no one can call the buy function while a price change is in progress. This effectively prevents a TOD attack because an attacker can no longer time their transaction to execute in between the owner's price change transaction. The order of transactions no longer matters because the state of the contract (the price) cannot be changed while a purchase is being made.

This way, even if an attacker sees an unconfirmed transaction from the owner changing the price, they cannot exploit this information because their purchase transaction will simply fail if they try to execute it before the price change transaction is mined. They will have to wait until after the price change transaction has been mined and locked has been set back to false, at which point they will have to pay the new price.

CONCLUSION

In conclusion, Transaction-Ordering Dependence (TOD) represents a significant vulnerability in smart contracts that attackers can exploit to manipulate the order of transactions and gain an unfair advantage. This paper has examined the technical aspects of TOD, including the role of the mempool, data races in transactions, and the impact of consensus mechanisms like Proof of Work and Proof of Stake.

To mitigate the risk of TOD, developers are recommended to implement a locking mechanism in their smart contracts to prevent race conditions. As blockchain technology continues to evolve, it is crucial for developers to stay abreast of these vulnerabilities and develop robust strategies to ensure the security and fairness of the blockchain.

REFERENCES

Cakal, Yasin. “Transaction-Ordering Dependence (TOD).” *Code of Code*, 7 Jan. 2023, codeofcode.org/lessons/transaction-ordering-dependence-tod/.

ImmuneBytes. “Transaction-Ordering Dependency: How Can It Harm Smart Contracts?” *ImmuneBytes*, 2 Aug. 2023, www.immunebytes.com/blog/transaction-ordering-dependency-how-can-it-hamper-a-smart-contract/.

S. Munir and C. Reichenbach, "TODLER: A Transaction Ordering Dependency anaLyzER - for Ethereum Smart Contracts," 2023 IEEE/ACM 6th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), Melbourne, Australia, 2023, pp. 9-16, doi: 10.1109/WETSEB59161.2023.00007. https://ieeexplore.ieee.org/document/10190831

Luu, Loi, et al. “Making Smart Contracts Smarter: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.” *ACM Conferences*, 1 Oct. 2016, dl.acm.org/doi/10.1145/2976749.2978309. https://dl.acm.org/doi/10.1145/2976749.2978309

Oualid, Z. “Transaction Order Dependence Attack in Smart Contract.” *TRANSACTION ORDER DEPENDENCE ATTACK IN SMART CONTRACT*, 13 May 2022, www.getsecureworld.com/blog/transaction-order-dependence-attack-in-smart-contract/.