Ethereum Smart Contract Public Audit Report

For @machinomy/contracts

January 8, 2022

Previous audits

None

Common clauses

Audit subject

Smart contracts located in GitHub repository
https://github.com/machinomy/machinomy/tree/192c924c7cb228950e23fc5dd6958a903aff996f/packages/contracts/contracts

Disclaimer

Please be advised that this audit does not guarantee bug-free smart contract after audit remarks were fixed. It will be beneficial to order multiple audits from different audit firms.

Gas consumption

Not applicable

Remarks

1 High severity (critical)

1. SWC-102: Outdated Compiler Version; Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version. There many compiler/EVM bugs were found since 0.4.24 (https://docs.soliditylang.org/en/latest/bugs.html)
2. SWC-103: Floating Pragma; Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.
3. SWC-107: Reentrancy; One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.
   Please consider using Reentrancy guards
   https://docs.openzeppelin.com/contracts/4.x/api/security#ReentrancyGuard
   or modify your smart contract respectively -- make sure all internal state changes are performed before the call is executed.

```
Reentrancy in TokenUnidirectional.claim(bytes32,uint256,bytes) (contracts/TokenUnidirectional.sol#148-165):
        External calls:
        - require(bool,string)(token.transfer(channel.receiver,channel.value),Unable to transfer token to channel receiver)
(contracts/TokenUnidirectional.sol#155)
        - require(bool,string)(token.transfer(channel.receiver,payment),Unable to transfer token to channel receiver)
(contracts/TokenUnidirectional.sol#157)
        - require(bool,string)(token.transfer(channel.sender,change),Unable to transfer token to channel sender)
(contracts/TokenUnidirectional.sol#159)
        State variables written after the call(s):
        - delete channels[channelId] (contracts/TokenUnidirectional.sol#162)
Reentrancy in TokenUnidirectional.deposit(bytes32,uint256) (contracts/TokenUnidirectional.sol#71-80):
        External calls:
        - require(bool,string)(token.transferFrom(msg.sender,address(this),value),Unable to transfer token to the contract)
(contracts/TokenUnidirectional.sol#76)
        State variables written after the call(s):
        - channel.value = channel.value.add(value) (contracts/TokenUnidirectional.sol#77)
Reentrancy in TokenUnidirectional.open(bytes32,address,uint256,address,uint256) (contracts/TokenUnidirectional.sol#40-56):
        External calls:
        - require(bool,string)(token.transferFrom(msg.sender,address(this),value),Unable to transfer token to the contract)
(contracts/TokenUnidirectional.sol#44)
        State variables written after the call(s):
        - channels[channelId] = PaymentChannel(msg.sender,receiver,value,settlingPeriod,0,tokenContract)
(contracts/TokenUnidirectional.sol#46-53)
Reentrancy in TokenUnidirectional.settle(bytes32) (contracts/TokenUnidirectional.sol#116-126):
        External calls:
        - require(bool,string)(token.transfer(channel.sender,channel.value),Unable to transfer token to channel sender)
(contracts/TokenUnidirectional.sol#122)
        State variables written after the call(s):
        - delete channels[channelId] (contracts/TokenUnidirectional.sol#124)

Reentrancy in TokenUnidirectional.claim(bytes32,uint256,bytes) (contracts/TokenUnidirectional.sol#148-165):
        External calls:
        - require(bool,string)(token.transfer(channel.receiver,channel.value),Unable to transfer token to channel receiver)
(contracts/TokenUnidirectional.sol#155)
        - require(bool,string)(token.transfer(channel.receiver,payment),Unable to transfer token to channel receiver)
(contracts/TokenUnidirectional.sol#157)
        - require(bool,string)(token.transfer(channel.sender,change),Unable to transfer token to channel sender)
(contracts/TokenUnidirectional.sol#159)
        Event emitted after the call(s):
        - DidClaim(channelId) (contracts/TokenUnidirectional.sol#164)
```

```
Reentrancy in TokenUnidirectional.deposit(bytes32,uint256) (contracts/TokenUnidirectional.sol#71-80):
        External calls:
        - require(bool,string)(token.transferFrom(msg.sender,address(this),value),Unable to transfer token to the contract)
(contracts/TokenUnidirectional.sol#76)
        Event emitted after the call(s):
        - DidDeposit(channelId,value) (contracts/TokenUnidirectional.sol#79)
Reentrancy in TokenUnidirectional.open(bytes32,address,uint256,address,uint256) (contracts/TokenUnidirectional.sol#40-56):
        External calls:
        - require(bool,string)(token.transferFrom(msg.sender,address(this),value),Unable to transfer token to the contract)
(contracts/TokenUnidirectional.sol#44)
        Event emitted after the call(s):
        - DidOpen(channelId,msg.sender,receiver,value,tokenContract) (contracts/TokenUnidirectional.sol#55)
Reentrancy in TokenUnidirectional.settle(bytes32) (contracts/TokenUnidirectional.sol#116-126):
        External calls:
        - require(bool,string)(token.transfer(channel.sender,channel.value),Unable to transfer token to channel sender)
(contracts/TokenUnidirectional.sol#122)
        Event emitted after the call(s):
        - DidSettle(channelId) (contracts/TokenUnidirectional.sol#125)

Reentrancy in Unidirectional.claim(bytes32,uint256,bytes) (contracts/Unidirectional.sol#132-147):
        External calls:
        - channel.receiver.transfer(channel.value) (contracts/Unidirectional.sol#138)
        - channel.receiver.transfer(payment) (contracts/Unidirectional.sol#140)
        - channel.sender.transfer(channel.value.sub(payment)) (contracts/Unidirectional.sol#141)
        State variables written after the call(s):
        - delete channels[channelId] (contracts/Unidirectional.sol#144)
        Event emitted after the call(s):
        - DidClaim(channelId) (contracts/Unidirectional.sol#146)
Reentrancy in Unidirectional.settle(bytes32) (contracts/Unidirectional.sol#103-110):
        External calls:
        - channel.sender.transfer(channel.value) (contracts/Unidirectional.sol#106)
        State variables written after the call(s):
        - delete channels[channelId] (contracts/Unidirectional.sol#108)
        Event emitted after the call(s):
        - DidSettle(channelId) (contracts/Unidirectional.sol#109)
```

4. Consider don't use send or transfer. Use .call.value(...)("") instead. There was a gas price change for these methods.
   https://chainsecurity.com/istanbul-hardfork-eips-increasing-gas-costs-and-more/
5. SWC-113: DoS with Failed Call; External calls can fail accidentally or deliberately, which can cause a DoS condition in the contract. To minimize the damage caused by such failures, it is better to isolate each external call into its own transaction that can be initiated by the recipient of the call. This is especially relevant for payments, where it is better to let users withdraw funds rather than push funds to them automatically (this also reduces the chance of problems with the gas limit).
   Sending values to other addresses (send, transfer, call.value) should be handled carefully (for the failed cases as well).
6. SWC-121: Missing Protection against Signature Replay Attacks. It's strongly recommended to use nonce value when dealing with hashes and signatures like at Unidirectional.sol: 133, 122 and TokenUnidirectional.sol: 138, 149 to avoid replay attacks.
   It would be beneficial to store all proceeded hashes in the state variable as well.
7. SWC-136: Sensitive data should not store on-chain non-obfuscated. Consider making channels field at Unidirectional.sol: 20 and TokenUnidirectional.sol: 22 private and store there hashed with salt only sender, receiver, value and tokenContract fields.

2 Medium severity (important)

1. Consider change storage modifier from storage to memory for short living variables
Like at Unidirectional.sol:85, 105 and TokenUnidirectional.sol: 63, 74, 87, 98, 108, 119, 135, 151, 172, 186

2. Provide an error message for require() at Unidirectional.sol:36, 62, 83, 104, 133 and support/TestToken.sol: 8, 9.

3. Strongly recommended to add require() checks for cases when receiver arg is incorrect in open() function at Unidirectional.sol: 35 and TokenUnidirectional.sol: 40. For open() function at TokenUnidirectional.sol: 40 is also recommended to add check for ERC20 nature of provided tokenContract argument.

4. Several functions should be declared external
Unidirectional.open(bytes32,address,uint256) (contracts/Unidirectional.sol#35-47)
Unidirectional.deposit(bytes32) (contracts/Unidirectional.sol#61-67)
Unidirectional.startSettling(bytes32) (contracts/Unidirectional.sol#82-89)
Unidirectional.settle(bytes32) (contracts/Unidirectional.sol#103-110)
Unidirectional.claim(bytes32,uint256,bytes)(contracts/Unidirectional.sol#132-147)

5. Since ERC20 contracts are used, examine the opportunity of usage SafeERC20 module from OpenZeppelin Solidity for safe handling of ERC20 operations.

Low severity (informational)

1. Consider moving to Solidity 0.8.0+. It has integrated Integer Underflow/Overflow control, no need to use SafeMath in this case. Also it would be beneficial to upgrade to the latest OpenZeppelin Solidity (from v2 to v4).

2. SWC-116: Block values as a proxy for time; Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use oracles.
See TokenUnidirectional.sol: 97, 103, 107 and Unidirectional.sol: 84, 94.

The End of Public Audit Report