Smart Contract Source Code Audit MoC Oracles

Prepared for Money On Chain • December 2020

v210127

1. Table Of Contents

2. Executive Summary

In December 2020, Money On Chain engaged Coinspect to perform a source code review of the smart contracts that comprise the Staking, Delay and Oracle Manager parts of the OMoC oracles system. The goal of the project was to evaluate the security of the smart contracts.

The assessment was conducted on the following private git repositories:

- https://github.com/money-on-chain/OMOC-Decentralized-Oracle (branch `release_1.0.0`, commit hash `8b83162aecef36a2e92921fa42ea439462f6f3ce`);
- https://github.com/money-on-chain/OMoC-SC-Shared (branch `release_1.0.0`, commit hash `5b8fb73949ff815ecbf05db04474188a63928e2e`);
- https://github.com/money-on-chain/MoC-Gobernanza (branch `release_1.0.0`, commit hash `3093d3b913c0c7c3a31c466916c13497b90607c3`);
- https://github.com/money-on-chain/MoC-Vesting (branch `release_1.0.0`, commit hash `0b6ce98a9d184a683d3bb072d0c5e461209e8945`).

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
1	2	4

The high risk issue (MOO-006) is about a flaw in the Voting Machine that allows a user to vote multiple times, and gives an attacker the possibility to choose the result of the voting process. The medium risk issues are two race conditions (MOO-007), and an issue about the lack of a minimum number of participating oracles required for the published price to be considered valid (MOO-003).

Tests are not extensive enough and it is recommended to improve them. Some of the tests failed to run or produced errors, and attempting to run test coverage also produced errors.

All issues have been addressed following Coinspect recommendations, and as of **January 27th** all fixes are in pull requests waiting to be merged.

There are concerns regarding the safety of the system when the number of participating oracles is too small. This assessment focused on the security of Solidity code, and it is suggested to consider doing a deeper analysis of the system from economic perspectives.

The off-chain components were out of scope for this assessment, and it is recommended to audit them in the future.

3. Introduction

The audit started on December 7th and was conducted on the contracts from the following private git repositories:

- https://github.com/money-on-chain/OMOC-Decentralized-Oracle (branch `release_1.0.0`, commit hash `8b83162aecef36a2e92921fa42ea439462f6f3ce`);
- https://github.com/money-on-chain/OMoC-SC-Shared (branch `release_1.0.0`, commit hash `5b8fb73949ff815ecbf05db04474188a63928e2e`);
- https://github.com/money-on-chain/MoC-Gobernanza (branch `release_1.0.0`, commit hash `3093d3b913c0c7c3a31c466916c13497b90607c3`);
- https://github.com/money-on-chain/MoC-Vesting (branch `release_1.0.0`, commit hash `0b6ce98a9d184a683d3bb072d0c5e461209e8945`).

A snapshot of each repository was provided by Money On Chain in a zip file.

The scope of the audit was limited to the following Solidity source files, shown here with their sha256sum hash:

```
OMoC-Decentralized-Oracle:
d9263f2d6f5a6edbc4d2e05e2ed1fd0fe0fa40f3c888713884eeea7376c488dd  CalculatedPriceProvider.sol
b67b43e1612ac771f8af7c9fadaa1adf73e0cef47c39476e4ee95a81f9e8506f  CalculatedPriceProviderStorage.sol
625b5709a1393362bdaf00e01e9599cf8339b0ff6cd3a80762ae4f8724d823ef  CoinPairPriceFree.sol
c56969d00b56dc2e62ffa170de5a638a62c87ce16fa3d831edf34bac3049d8b7  CoinPairPrice.sol
ae0bc711ed596946c515dabf22789dfcba2e57dad992d06a1e0abab186c77265  CoinPairPriceStorage.sol
34f969a5c18958e796026d9a8f72f05098c1d075891ec987af40e8e5398ba94f  DelayMachine.sol
01e24292a58dd5568f705175031272d8b974966f00c6ae869613122bc25fe1e1  DelayMachineStorage.sol
798b299877eb9f99610d2c43dc14e6b04eebe9bf67f254acdd779d234ee01df4  InfoGetter.sol
9c0059d685a29ac616886b83738a99885ac9e1e6005078d9aca9b68ae23e3e30  Migrations.sol
474637a2b3182c0c4d028f8cce7f6df53a2679bdd8786099b2529212e32ad955  OracleManager.sol
001a16d9ebac62c74ce87519f8247d7e4c16c013df15b7d21954024aec900b20  OracleManagerStorage.sol
916ea9144336b5c00909f940b18dd5d2cb73b2d0b3dfc42d58e790221c3798e8  PriceProviderRegister.sol
fa28201677a7ad6ef5ab602bfe9b578fe14873eec9a00f5f534653e8da12cf08  PriceProviderRegisterStorage.sol
cadf06ba912a40fe7e8761ac30ffcc3b10b89a2201d560d22da803d3a561ad8e  Staking.sol
4c613d812c2dd52bc0b83151d1f7832d8eef093cca01657d850e3ae64bfbb7e1  StakingStorage.sol
c88929559f250c3d543c902c8a928c66ffc31fe047a80c59738ec701599195f6  Supporters.sol
e5f2bd1db98b57472d7883dc7f49df4afacc0903e794d1e608a263733682572a  SupportersStorage.sol
eb739d08aadb7c986dc6b7525d00c543a0a03bc186ddfcf24ba7397dd2ec43d4
CoinPairEmergencyPeriodInBlocksChange.sol
132d58dec1a36ab1df8f28d814bdd6ca81ed582d7c6b1c6f12dd7be0378d3867
CoinPairEmergencyWhitelistChange.sol
e7cb40428370667c3fb8b29a2d19401f97a01e27b09bbcef2d549987d4117a34
CoinPairEmergencyWhitelistListChange.sol
ba4fcfbab9fb07ea034fe3a404b83b9aeec8e2b9df1524277d82dbb4e8f52fe1
CoinPairPriceAddCalculatedPriceProviderChange.sol
02a48130d50ab3f211d9c1ca43811d328a3419f5b087688b7785df340a38bda0
CoinPairPriceMaxOraclesPerRoundChange.sol
28e0ea9f25c8c2cf3ac9a610894e8fc0f875321fcf27938c477b002d2b8d62cc
CoinPairPriceRoundLockPeriodChange.sol
1e824942866e16bee88607ea721a8685ef2ba1d6560770514a2843b703d705a5
CoinPairPriceValidPricePeriodInBlocksChange.sol
87b555dc315856be7bf81e41557829f42cf0a482f59e3c9d06090ed43eecedb0  GovernorChange.sol
```

```
e997409d09a6c5295d0a4b08058dac157bdfacf092d26c0a61e54789a4fd60f1
MocRegistryEnteringFallbacksAmountsChange.sol
3c669437562a046d845261642436ecbeca59e8d7b84b13994493e399a2af9c5d  MocRegistryInitChange.sol
3a1dc0a4e74cf0c87d0d5eb283617240b861492aa8296a5016de98e5dd6ac2de  MocRegistrySchedulerDelayChange.sol
effe4e18273a685e9d88f84bac0cdc3a5d22ff0738e3db71575398d4e153d7e1  OracleManagerPairChange.sol
685ea5ee9c075618945ee6a17059fbce1ec744fac3216e09679551a51ba12463  OracleManagerRemoveChange.sol
563f45511e7ad2bf2acbe2a8c1f6d668019f584768c0d4a3cd53904bcd623c07  OracleManagerUnsubscribeChange.sol
8ed6272615a46c640e5cb70092ca24120ee67c6af8c9995396c52afe5747f33a
PriceProviderOracleManagerRegisterPairChange.sol
53f67846d5093c8f5f1cba11af60b2a49a6c5bb6846381c6b0a83a4dd26f1d29  PriceProviderRegisterPairChange.sol
dce8ea6e02a0cdb4b9d539da19750bb3d14abf3b82023fc58404a69e46bb7cca  SupportersPeriodChange.sol
f919d34876c5bfb70e3c994af85f20ffe7075dadd78951a3cf11e6a96b9c6bf2  TestMOCMintChange.sol
7000229b59b4b903c57f92837aea64c2e55c1341bfd344c03ca89779220c1295  CalculatedPriceProviderLib.sol
9f458c26ee5c3069270a5de38c2f47056a7c8768c0dc2887962a90f8f07a87ee  CoinPairRegisterLib.sol
5b1511e12dacffb31cf5f5c3b908b98ff1a5868e73b74166be33b7b91b850f58  IterableOraclesLib.sol
f9a2a8c06b3c5bc567f45d7f9413d62948633081f22cf5b13d7cd929a132d106  IterableWhitelistLib.sol
3386261344d0b8d508e82705b159e8453982e09354a6ccda8cffe0db8da0bd1c  RoundInfoLib.sol
a594eee321c0b61fc7d2c11099f6e03c6f283eb25693a732b7a90aa598992401  SubscribedOraclesLib.sol
3f4c9811cc0d059a01b0d6d37635e79f89186104dc642357cbb22c28ecd8de24  SupportersLib.sol

MoC-Gobernanza:
ed6f123f6b055b07e9d45f82119db7eaf3c4511e76401a94b096525a2b47a913  Migrations.sol
ed28e5454a18ee44ed234fbff0da917842c7300e1e959f8fa73de4b4e8300020  VotingMachine.sol
a08b48d5d0e304d3fd6bf7b02dfc942fbc1fc1deba8c10b4b0b5abcc368f7ab9  VotingMachineStorage.sol
9ce16ed673c867477e6fab433dfd4d79b167c279035345c5fe8beac8c6ced21d  MocRegistryInitChange.sol
f86514340a83fbfcd11afc540776617cf2493138f67c0b233137bc26ed688dd2  SkipVotingProcessChange.sol
d8061263b3ba905d9f6f66527de630d9e844159b321929110bca382893b92a6d  PreVotingDataLib.sol
0fef0e21b062dc2838499aeb3b77055b29bbf5db123560125b984fed0de2d5e1  ProposalInfoLib.sol
9ea2f01cad8487ee64171f41397f152994c650024bd4d01346d7674d05c67f97  RegistryValues.sol
351f566b8e01d5e8081def3bf7c315c67ec9f49e8503a9e86d35b4563e6815de  VotingDataLib.sol

MoC-Vesting:
ed6f123f6b055b07e9d45f82119db7eaf3c4511e76401a94b096525a2b47a913  Migrations.sol
8860b47df6f88c1c1481165b4ad16f9879b82b4b727af5840ff8b3199ae48490  VestingFactory.sol
8e6cb6f3c654d6de2195e0efcb74201454843c0f146e9d757c0d1ca625c53680  VestingFactoryStorage.sol
46d2c516ed7693547b2e64e5869c28486bdae64297d05f91eab808cbdcb3f92c  Vesting.sol
e77ec68b383cf438b60e5d40cac80f81632babcd5ee362ebe1921ead0dcd9730  VestingStorage.sol
28836189c615220d151df165b842c19ec964f7db42872d29ed38ad49f2591315  MocRegistryInitChange.sol
6f8cfc35676e7e8e39ad0fda366d9dd012dcb6d60859ffbee03073bc7d129b0a  VestingHolderChange.sol
201d8b218dd9ee93371159cc7e54c723949c760264f5c701f08896ec033a81d2  WhitelistAddChange.sol
831ec272d38896f2784dc38320b6ad75e0e27337cfeb095bf58695811bbed434  WhitelistRemoveChange.sol

OMoC-SC-Shared:
fd28352b2685608a48776bb28efa595b435498de7962a3073ffb4fd4c0b15685  GovernedRegistry.sol
1a682b9529f48ebb0dacfeefb6aab6e76c5bd68dc0834283b4df133039a445e8  ICoinPairPrice.sol
0f65a23d96005d2c209c666afdac5539369d29d4a184c2fae3e3575b4c0f72ea  IDelayMachine.sol
939de7eef99e793c7991c9923dc0dc5931f075f99b2730c1a209de3f33f93b7b  IMintableERC20.sol
0b976eeab6631400442a2b140f8bf1852f7b3e7b69cb3e5b2cb3adf8a6c3b83c  IOracleInfoGetter.sol
f3308c92ba4d8f10ea2ecee8ff36afe2998abe71adcbafe1ef1bf5e2023290fc  IOracleManager.sol
5cdafaaab576531b9c4e4086bc37e035060d96070ea418959ee2fe5619ece46c  IPriceProviderRegisterEntry.sol
1018d947054b3ab915c602ad38b7421ed3f54cd33ce89a3eda699f18059844de  IPriceProvider.sol
aa4570f26a003bdc29374a078322161741d9c98f0e3071208760ba9c17dd7cf8  IRegistry.sol
96624a71aaf7e6445b56f477f5f22dc23d75c79694644b42050577717758b406  IStakingMachine.sol
e38f21c8bf1bbe4e6c341f4b8df055848980115c753659da1638846760a85fc4  ISupporters.sol
b5823f0abf6d4a767e3779b7468856730d0063f9c480bf7035b05ccea658c537  IterableWhitelistLib.sol
1dc01b656e72e8a7ae6adb0e5c883b16cfa9b7a3715f21050e9ca81317434b08  IVestingFactory.sol
59f754cab79d64d2049ee79c3e96504ca1be5d99dff20abd1dd50469f32003e1  IVestingMachine.sol
```

```
18c1f73fa1c2a51b4039f0fd563633a919bc1494239e39a49c255a1bd5e63f6f  IVotingMachine.sol
ed6f123f6b055b07e9d45f82119db7eaf3c4511e76401a94b096525a2b47a913  Migrations.sol
651f034e5b48c4cad817571ab76843858d2d73de91b3cc781baa7f1618e7046f  MocToken.sol
838fb890206d40d676a0a486df94eb067564eb78d31053e9839b7639cca2f212  RegistryConstants.sol
ddd3108ecd3ed08ee00f8cd03657667a196b373a8ff41c0caeb6e5a72dc3cbd9  MocRegistryInitChange.sol
0eb75e03afed4dd62619918f298a38fce29f79f6e048fc21c46d1fdb32f700cc  TestMOCMintChange.sol
3aeaf20fd70ab347ddc643edbfe7152779f4b485c2bb76a1c3afc89f75d72e1c  AddressSetLib.sol
d7dd7f8dab2d6302048b30979834c3ba710f41583e4270bbc0cf4272e380ac68  IOZAdminUpgradeabilityProxy.sol
c6fdb9b7269cfeb9d99ec87d1211db25113d6da6c9d85e294e276db1ab031af7  IOZProxyAdmin.sol
6e5da4fe3e221c119a350a5d194eaf7892655c2af9a4b74bd510292b95036830  UpgraderTemplate.sol
f2bff453e4e4241748c51a08e7385d8e8413d3c2a0ad98e68b5bd51e63ec8e1d  ChangeContract.sol
f5b975a6e0273f1669ab32019af49e4490c3094b110f324657dac7b7aa27a52b  Governed.sol
5adf69350d996cfbd2f6fed437cee18eb3720f8bc5a10386926cdfd9b14d8ee5  Governor.sol
0df330e5b756e1ec30222d9cb3f0bded775e27d717f2d94d7f653b6d9ee249ab  IGovernor.sol
d15bd4124c78b8e415bb99c7d836beddffdcab657e08d289089b570f71a754b1  Ownable.sol
5f217ebc9b5b9a2a4fb81fca236ba1cef1ba3693271a2cf120427c3992d96430  ReentrancyGuard.sol
a37b375e2dda1f9c8c1fe4f047fe1264e09c7987c7e3c93414074bc5d58a0cdd  UpgradeDelegator.sol
```

The reviewed code comprises mainly the following on-chain components:

- Staking Machine: allows users to operate with their stake and their oracles;
- Delay Machine: it holds a users' withdrawn stake for a time period until the user is allowed to definitely withdraw them;
- Vesting Machine: allows users to operate with their stake and withdraw according to certain rules;
- Voting Machine: allows users to vote for changes in the organization and interacts with the Staking machine to lock stake for the voting period.

All contracts are fully upgradeable. There are also:

- Changer contracts: are used for making authorized changes to the state of other contracts;
- Supporting libraries.

The off-chain components are the oracle servers run by users of the system. They achieve consensus by communicating over the internet. Each oracle server has a public IP address and they connect to each other in order to achieve consensus and obtain signatures for price updates that are then sent to the on-chain components. Relying on a public IP address for this communication can make them a target for DoS attacks.

The oracle servers are out of scope for this audit, but they security is key for the proper functioning of the system and it is recommended to audit them in the future.

4. Assessment

4.1. Stacking Machine

The Staking Machine includes the following main contracts:

- `Staking`: facade for all the operations regarding stake and oracles; the corresponding operations are delegated to `OracleManager` and `Supporters`;
- `OracleManager`: manages oracles operations and states such as registration and subscription to coin pairs;
- `Supporters`: owns the MoCs and pays supporters rewards;
- `CoinPairPrice`: organizes rounds and receives published prices.

The contracts are specified to be compiled with Solidity version 0.6.12. This is the latest version of the 0.6.x series.

The contracts compile without warnings, except four minor warnings about some OpenZeppelin contracts that are missing a SPDX license identifier. Linting with solhint produces 24 warnings, one about a variable without explicit visibility marker, and 23 about `require` messages that are too long.

The repository includes tests for the contracts, and 18 of them don't pass. It is recommended to review coverage. The tests are not extensive enough, for example most of the events emitted by the contracts are not tested, and many of the change contracts are also not included in the tests (see MOO-001).

It was found that the entries for `repository`, `bugs` and `homepage` in `package.json` contain references to the *MoC-Gobernanza* repository instead of *OMOC-Decentralized-Oracle*. Also, `package.json` includes dependencies that are downloaded from a Coinfabrik website over plain HTTP.

In general, the reviewed Solidity code is well written and clear. Some comments, however, are incorrect (for example the comment in the function `initialize` in `DelayMachine` and function `roundLockPeriodSecs` in `CoinPairPrice`), probably because copy & paste, or because code was modified and comments were not modified accordingly.

The contracts contain some unbound loops, i.e. loops over lists that could grow arbitrarily large. This means that the gas consumed by those loops can potentially be so large that using the contracts could become uneconomical. However, if the number of oracles and coin pairs are low this is not a problem.

A race condition was found in the function `publishPrice` in `CoinPairPrice`: 1) the function reverts if any of the provided signatures doesn't belong to one of the oracles participating in the current round, and 2) oracles could be removed at any time from the current round, for example due to withdrawal of staked funds (see MOO-004).

The private function `_recoverSigner` in `CoinPairPrice` is implemented in such a way that it returns `address(0)` to indicate failure. This means that every time it is called, the caller needs to make sure to check the return address for zero. The function is called only once, but still it is recommended to change it so that it never returns `address(0)` and instead it reverts if it fails to recover the signer. This is in general a safer way to do it because it prevents potential problems in the future (see MOO-002).

In the function `_distributeRewards` in `CoinPairPrice` the value computed in the variable `distSum` is never used, and it is recommended to remove it because it only wastes gas.

The function `deposit` in `DelayMachine` was found to contain an integer overflow when adding the current block timestamp with the expiration parameter (see MOO-05). A similar problem is present in function `switchRound` in `RoundInfoLib`, and it is not exploitable because the data is not user-controllable, but it is better to fix it.

4.2. Voting Machine

The contracts are specified to be compiled with Solidity version 0.6.12. This is the latest version of the 0.6.x series.

The contracts compile without warnings, except six minor warnings about some OpenZeppelin contracts that are missing a SPDX license identifier. Linting with solhint produces 12 warnings. Although none of them is important, it is recommended to fix them.

The repository includes 60 tests for the contracts, and 3 of them fail with errors. The tests don't seem extensive enough and it is recommended to review coverage and add more tests as needed (see MOO-001).

The event `VoteEvent` in `VotingMachine` is never emitted, and also the comment is wrong. This is probably a remnant from an older version.

Several functions contain integer overflows when adding timestamps. For example, in the function `preVote` in `VotingMachine`:

```
_lockStake(registry, msg.sender, block.timestamp +
_getVotingTimeDelta(registry));
```

Similar problems can be seen in function `__newProposal` in `PreVotingDataLib` and function `_startVoting` in `VotingDataLib`. The data is not user-controllable and these issues are currently not exploitable, but it would be better to use SafeMath to avoid any potential problems in the future.

The voting mechanism has flaws that allow users to call the function `vote` multiple times and cast votes as if they had an arbitrarily large stake, allowing them to change the vote outcome (see MOO-006).

4.3. Vesting Machine

The Vesting Machine comprises mainly two smart contracts:

- `Vesting`: stores the conditions for withdrawing funds in the future;
- `VestingFactory`: stores the timestamp of the TGE (Token Generation Event) and is common to all MoC holders.

The contracts are specified to be compiled with Solidity version 0.6.12. This is the latest version of the 0.6.x series.

The contracts compile without warnings, except seven minor warnings about some OpenZeppelin contracts that are missing a SPDX license identifier. Linting with solhint produces two warnings about missing visibility markers in two variables in `VestingHolderChange`.

The repository includes 21 tests for the contracts, and all tests pass except one that is marked as "pending". The pending test is for `VestingHolderChange` and according to the output it was "decided not to implement it by governance".

The tests don't seem extensive enough and it is recommended to review coverage and add more tests as needed (see MOO-001).

Many functions in the `Vesting` contract start with a `require(msg.sender == holder)`, it is recommended to implement a `onlyHolder` modifier and remove duplicated code.

The function `verify` in `Vesting` is intended to be called by the holder to mark the vesting contract as "verified", signaling the holder agreement with the vesting parameters (percentages and time deltas). Once the vesting is marked as verified, the parameters cannot be changed. However, this function is vulnerable to a race condition (see MOO-007), because the holder would look at the parameters and send a transaction to call `verify`, but at some point before the holder transaction is mined the vesting factory might call the function `update` and change the parameters.

There's also an inconsistency between functions `withdraw` and `withdrawAll`: the former requires the `Vesting` contract to be verified by the holder, while the later doesn't. It is recommended to fix this making both require the contract to be verified before allowing withdrawals.

The function `execute` in the change contracts `WhitelistAddChange` and `WhitelistRemoveChange` contain a *TODO* comment saying they should be made usable only once as other change contracts. It is recommended to either modify the contracts to be usable only once, or remove the comment, since in the current state it looks like they are incomplete.

4.4. Shared Contracts

The repository of shared contracts contains common interfaces, the AddressSetLib library that implements indexed sets of addresses, the MoC ERC20 token, some contracts for upgradeability based on OpenZeppelin's, and the governance base contracts (`Governor`, `Governed`, `ChangeContract`).

Some of the contracts are specified to be compiled with Solidity 0.6.12, which is the latest version of the 0.6.x series. But some contracts are specified for Solidity 0.6.0 and 0.6.2, which are slightly older, and it is recommended to change them to use Solidity 0.6.12.

The contracts compile without warnings, except seven minor warnings about some OpenZeppelin contracts that are missing a SPDX license identifier. Linting with solhint produces three warnings about missing visibility markers in two variables in `MocRegistryInitChange` and one in a mock contract.

The repository includes 52 tests for the contracts, and all tests pass.

5. Summary of Findings

ID	Description	Risk	Fixed
MOO-001	Incomplete tests	Low	✓
MOO-002	Function _recoverSigner should not return address(0)	Low	✓
MOO-003	Minimum number of oracles not enforced	Medium	✓
MOO-004	Race condition in function publishPrice	Low	✓
MOO-005	Integer overflow in function deposit in DelayMachine	Low	✓
MOO-006	Voting multiple times with the same stake is allowed	High	✓
MOO-007	Race condition in Vesting	Medium	✓

6. Findings

MOO-001 Incomplete tests

Total Risk	Impact	Location
Low	Low	test/*
Fixed	**Likelihood**	
✓	-	

Description

The tests included in the repositories don't cover events.

Also, many of the change contracts (contracts/change/*.sol) are not tested.

Recommendation

Add new tests for events. For each emit statement in the contracts, there should be a test that verifies that the event is emitted as expected and with the correct parameters.

Also verify test coverage, and add more tests as necessary to achieve full coverage.

Resolution

More tests were added in commits `c25a8a6e4530478c28abf09eddb6ea73d74abcb5` and `939228ab5a2434e4780ee5dab54b5a7e6a913511`. Coverage is now almost complete, although coverage of some contracts has room for improvement.

MOO-002 Function _recoverSigner should not return address(0)

Total Risk	Impact	Location
Low	Low	CoinPairPrice.sol
Fixed	Likelihood	
✓	High	

Description

The private function `_recoverSignature` in contract `CoinPairPrice` returns `address(0)` if it fails to recover the signer address from the provided signature. This is not a good practice because it forces each caller to check the returned value for zero, and if any caller forgets to do this it automatically introduces a bug.

Recommendation

Although the function `_recoverSigner` is called only once, it is recommended to modify it so that it never returns `address(0)`. If it fails to recover the signer address from the signature, it should revert.

Resolution

Fixed in commit `a27b4e296964fd26a159920d54229d3e6a3515f0`.

MOO-003 Minimum number of oracles not enforced

Total Risk	Impact	Location
Medium	High	CoinPairPrice.sol
Fixed	Likelihood	
✓	Medium	

Description

The contract `CoinPairPrice` enforces a maximum number of oracles allowed to participate in each round. However, it doesn't enforce a minimum number of oracles. When the number of participating oracles is too low, there is a greater risk of publishing the wrong price (for example due to cheating oracles manipulating the price for their own benefit, or because an oracle gets hacked, etc).

Recommendation

It is advisable to set a minimum number of oracles required to participate in a round in order to consider the price valid.

Resolution

Fixed in commit `e6cb8d9fb76936c536c756ebb022510c068782f3`.

MOO-004 Race condition in function publishPrice

Total Risk	Impact	Location
Low	Medium	CoinPairPrice.sol
Fixed	**Likelihood**	
✓	Low	

Description

The function `publishPrice` in contract `CoinPairPrice` receives a price feed from the selected oracle together with signatures that the oracle retrieved off-chain from other oracles participating in the round. It requires the signature of more than half of the participating oracles:

```
require(
   _sigS.length > roundInfo.length() / 2,
   "Signature count must exceed 50% of active oracles"
);
```

Then it requires the signatures to be sorted in ascending order, and it requires the signatures to be valid, the addresses to be distinct from each other, and each signer address has to be selected for the current round:

```
address lastAddr = address(0);
for (uint256 i = 0; i < _sigS.length; i++) {
    address rec = _recoverSigner(_sigV[i], _sigR[i], _sigS[i], messageHash);
    require(rec != address(0), "Cannot recover signature");
    address ownerRec = oracleManager.getOracleOwner(rec);
    // require(subscribedOracles.contains(ownerRec), "Signing oracle not
subscribed");
    require(roundInfo.isSelected(ownerRec), "Address of signer not part of this round");
    require(lastAddr < rec, "Signatures are not unique or not ordered by
address");
    lastAddr = rec;
}
```

However, if one of the signers does a partial withdrawal of funds, it could end up removed from the current round, and the call to `publishPrice` would revert.

This allows a malicious oracle to withdraw funds and at the same time prevent the next price update, and encourages selfish behaviour from the oracles breaking the consensus protocol.

Recommendation

Instead of first requiring the number of signatures to be greater than `roundInfo.length() / 2`, first run the loop and count how many signatures correspond to oracles selected in the current round (don't revert if a signer is not selected, just ignore it). Then, at the end, require for the count of valid signatures to be greater than `roundInfo.length() / 2`.

Resolution

Fixed in commit `b2915851567a22ba476c46937e8316f53207d6a1`.

MOO-005 Integer overflow in function deposit in DelayMachine

Total Risk	Impact	Location
Low	Low	DelayMachine.sol
Fixed	Likelihood	
✓	Medium	

Description

The function deposit in contract `DelayMachine` contains an integer overflow:

```
payments[_id].expiration = block.timestamp + expiration;
```

This is harmless within the context of the contracts, but due to the emitted `PaymentDeposit` event it could lead to problems, depending on how the event is handled off-chain.

Recommendation

Use `SafeMath`'s library `add` instead of `+`.

Resolution

Fixed in commit `4168b8b4c040018b776989dd14f9c52db2d085b9`.

MOO-006 Voting multiple times with the same stake is allowed

Total Risk	Impact	Location
High	High	VotingMachine.sol
Fixed	**Likelihood**	
✓	Medium	

Description

During voting, users call the `vote` function in `VotingMachine` to vote either in favor or against a proposal:

```
    function vote(bool _inFavorAgainst) external override atState(VotingState.Voting) {
        uint256 stake = _getStake(registry, msg.sender);
        Vote memory _vote = votes[msg.sender];
[...]
        votes[msg.sender] = Vote(votingData.winnerProposal, uint8(votingRound));
        votingData._addVotes(stake, _inFavorAgainst);
    }
```

In the function `_addVotes`, the votes are weighted by the user's stake:

```
   function _addVotes(
        VotingData storage self,
        uint256 _stake,
        bool _inFavorAgainst
    ) internal {
        if (_inFavorAgainst) {
            self.inFavorVotes = self.inFavorVotes.add(_stake);
        } else {
            self.againstVotes = self.againstVotes.add(_stake);
        }
    }
```

Note that users can call the `vote` function multiple times, and each time their stake is added again. This allows users to change the results, regardless of how much stake they have, only at the cost of the gas necessary to call the `vote` function enough times.

Recommendation

Each user's stake must be counted only once. New tests should be created to verify that voting more than once with the same stake is not allowed.

Resolution

Fixed in commit `a5c597c165daf3dcb2b3bab2f4414e2c2bc6e70e`.

MOO-007 Race condition in Vesting

Total Risk	Impact	Location
Medium	High	Vesting.sol
Fixed ✓	Likelihood Low	

Description

The function `verify` in `Vesting` is intended to be called by the holder to signal the holder's agreement with the vesting parameters (percentages and time deltas):

```
    function verify() public override {
        require(msg.sender == holder, "Only holder can verify contract");
        verified = true;
    }
```

Once the function is called the vesting is marked as *verified*, and the parameters can no longer be changed:

```
    function update(uint256[] memory _percentages, uint256[] memory _timedeltas) public
override {
        require(msg.sender == address(vestingFactory), "Cannot modify parameters");
        require(!isVerified(), "Cannot change after verification");
        _updateParameters(_percentages, _timedeltas);
    }

   function isVerified() public view override returns (bool) {
        return verified;
    }
```

However, this contract is vulnerable to a race condition, because the holder would look at the parameters and send a transaction to call `verify`, but at some point before the holder transaction is mined the vesting parameters could be changed again by calling the function `update`.

Recommendation

Either remove the function `update` that currently allows changing vesting parameters, or implement some mechanism by which the holder could specify the parameters being verified.

Resolution

Fixed in commit `3c4759eeae170d5f537474bcdf7c2dfaae1218da`.

7. Disclaimer

The present security audit does not cover the endpoint systems and wallets that communicate with the contracts, nor the general operational security of the company whose contracts have been audited. This document should not be read as investment advice or an offering of tokens.