Quantstamp Security Assessment Certificate

March 22nd 2022 — Quantstamp Verified

TreasureDAO

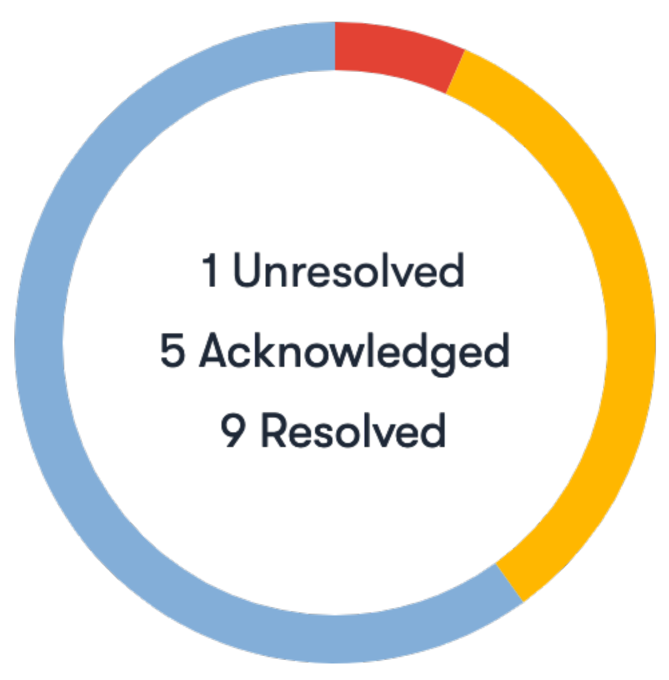
This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Ethereum
Auditors	Hisham Galal, Research Engineer Roman Rohleder, Research Engineer Fayçal Lalidji, Senior Security Engineer
Timeline	2022-03-14 through 2022-04-27
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	High
Test Quality	High
Source Code	

Repository	Commit
treasure-staking	b0d1576
treasure-marketplace	c731073

Total Issues	**15** (9 Resolved)
High Risk Issues	**2** (2 Resolved)
Medium Risk Issues	**4** (2 Resolved)
Low Risk Issues	**6** (3 Resolved)
Informational Risk Issues	**2** (1 Resolved)
Undetermined Risk Issues	**1** (1 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
? Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

We have reviewed TreasureDAO `Marketplace`, `AtlasMine`, `MasterOfCoin` contracts. Issues ranging across all severity levels have been found, therefore we strongly suggest reviewing all findings before using the code in production. The test suite and specifications were of high quality.

ID	Description	Severity	Status
QSP-1	`updateListing` can be front-run and exploited	High	Fixed
QSP-2	Incorrect vested amount computation in `AtlasMine.withdrawPosition(...)`	High	Fixed
QSP-3	`cancelListing` might throw	Medium	Fixed
QSP-4	`buyItem` might throw	Medium	Fixed
QSP-5	Precision loss due to unused base multipliers	Medium	Unresolved
QSP-6	Loop concerns with high gas	Medium	Acknowledged
QSP-7	`addStream` doesn't check the start timestamp	Low	Acknowledged
QSP-8	Privileged roles and ownership	Low	Mitigated
QSP-9	Unhandled token balance when funding and defunding a stream	Low	Acknowledged
QSP-10	`defundStream` doesn't check the actual stream balance	Low	Acknowledged
QSP-11	Magic reward boost doesn't follow the specifications	Low	Fixed
QSP-12	Uncapped fees	Low	Fixed
QSP-13	Events not emitted on state change	Informational	Acknowledged
QSP-14	Reentrancy in `createListing`	Informational	Fixed
QSP-15	Multiple 1:1 legions stakeable in `legionMetadataStore` error case	Undetermined	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.3

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 `updateListing` can be front-run and exploited

Severity: *High Risk*

Status: Fixed

File(s) affected: `TreasureMarketplace.sol`

Description: `TreasureMarketplace.updateListing(...)` can be front-run by a buyer to possibly acquire more NFTs than the user is selling, especially in the case of ERC1155. `TreasureMarketplace._cancelListing(...)` can also be front-run but has less impact on the end user.

Recommendation: Add a user input to validate that the quantity was not modified (withdrawn by a buyer that front run the seller).

QSP-2 Incorrect vested amount computation in `AtlasMine.withdrawPosition(...)`

Severity: *High Risk*

Status: Fixed

File(s) affected: `AtlasMine.sol`

Description: `AtlasMine.calcualteVestedPrincipal(...)` returns an incoherent vested amount value. If `block.timestamp >= vestingEnd || unlockAll` is true it returns the user full `originalDepositAmount` and if `block.timestamp > user.vestingLastUpdate` it returns the vested value since the last update of the user `vestingLastUpdate`. Consider an example where the user passes an amount that is less than the vested amount using AtlasMine.withdrawPosition(...). If the amount is lower than the ongoing vested amount the user won't be able to withdraw the totality of his vested fund until the end of the vesting period (even if they are already vested).

Recommendation: Reimplement the above-mentioned function to solve the incorrect vesting computation.

QSP-3 `cancelListing` might throw

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `TreasureMarketplace.sol`

Description: `TreasureMarketpalce.cancelListing` might not work especially for the case of ERC1155 due to the fact that a user can transfer its NFT outside of the `TreasureMarketplace`, this is applicable due to the following requirement:

```
require(nft.balanceOf(_msgSender(), _tokenId) >= listedItem.quantity, "not owning item");
```

The requirement checks if the listedItem.quantity is lower or equal to the owner balance which might not be true, allowing possible Marketplace users to front-run the user and buy his NFTs.

Recommendation: We recommend checking if the msg.sender is actually the user that listed the tokens and deleting the listed item without checking if the quantity is enough.

QSP-4 `buyItem` might throw

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `TreasureMarketplace.sol`

Description: `TreasureMarketplace.buyItem(...)` might throw even if the below requirement is true:

```
require(listedItem.quantity >= _quantity, "not enough quantity");
```

Since the owner can transfer his NFTs outside of the Marketplace contract.

Recommendation: We recommend checking both user balance and the set quantity in the `TreasureMarketplace` contract to return a correct error message.

QSP-5 Precision loss due to unused base multipliers

Severity: *Medium Risk*

Status: Unresolved

File(s) affected: `MasterOfCoin.sol`

Description: Computing the reward rate in `MasterOfCoin._fundStream(...)` or any other function should use a multiplier to raise the computation precision since a division is used to compute the new reward rate, this issue is also applicable to `MasterOfCoin.addStream(...)`, `defundStream(...)`, etc...

Recommendation: Multiply and divide the reward rate by a base multiplier wherever it is needed.

QSP-6 Loop concerns with high gas

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `AtlasMine.sol`

Description:

- Loop concerns regarding the execution of `AtlasMine.utilization(...)`, the function is looping through the excluded addresses list which can reach a high gas consumption or possibly block gas limit.
- `AtlasMine._recalculateLpAmount(...)` loops through all user deposit ids which can quickly raise the gas consumption possibly reaching block gas limit and denying service for NFT staking boost feature.
- Similarly `AtlasMine.withdrawAll(..)`, `harvestAll(...)` and `withdrawAndHarvestAll(..)` can throw if the user's deposit Ids are large enough.

Recommendation: We recommend carefully analyzing the possible gas consumption and its associated risks.

Update: Team response: AtlasMine.utilization(...) is a very small array and is not a concern. AtlasMine.withdrawAll(..), harvestAll(...) and withdrawAndHarvestAll(..) AtlasMine._recalculateLpAmount(...) are risks that we are aware of. For time being we accept these limitations as none of them are irreversible (closing positions by user makes them usable again) however we will do our best to design better mechanism in future versions.

QSP-7 `addStream` doesn't check the start timestamp

Severity: Low Risk

Status: Acknowledged

File(s) affected: `MasterOfCoin.sol`

Description: `MasterOfCoin.addStream(...)` does not check the start timestamp to be greater than the actual block timestamp, which can result in an incorrect stream reward distribution.

Recommendation: Add the necessary requirement.

Update: Team response: AtlasMine.utilization(...) is a very small array and is not a concern. AtlasMine.withdrawAll(..), harvestAll(...) and withdrawAndHarvestAll(..) AtlasMine._recalculateLpAmount(...) are risks that we are aware of. For time being we accept these limitations as none of them are irreversible (closing positions by user makes them usable again) however we will do our best to design better mechanism in future versions.

QSP-8 Privileged roles and ownership

Severity: Low Risk

Status: Mitigated

File(s) affected: `AtlasMine.sol`, `MasterOfCoin.sol`, `TreasureMarketplace.sol`

Description: Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.
The `AtlasMine.sol` contract contains the following privileged roles:

- `ATLAS_MINE_ADMIN_ROLE`, as initialized during `init()` to `msg.sender`:
  - Add/Remove addresses to `excludedAddresses`, which impacts the utilization calculation, by calling `addExcludedAddress()` and `removeExcludedAddress()`, respectively.
  - Set/Unset an arbitrary override value for the value returned by `utilization()`, by calling `setUtilizationOverride()`.
  - Change at any time the `magic` token address, which is set during `init()`, to an arbitrary one, by calling `setMagicToken()`.
  - Set `treasure` to an arbitrary address (including `address(0)`, in which case treasure staking/unstaking is disabled), by calling `setTreasure()`.
  - Set `legion` to an arbitrary address (including `address(0)`, in which case legion staking/unstaking is disabled), by calling `setLegion()`.
  - Set `legionMetadataStore` to an arbitrary address (used for legion 1:1 checking and legion nft boost computation), by calling `setLegionMetadataStore()`.
  - Re-set the `legionBoostMatrix` array to arbitrary values, by calling `setLegionBoostMatrix()`.
  - Set/Unset the emergency `unlockAll` state, by calling `toggleUnlockAll()`.
  - Withdraw all undistributed rewards to an arbitrary address, by calling `withdrawUndistributedRewards()`.

The `MasterOfCoin.sol` contract contains the following privileged roles:

- `MASTER_OF_COIN_ADMIN_ROLE`, as initialized during `init()` to `msg.sender`:
  - Add or remove streams, by calling `addStream()` and `removeStream()`, respectively.
  - Increasing an active stream's `ratePerSecond` and `totalRewards`, by calling `fundStream()`.
  - Decrease an active stream's `ratePerSecond` and `totalRewards`, by calling `defundStream()`.
  - Modify a stream's `startTimestamp`, `lastRewardTimestamp`, `endTimestamp` and indirectly `ratePerSecond`, by calling `updateStreamTime()`.
  - Enable/Disable registered stream addresses as callbacks, by calling `setCallback()`.
  - Withdraw an arbitrary `magic` token amount to an arbitrary address, by calling `withdrawMagic()`.
  - Set the `magic` token address to an arbitrary address, by calling `setMagicToken()`.

The `TreasureMarketPlace.sol` contract contains the following privileged roles:

- `TREASURE_MARKETPLACE_ADMIN_ROLE`, as initialized during `initialize()` to `msg.sender`:
  - Set an arbitrary fee up to `1500` (`BASIS_POINTS`), by calling `setFee()`.
  - Set an abitrary address as a fee recipient, by calling `setFeeRecipient()`.
  - Change the token approval status of arbitrary addresses (`NOT_APPROVED`/`ERC_721_APPROVED`/`ERC_1155_APPROVED` - given they support the corresponding interfaces), by calling `setTokenApprovalStatus()`.
  - Pause/Unpause the contract, thereby impacting the callability of `createListing()`, `updateListing()` and `buyItem()`, by calling `pause()`/`unpause()`.

Recommendation: Clarify the impact of these privileged actions on the end-users via publicly facing documentation.

QSP-9 Unhandled token balance when funding and defunding a stream

Severity: Low Risk

Status: Acknowledged

File(s) affected: `MasterOfCoin.sol`

Description: Both `MasterOfCoin.fundStream(...)` and `defundStream(...)` do not handle the reward token balance, meaning that the owner has the ability to possibly allow more or less reward than what the actual contract balance contains.

Recommendation: We recommend verifying the contract specification to avoid issues when distributing the rewards to the users.

Update: Team response: Design assumption was to decouple reward distribution from token balance of the contract so the contract can be refilled periodically and does not require to store extensive amount of token. Notice that if contract runs out of token, it stops rewards, however, the moment it is refilled, it resumes rewards as if nothing happened paying for the paused period as well.

QSP-10 `defundStream` doesn't check the actual stream balance

Severity: Low Risk

Status: Acknowledged

File(s) affected: `MasterOfCoin.sol`

Description: `MasterOfCoin.defundStream(...)` does not check if the actual stream balance is enough, meaning that the amount to be removed can be lower than the actual contract balance `amount <= (stream.totalRewards - stream.paid)`.

Recommendation: We recommend adding the necessary check and similarly validating `MasterOfCoin.fundStream(...)` to be valid (enough tokens in the contract balance).

Update: Team response: Note, as mentioned in QSP-9, decoupling stream rewards from contract's actual token balance is intentional.

QSP-11 Magic reward boost doesn't follow the specifications

Severity: Low Risk

Status: Fixed

File(s) affected: `AtlasMine.sol`

Description: `AtlasMine.getRealMagicReward(...)` does not follow the specification written in the project README. The percentages are not respected. We recommend clarifying this discrepancy.

Recommendation: Validate if it is an intended change.

QSP-12 Uncapped fees

Severity: Low Risk

Status: Fixed

File(s) affected: `TreasureMarketplace`

Description: Function `TreasureMarketplace.setFee()` does not constrain the input `_fee` other than the maximum of 100% (BASIS_POINTS = 10000). This can subject users high fees with no prior announcements.

Recommendation: Consider adding a MAX_FEE bound and state the max fee in public-facing documentation.

QSP-13 Events not emitted on state change

Severity: Informational

Status: Acknowledged

File(s) affected: `AtlasMine.sol`, `MasterOfCoin.sol`

Description: An event should always be emitted when a state change is performed in order to facilitate smart contract monitoring by other systems which want to integrate with the smart contract.
This is not the case for the functions and the correspondingly modified state variables:

1. `AtlasMine.addExcludedAddress()`, after changing `excludedAddresses`.
2. `AtlasMine.removeExcludedAddress()`, after changing `excludedAddresses`.
3. `AtlasMine.setUtilizationOverride()`, after changing `utilizationOverride`.
4. `AtlasMine.setMagicToken()`, after changing `magic`.
5. `AtlasMine.setTreasure()`, after changing `treasure`.
6. `AtlasMine.setLegion()`, after changing `legion`.
7. `AtlasMine.setLegionMetadataStore()`, after changing `legionMetadataStore`.
8. `AtlasMine.setLegionBoostMatrix()`, after changing `legionBoostMatrix`.
9. `AtlasMine.toggleUnlockAll()`, after changing `unlockAll`.
10. `AtlasMine.withdrawUndistributedRewards()`, after changing `totalUndistributedRewards`.

11. `MasterOfCoin.setMagicToken()`, after changing `magic`.

Recommendation: Emit an event in the aforementioned functions.

Update: The team acknowledges that events are a best practice and they should be added. However, events are not currently used and the team wants to minimize number and size of a code updates to smart contracts at this time. This will be improved in future version.

QSP-14 Reentrancy in `createListing`

Severity: Informational

Status: Fixed

File(s) affected: `TreasureMarketplace`

Description: `TreasureMarketplace.createListing()` performs calls to an external contract `_nftAddress`, before updating state variables `listings` and thereby violating the "Checks-Effects-Interactions" pattern. As only whitelisted addresses may be used for `_nftAddress` and the state changes done are only the creation of a listing, the likelihood as well as the impact for this are low and therefore this is purely informational.

Recommendation: Consider adding the `nonReentrant` modifier to function `createListing()`.

QSP-15 Multiple 1:1 legions stakeable in `legionMetadataStore` error case

Severity: Undetermined

Status: Fixed

File(s) affected: `AtlasMine.sol`

Description: Function `AtlasMine.isLegion1_1()` is used in `AtlasMine.stakeLegion()` to check if a given `_tokenId` corresponds to a 1:1 legion and if so check whether or not a 1:1 legion was already staked. In the scenario where `legionMetadataStore` returns an error within `isLegion1_1()`, the function would just return false (not revert) and would allow for staking/unstaking multiple 1:1 legions.

Recommendation: Clarify this scenario and/or consider replacing the `return false` in `isLegion1_1` in the metadata store error cases with reverts.

Automated Analyses

Slither

Slither did not return any significant result

Adherence to Best Practices

1. `TreasureMarketplace._buyItem` does not respect the CEIP. Even if there is possibly no associated risk right now due to the usage of `nonReentrant` modifier we recommend moving all state variable changes before any external call.
2. For better readability and gas consumption we recommend changing the implemented logic in `AtlasMine.getTreasureBoost(...)` to mapping to avoid any confusion.
3. Typo: `AtlasMine.calcualteVestedPrincipal`, `TreasureMarketplace.creatisgn`
4. To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the (other) address parameters in
    1. `MasterOfCoin.StreamGrant()`,
    2. `MasterOfCoin.Withdraw()`,
    3. `MasterOfCoin.CallbackSet()`,
    4. `AtlasMine.Staked()`,
    5. `AtlasMine.Unstaked()`,
    6. `TreasureMarketPlace.UpdateFeeRecipient()`,
    7. `TreasureMarketPlace.ItemListed()`,
    8. `TreasureMarketPlace.ItemUpdated()`,
    9. `TreasureMarketPlace.ItemSold()`
    10. `TreasureMarketPlace.TokenApprovalStatusUpdated()`.
5. For improved readability and code quality it is advised to remove duplicate or unused code. In this regard the check `require(listedItem.pricePerItem > 0, "TreasureMarketplace: listing price invalid");` in L298 of `TreasureMarketplace.sol` may be removed, as for listings and updates it is ensured that `pricePerItem` is always greater or equal to `MIN_PRICE` (1e9).

Test Results

Test Suite Results

```
TreasureMarketplace
    init
        ✓ initialize()
        ✓ setFee() (89ms)
        ✓ setFeeRecipient() (73ms)
        ✓ approve token (96ms)
        ✓ unapprove token (106ms)
```

```
       ✓ pause() & unpause() (68ms)
    ERC721
      with NFT minted
         ✓ createListing() (202ms)
        with listing
          1) updateListing()
           ✓ cancelListing() (63ms)
           ✓ buyItem() (265ms)
           ✓ buyItem() with quantity 0
          token approval revoked
             ✓ buyItem()
    ERC1155
      with NFT minted
         ✓ createListing() (136ms)
        expirationTime
           ✓ success (152ms)
           ✓ expired
        with listing
          2) updateListing()
           ✓ cancelListing() (107ms)
          buyItem()
             ✓ all (183ms)
             ✓ partial (159ms)


  17 passing (6s)
  2 failing

  1) TreasureMarketplace
       ERC721
         with NFT minted
           with listing
             updateListing():
     AssertionError: Expected transaction to be reverted with Pausable: paused, but other exception was thrown: Error: missing argument: passed to contract (count=5, expectedCount=6, code=MISSING_ARGUMENT,
version=contracts/5.6.0)


  2) TreasureMarketplace
       ERC1155
         with NFT minted
           with listing
             updateListing():
     AssertionError: Expected transaction to be reverted with not listed item, but other exception was thrown: Error: missing argument: passed to contract (count=5, expectedCount=6, code=MISSING_ARGUMENT,
version=contracts/5.6.0)

  AtlasMine
     ✓ init()
     ✓ getLockBoost()
     ✓ setMagicToken()
     ✓ setTreasure()
     ✓ setLegion()
     ✓ setLegionMetadataStore()
     ✓ setLegionBoostMatrix() (319ms)
     ✓ isLegion1_1() (147ms)
    with multiple deposits
       ✓ deposit()
       ✓ magicTotalDeposits()
       ✓ withdrawPosition() (1104ms)
       ✓ withdrawAll() (806ms)
       ✓ harvestPosition() (515ms)
       ✓ harvestAll() (4560ms)
       ✓ withdrawAndHarvestPosition() (794ms)
       ✓ withdrawAndHarvestAll() (4709ms)
       ✓ toggleUnlockAll() (807ms)
       ✓ withdrawUndistributedRewards() (402ms)
      - calcualteVestedPrincipal()
      utilization
         ✓ getExcludedAddresses() && utilization() && addExcludedAddress() && removeExcludedAddress() (1672ms)
         ✓ setUtilizationOverride() (720ms)
         ✓ getRealMagicReward() (536ms)
      NFT staking
         ✓ getNftBoost()
         ✓ harvest scenarios with staking (4352ms)
        stakeTreasure()
           ✓ Cannot stake Treasure (39ms)
           ✓ Amount is 0
           ✓ Max 20 treasures per wallet (765ms)
           ✓ stake boosts (744ms)
        stakeLegion()
           ✓ Cannot stake Legion
           ✓ NFT already staked (258ms)
           ✓ Max 3 legions per wallet (968ms)
           ✓ Max 1 1/1 legion per wallet (401ms)
           ✓ stake boosts (1061ms)
        limit of deposits
Deposits: 3, GasLimit: 227357
Deposits: 10, GasLimit: 241724
Deposits: 50, GasLimit: 624994
Deposits: 100, GasLimit: 1107511
Deposits: 200, GasLimit: 2064665
Deposits: 500, GasLimit: 4968011
Deposits: 1000, GasLimit: 9693977
Deposits: 1250, GasLimit: 12104647
Deposits: 1500, GasLimit: 14497133
Deposits: 1750, GasLimit: 16897584
Deposits: 2000, GasLimit: 19313720
Deposits: 2250, GasLimit: 21700950
Deposits: 2500, GasLimit: 24082428
Deposits: 2750, GasLimit: 26480028
Deposits: 3000, GasLimit: 28905377
           ✓ makes deposits and stakes NFT (834776ms)
        with NFTs staked
           ✓ Withdraw amount too big
           ✓ NFT is not staked
           ✓ unstake boosts (2179ms)

  MasterOfCoin
    use proxy
       ✓ init()
       ✓ magic()
       ✓ MASTER_OF_COIN_ADMIN_ROLE()
       ✓ hasRole()
       ✓ grantRole() (59ms)
       ✓ addStream() (114ms)
      with streams
         ✓ getStreams()
         ✓ addStream()
         ✓ getRatePerSecond()
         ✓ getGlobalRatePerSecond()
         ✓ getPendingRewards() (97ms)
         ✓ grantTokenToStream() (168ms)
         ✓ fundStream() (131ms)
         ✓ defundStream() (128ms)
         ✓ updateStreamTime() (365ms)
         ✓ removeStream() (161ms)
         ✓ withdrawMagic() (56ms)
    requestRewards()
       ✓ [0] requestRewards() (45ms)
       ✓ [1] requestRewards() (45ms)
       ✓ [2] requestRewards() (49ms)
       ✓ [3] requestRewards() (64ms)
       ✓ [4] requestRewards() (50ms)
       ✓ [5] requestRewards() (48ms)
       ✓ [6] requestRewards()


  60 passing (15m)
  1 pending
```

Code Coverage

Quantstamp usually recommends developers increase the branch coverage to 90% and above before a project goes live, in order to avoid hidden functional bugs that might not be easy to spot during the development phase. For branch code coverage, the current targeted files by the audit achieve a lower score that should be improved further.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
AtlasMine.sol	**82.93**	**72.63**	**93.18**	**79.1**	**… 656,658,660**
MasterOfCoin.sol	96.2	67.5	100	98.75	139

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
d0c6149e948fdc911d87ce4d2c887233dbbbec9c802f1b462fba41b3282e937 ./TreasureMarketplace.sol
369c21c7032f6db143b30bcb03d0290b0a6a23150e421fd6053f991ecfe2daef ./AtlasMine.sol
8bee631a7723d19701cf1b84bac7e8a3cfc2e78cfaedb4c858ac433f1b0346bf ./AtlasMineV1.sol
58ab93a66f75e0a7c763b4554358686546c61ce4b39c18fa68f3d803b788a076 ./MasterOfCoin.sol
ecf2037f519cd1ca2be9a8bc8530001f0d7401b97c0a15ede8495239615be27f ./MasterOfCoinV1.sol
```

Tests

```
59556024a2812ce29677ff3bb8123f8a841049cd2577befe0ccccf1a5279c922 ./TreasureMarketplace.test.ts
f855e06bc93091c63db11b24178f73e531fbb33560541288e3fcc981280eb2c5 ./AtlasMine.test.ts
d4b5f17f393c3af8ecfcbf52048c8a650ff3b145397a257545a5d009853abead ./MasterOfCoin.test.ts
```

Changelog

- 2022-03-20 - Initial report
- 2022-04-07 - Audit commit 940ffde for TreasureMarketPlace
- 2022-04-22 - Final report delivery
- 2022-04-27 - Updated report for QSP-1 fix commit c731073

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.