Quantstamp Security Assessment Certificate

May 1st 2022 — Quantstamp Verified

Unicly

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

Executive Summary

Type	NFT Marketplace
Auditors	Kacper Bąk, Senior Research Engineer Philippe Dumonet, Senior Research Engineer
Timeline	2022-04-12 through 2022-05-01
EVM	London
Languages	Solidity, TypeScript
Methods	Architecture Review, Unit Testing, Computer-Aided Verification, Manual Review
Specification	Litepaper
Documentation Quality	Medium
Test Quality	Medium

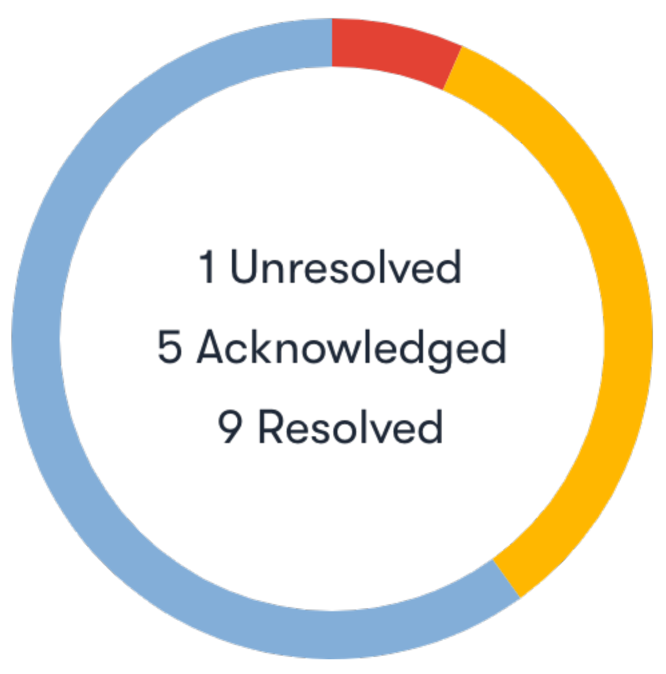
Source Code

Repository	Commit
NFT-Bazaar-Backend	5043c1c

Goals

- Can funds get locked up in the contract?
- Are auctions implemented correctly?

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

During the audit we have uncovered quite a few issues that span all severity levels. Notably, high, medium, and undetermined severity level issues are related to the business logic of the contracts. Furthermore, the test suite is limited and appears to cover only some functionalities of the code. We highly recommend addressing the issues, improving the test suite, and expanding the code documentation/specification.
It is important to note that only the following files were in the scope of the audit:

```
src/contracts/AuctionHandler.sol
src/contracts/Converter.sol
src/contracts/ConverterGovernorAlpha.sol
src/contracts/ConverterGovernorAlphaConfig.sol
src/contracts/UnicConverterGovernorAlphaFactory.sol
src/contracts/UnicConverterProxyTransactionFactory.sol
src/contracts/UnicFactory.sol
```

The team addressed most of the issues as of commit `8d43b00`.

ID	Description	Severity	Status
QSP-1	Arbitrary Call Execution From Pool	High	Fixed
QSP-2	Early Share Redemption Requires Giving Up On Remaining Pool Value	Medium	Acknowledged
QSP-3	Missing Airdrop	Medium	Fixed
QSP-4	Potential Airdrop Farming	Medium	Mitigated
QSP-5	Impossible Refunds	Medium	Fixed
QSP-6	Privileged Roles and Ownership	Low	Acknowledged
QSP-7	Arguments of Type `address` Are Not Checked to Be Non-zero	Low	Fixed
QSP-8	New Bids Can Be Placed Below Previous	Low	Fixed
QSP-9	Lack Of Events On Critical State Changes	Informational	Acknowledged
QSP-10	Clone And Own	Informational	Unresolved
QSP-11	Array Length Mismatch	Informational	Fixed
QSP-12	Trigger Price Does Not Account For Fee	? Undetermined	Acknowledged
QSP-13	Cannot Unset Airdrop Collection	? Undetermined	Fixed
QSP-14	Auction May Be Extended To Twice The Auction Extension	? Undetermined	Fixed
QSP-15	`AuctionHandler` Variables Limited To Small Ranges	? Undetermined	Acknowledged

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 v0.8.2

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 Arbitrary Call Execution From Pool

Severity: *High Risk*

Status: Fixed

File(s) affected: `UnicFactory.sol`, `Converter.sol`

Description: When creating `Converter` NFT pools via the `UnicFactory.createUToken()` method the new pool gets approved to spend 1M UNIC tokens (current max supply). This is done to allow the new pool to claim airdrops. Additionally, if the `createUToken()` method's `enableProxyTransactions` parameter is set to `true`, the factory will also configure the pool's `converterTimeLock` address. This `converterTimeLock` is configured based on the user-supplied `proxyTransactionFactory` address which is not validated in any way. The `converterTimeLock` address can call the pool's `forwardCall()` method which allows the pool to execute arbitrary calls on behalf of the pool.

Exploit Scenario:

1. The attacker creates and deploys a contract using the `createProxyTransaction(address, address)` method which returns a tuple `(address, address)` of their address.
2. The attacker creates a new `Convert` NFT pool by calling the factory and passing their contract's address as `proxyTransactionFactory` and setting `enableProxyTransactions` to `true`.
   - The custom contract is called returning the attacker's address and setting it as the `converterTimeLock` address.

- The factory approves the new pool to spend UNIC tokens (L88).

3. The attacker can then call the new pool's `forwardCall()` method, passing an encoded `transferFrom(factory, attacker, factoryFullBalance)` call with the target being the UNIC token contract.
    - Due to the pool being approved by the factory, it can successfully spend its tokens.

Beyond draining the factory's airdrop balance, this vulnerability also gives the attacker the ability to transfer out any deposited NFTs without having to initiate an auction. The data stored in the `nfts` mapping would, however, still persist, allowing the attacker to initiate auctions in the `AuctionHandler` for NFTs that the pool no longer has the access to. Combining this with the redemption of `Converter` pool uToken shares would allow the attacker to take the final bids resulting from fake auctions, leading to further damages.

Recommendation: We recommend validating the `proxyTransactionFactory` address. Alternatively, it could be stored in the factory as a variable only configurable by the protocol administrators. Additionally, we recommend to move the core airdrop validation logic to the factory so that the ERC20 approval of pools is unnecessary. This will have the following benefits: 1) reduction of the potential attack surface, 2) making the `Converter` contract more concise, and 3) improved gas usage as fewer individual calls to the factory contract are necessary.

QSP-2 Early Share Redemption Requires Giving Up On Remaining Pool Value

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `Converter.sol`, `AuctionHandler.sol`

Description: `Converter` NFT pools hold tokens configured by the pool creator (also referred to as "issuer" or "curator"). After depositing and configuring a set of NFTs the creator may execute a one-time issuance of pool shares. Shares are meant to depict partial ownership of the deposited assets within a pool. Assets held in a given pool may be auctioned off in return for ETH if someone is willing to bid the configured "trigger price". The pool creator can, however, set the trigger prices to any arbitrary value, any time they choose. This can allow pool owners to permanently block remaining NFTs in the pool by setting an extremely high price, which would prevent anyone from bidding on the assets. The pool owner has this ability regardless of how many pool shares they own.
Furthermore even if trigger prices aren't maliciously manipulated by the pool owner and auctions are allowed to be initiated at market or below market values, share redemption will still not allow shareholders to retrieve their proportional share of the pool's value. This is due to the fact that share redemption via the `AuctionHandler` contract's `burnAndRedeem()` method only redeems a proportional share of already collected auction proceeds, with no way for the redeeming shareholder to retrieve value from the remaining, unsold assets still in the pool. A shareholder who wants to redeem their shares for the proportional value of the entire pool would need to either wait for all the assets to be auctioned off, potentially requiring them to trigger the auctions themselves, which would come with obvious liquidity issues for certain users.
A shareholder could attempt to sell their shares on the secondary market but this may be difficult due to potentially limited liquidity for most pools. This is assumed due to the fact that there is no efficient way to add assets to existing pools after shares have begun circulating, pools will, therefore, likely be relatively small and fractured, providing little incentive to any potential liquidity providers.

Recommendation: We recommend to either redesign the NFT indexing method altogether, or, more simply, disable outright share redemption. Instead allowing shareholders to claim their share of auction proceeds without having to dispose of shares. This would present an improvement as shareholders would be able to extract auction proceeds in the short-term without having to give up the future ownership over the remaining assets.

QSP-3 Missing Airdrop

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Converter.sol`

Description: The function `deposit()` allows for adding 50 NFTs at a time and stores the total number of NFTs in `currentNFTIndex`. The function `issue()` locks the NFT collateral and also issues airdrops for eligible NFTs. In line 142, however, the loop exits after at most 50 iterations. Therefore, if any airdrop-eligible NFT is stored at an index higher than 49, its airdrop is ignored.

Recommendation: Allow for iterations over all airdrop-eligible tokens.

Update: The team addressed the issue but it's important to note that the successful loop execution depends on the amount of gas.

QSP-4 Potential Airdrop Farming

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `UnicFactory.sol`

Description: The `UnicFactory` contract allows the creation of multiple NFT pool `Converter` contracts. These pools allow the creator to issue uTokens via the `issue()` method, which act as shares of the pool. The factory owner address (`UnicFactory.owner()`) can configure a collection of contracts which will be eligible for an airdrop. Airdrops are issued by pools when the `issue()` method is called based on whether NFTs which were configured by the factory were deposited into the pool.
The current implementation of the airdrop mechanism may allow arbitragers / attackers (definition dependent) to repeatedly extract value from the system.

Exploit Scenario:

1. The arbitrager buys the cheapest NFT that belongs to a configured collection.
2. The arbitrager creates a `Converter` pool via the factory.
3. The arbitrager deposits the purchased NFT into their newly created pool.
4. The arbitrager issues pool shares by calling the pool's `issue()` method. Consequently, the airdrop reward is awarded to the arbitrager.
5. The arbitrager triggers an auction for the NFT in the pool.
6. Once the auction finalizes the arbitrager redeems their pool shares for the auction proceeds.
7. If the arbitrager bought the NFT from themselves, they can choose to resell.

As a result, the arbitrager's profit is: $\mathrm{v_a} * (1 - \mathrm{f_p}) * (1 - \mathrm{f_a}) + \mathrm{v_r} - \mathrm{c_t}$, where: $\mathrm{v_a}$ is the value of the NFT auction proceeds, $\mathrm{f_p}$ is the pool fee, $\mathrm{f_a}$ is the auction handler fee, $\mathrm{v_r}$ is the value of the airdrop reward, and `c_t` is the cost of the NFT. Note that the formula does not account for gas costs and the fact that the arbitrager would be exposed to the price of the deposited NFT and ETH for the duration of the auction.
While this strategy will not always be profitable (depending on the configured fees, airdrop reward value, volatility and NFT prices), it has been marked as an issue as it will incentivize artificial use of the protocol by arbitragers and may lead to a general loss to the protocol if the strategy does become profitable. The developers seem to have tried to prevent this by implementing a check whether or not the creator has already claimed an airdrop (`Converter.sol`: L139;L151), however this can very easily be circumvented by simply creating more accounts, also known as a *Sybil Attack*.

Recommendation: We recommend to adjust the airdrop mechanism such that it does not potentially incentivize artificial use of the protocol. Instead of the current design users could be airdropped based on their share of daily auction volume. Such a mechanism may be preferable due to it being harder to game and the provided value being more easily measurable due to the fact that the airdrop is now based on a continuous value (ETH transacted) rather than a discrete value (whether or not the pool contains a whitelisted NFT).

QSP-5 Impossible Refunds

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `AuctionHandler.sol`

Description: If `minBidMultiplier` is inadequate with respect to `feeDivisor`, it may be impossible for the old bidder to receive their refund since the `topBid.amount` is the full amount including the forwarded fee.

Recommendation: Make sure that whenever `minBidMultiplier` and `feeDivisor` are set, they make refunds possible.

QSP-6 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `ConverterTimeLock.sol`, `UnicFactory.sol`, `ConverterGovernorAlphaConfig.sol`, `Converter.sol`, `AuctionHandler.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. Specifically, we noticed the following:

- `admin` in `ConverterTimeLock.sol`,
- `feeToSetter`, `owner` in `UnicFactory.sol`,
- `owner` in `ConverterGovernorAlphaConfig.sol`,
- `owner` in `Converter.sol`, and
- `owner` in `AuctionHandler.sol`.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

QSP-7 Arguments of Type `address` Are Not Checked to Be Non-zero

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ConverterTimeLock.sol`, `UnicFactory.sol`, `UnicConverterProxyTransactionFactory.sol`, `Converter.sol`, `AuctionHandler.sol`

Description: Arguments of type `address` are not checked to be non-zero in the following functions;

- `ConverterTimeLock.constructor()`,
- `UnicFactory.initialize()`,
- `UnicConverterProxyTransactionFactory.constructor()`,
- `Converter.initialize()`,
- `Converter.setConverterTimeLock()`, and
- `AuctionHandler.initialize()`.

Recommendation: We recommend adding relevant checks.

QSP-8 New Bids Can Be Placed Below Previous

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AuctionHandler.sol`

Description: The `bid()` method is meant to allow new bidders to enter a bid in NFT auctions. However, new bidders may replace current bids by bidding less than the current bid. This becomes possible when the `AuctionHandler` contract's `minBidMultiplier` attribute is set to less than 100. The `minBidMultiplier` may be initialized to less than 100 as there is no check in the `initializer()` method.

Recommendation: Add a check enforcing that the `minBidMultiplier` value is above 100 in the contract's initializer method.

QSP-9 Lack Of Events On Critical State Changes

Severity: *Informational*

Status: Acknowledged

File(s) affected: `UnicFactory.sol`, `Converter.sol`

Description: The `UnicFactory` contract does not emit any events when its variables are updated via their setter methods (e.g. `setFeeTo()`, `setFeeToSetter()`, `setConverterImplementation()`, etc.). This presents an issue as events are necessary for the efficient and practical external tracking of state changes in smart contracts. A lack of events for critical state changes may also make the development of the protocol's UI more difficult.

The Converter contract also lacks events for important state changes in e.g. `setCurator()` or `setConverterTimeLock()`.

Recommendation: Add events which are emitted on all critical state changes.

QSP-10 Clone And Own

Severity: Informational

Status: Unresolved

File(s) affected: `UnicFactory.sol`

Description: The UnicFactory contract implements standard ERC1167 creation code in its `deployMinimal()` method. According to code comments (L136, L138) this code is copied from external repositories. This may be problematic as it increases development overhead as well as being prone to security-impacting errors accidentally introduced in the copying process.

Exploit Scenario: It is recommended to include external dependencies via imports and a package manager instead of manually copying existing code whenever possible. Both the normal and upgradeable OpenZeppelin contract repositories already offer ERC1167 proxy creation code.

QSP-11 Array Length Mismatch

Severity: Informational

Status: Fixed

File(s) affected: `Converter.sol`

Description: The `setTriggers()` and `deposit()` methods accept several arrays storing connected data. These methods however do not ensure that the given arrays match in length. This presents an issue as methods may fail with unexpected error messages or may unexpectedly ignore elements if arrays of different lengths are passed as parameters.

Recommendation: We recommend adding array length checks to the identified methods.

QSP-12 Trigger Price Does Not Account For Fee

Severity: Undetermined

Status: Acknowledged

File(s) affected: `AuctionHandler.sol`

Description: The `AuctionHandler` contract's `newAuction()` accepts an ETH payment and compares that with the stored trigger price. The actual bid however is the value sent (`msg.value`) minus the protocol fee. The actual starting bid may therefore be lower than the trigger price, triggering the auction regardless.

Recommendation: It is recommended the expected behavior of the logic be specified. If the fee should be accounted for in the `newAuction()` method the payment should be compared with the trigger price after the fee has been deducted.

QSP-13 Cannot Unset Airdrop Collection

Severity: Undetermined

Status: Fixed

File(s) affected: `UnicFactory.sol`

Description: Once configured as an NFT collection which is eligible to receive an airdrop via the `setAirdropCollections()` method, there is no way for a collection to be unset. This may pose an issue if the protocol administrators ever want to stop the distribution of airdrop rewards to a certain collection.

Recommendation: Specify whether or not the protocol owners are meant to be able to remove a collection and if so add the ability to do so.

QSP-14 Auction May Be Extended To Twice The Auction Extension

Severity: Undetermined

Status: Fixed

File(s) affected: `AuctionHandler.sol`

Description: The `bid()` method of the `AuctionHandler` extends the end of the auction if a bid is submitted within a certain time margin `auctionExtension` at the end of the auction. Depending on the value that is set for `auctionExtension`, this can serve as an effective way to limit auction sniping as any last minute bids will extend the auction, giving other bidders a chance to counter. However, when the auction is extended, the new end of the auction is set to the previous end + the time margin. This may result in the remaining time in the auction being almost double the configured extension. This may not be intended.

Recommendation: Clarify in the documentation how auction extension is intended to function. If it is intended for the `auctionExtension` parameter to act as the minimum remaining auction time the auction end should be set to the current timestamp + the auction extension. This will ensure that the auction settles only after the final bid has remained uncontested for `auctionExtension` seconds.

QSP-15 `AuctionHandler` Variables Limited To Small Ranges

Severity: Undetermined

Status: Acknowledged

File(s) affected: `AuctionHandler.sol`

Description: When being set through the `setAuctionParameters()` method, the `minBidMultiplier` and `feeDivisor` parameters are limited to the raw value 255 due to the method parameters for those attributes having the datatype uint8. For the `minBidMultiplier` attribute this translates to the contract being limited to requiring an increase of 1.01x - 2.55x the previous bid. For the `feeDevisior` this limits the contracts fee to values $\frac{1}{x}; x \in [1, 255]$. The available fee values are quite limited on the higher end >3% and begin increasing in resolution as the fee gets lower. The contract's fee is also limited to be at least ~0.4% with there being no option to set the fee to 0.

Recommendation: We recommend to clarify either in the documentation or a specification whether these limited value ranges are desired and sufficient to meet the necessary requirements.

Automated Analyses

Slither

- ETH transfer to arbitrary user in `ConverterGovernorAlpha.sol#230`. We classified it as a false positive.
- Reentrancy in `AuctionHandler.sol#180`. We classified it as a false positive.
- Return values ignored in
  - `Converter.sol#162`
  - `ConverterGovernorAlpha.sol` in lines 219 and 230,
- Reentrancy in `Converter.sol` in lines 182, 183, and 186.

Adherence to Best Practices

1. Magic constant in `Converter.sol#100`.
2. The `UnicConverterGovernorAlphaFactory` contract deploys `ConverterGovernorAlpha` instances in their entirety. This results in high upfront gas usage as every deployment will have to deploy the entirety of the contract's code. Depending on the requirements it may be advisable to leverage the ERC1167 standard in the factory contract, to decrease the large upfront cost in return for a small increase in the general cost of running the contract.
3. The bid method of the `AuctionHandler` contract blocks bids if the bidder has not withdrawn the refund from being overbid on the auction in the past. Furthermore refunds are stored separately for every auction. This not only degrades UX but is also inefficient as it will result in higher gas cost for frequent bidders who will require multiple individual calls to claim refunds across auctions. It is recommended that refunds from bids from different auctions be stored in a single value per account and that the bid method does not block new bids even if there are accessible refunds.
4. In `Converter.sol`, the "issuer" is sometimes referred to as a/the "curator" in method and variable names throughout the contract. It is recommended to retain a regular naming convention for system participants and components.
5. Both the `Converter` pools and `AuctionHandler` charge fees on behalf of the protocol. This is not necessary as a similar fee rate could be depicted by simply removing one fee and adjusting the second. It is recommended the bid fee in the `AuctionHandler` be removed to improve the contract's overall gas usage. The protocol would still earn a share of all auction proceeds through the shares it receives upon initial issuance (`Converter.sol`, L129-134) but bidders would no longer have to pay gas to compute and store the protocol's fee for every new bid.
6. In `Converter.sol`: L79-81, L104-106, L112-114: Inefficient end of array check. Instead of repeatedly checking whether the end of the array has been reached the for-loop's max iteration count should simply be the array's length. This would remove the necessity for the repeated if-then-break check within the loop.
7. In ` Converter.sol: Use of non-256-bit datatypes. The Converter contract often uses the 8-bit unsigned integer uint8 type for the storage of loop counters. It is recommended that smaller datatypes not be used, unless storage packing or consistent slicing is being leveraged. This is due to the fact the use of smaller number types leads to higher gas usage (outside of the noted exceptions) due to the compiler adding code to consistently slice and/or check the values to ensure that remain in their valid value ranges.
8. In ` UnicConverterGovernorAlphaFactory.sol: L12-15, L17: Inconsistent parameter ordering. The two methods UnicConverterGovernorAlphaFactory.createGovernorAlpha and ConverterGovernorAlpha.constructor accept the same set of parameters. However the order in which the parameters are passed and interpreted in is different. This is not ideal as it can be quite error prone to rearrange parameters between similar methods.
9. In ` UnicConverterProxyTransactionFactory: L10, L12: Unchanged variables not defined as immutable. The config and governorAlphaFactory values are only ever set in the constructor and could thus be defined as immutable. This is recommended as it can save gas and make the code clearer.

Test Results

Test Suite Results

All tests passed successfully. We note, however, that the test suite is limited and does not cover all of the functionalities.

```
AuctionHandler
  ✓ start new auction (634ms)
  ✓ bid and unbid (1309ms)
  ✓ claim (902ms)
  ✓ burn and redeem (584ms)

Converter
  ✓ state variables (218ms)
  ✓ issue (484ms)
  ✓ issue after fee is on (368ms)
  ✓ deposit (1455ms)
  ✓ refund (1411ms)
  ✓ deposit after issue (742ms)
  ✓ set vault creator (387ms)
  ✓ set triggers (542ms)

ConverterGovernorAlpha
  ✓ Voting right should be adjusted on token transfer (366ms)
  ✓ should be possible to execute a proposal (1688ms)
  ✓ should not be possible to execute a proposal multiple times (1761ms)
  ✓ should be possible to execute a complex proposal (2388ms)
  ✓ should not be possible to queue a proposal that has been rejected (1659ms)
  ✓ should not be possible to execute a proposal without queueing it (1474ms)
  ✓ should not be possible to execute a proposal before the time lock has expired (1569ms)
  ✓ should not be possible to execute an expired proposal (1617ms)
  ✓ should not be possible to vote multiple times (524ms)
  ✓ should not be possible to vote multiple times by transferring tokens (2134ms)
  ✓ should be possible to delegate votes (1483ms)
  ✓ should not be possible to vote with the help of already used delegate votes (1844ms)
  ✓ should be possible for the guardian to cancel a proposal (548ms)
  ✓ should be possible to cancel a proposal if the proposal threshold has changed and the proposer is below it (707ms)
  ✓ should not be possible to execute a canceled proposal (1728ms)
  ✓ should not be possible to make a proposal without exceeding the proposal threshold (338ms)
  ✓ should be possible to handle multiple proposals at once (4981ms)
  ✓ should not be possible to call the factory with a proxy transaction (2061ms)
  ✓ should not be possible to call the unic contract with a proxy transaction (1706ms)
```

```
31 passing (2m)
```

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

```
a72973b2378cb40703c97aef95545ab25161bb0babe3ec8501dbcb1ed476d6e6 ./contracts/UnicConverterGovernorAlphaFactory.sol
899de50ce32331a7e3a52c7696b1b61a36ceb1ba23c5bef546add4e4f8b509c5 ./contracts/UnicFactory.sol
491b7cf7ab63b678f21e300b5df9b0f946c27bf2d15cfa92577f6fbbe78c60b3 ./contracts/AuctionHandler.sol
f8394133d25bd5f004c50b7873304bf320939ddeb04c9700b63e130cff26f2af ./contracts/ConverterGovernorAlpha.sol
31a68b701ab7facfd3a6b3ffdd7fcab8b0b89e507174686804d1349c2ea3dc84 ./contracts/ConverterGovernorAlphaConfig.sol
2c2f08336bf0f0061936f6d734c9e0a085da102df5b358d989e2bb10def339ce ./contracts/Converter.sol
d915d7d2ecc0d7547c0c2df0ce8a97c82f999d0f1d050357b387298c8d5a0cb9 ./contracts/abstract/EmergencyWithdrawable.sol
```

Tests

```
f173fa2c73080a17c36b15b91c6ae6571b27456c240546f1b02f259094b8fa0d ./test/fixtures.ts
fd8da216e506a94b32783fb8ed65ac25b5f21d6d1a6172a84642358d76ff6f2a ./test/UnicFarm.spec.ts
8a7a0450be7d02e7d6a27045b03077f73cef0ae4c2daa4a65c3090020f597acb ./test/PointShop.spec.ts
7859e8750da8a2e314dc5cd9b6d2aa31cb1e8a74608ebf2a02d0da53b695a841 ./test/UnicFactory.spec.ts
6296c87e83ca72c82414bf8cb3d920858149cd635e03722447c3b4bf9e82d423 ./test/UnicGallery.spec.ts
68ed64c98e3cbf616841a679fb0e4ea0b0d46f102efa96b65229afa031695c80 ./test/UnicVester.spec.ts
d3a26c4a296c435ac51e370a3fa353fb93a7ff620302edf92c573bfe0c19a13c ./test/ConverterGovernorAlpha.spec.ts
d6378491a6b95c6d2ea05db62a1c143ff73726bcf2db504e6f7b77f144b7cec0 ./test/utils.ts
1a79feeaeb3714cf984a1f0ec4fe37e6f9cd99b5a1f97f2be34e83365e2098cb ./test/ProtocolUnicRewards.spec.ts
7389b20d22f0e2fddc14532bba420863f7b690c311d24211037db231904fa51b ./test/Unic.spec.ts
a1593286b721f26240e9e87edabaacb49ecd83e759ac59b4f2fddde39fe82a1d ./test/GovernorAlpha.spec.ts
f9fa3af15f806938c13e01e92b18ef534806d7f2fbf0e567683261b3bfff9ea3 ./test/UnicPumper.spec.ts
7f02f7c2de61aa5af9a3cf5b7b61cfa1080f2467e6d0899c226102f586e7ca55 ./test/AuctionHandler.spec.ts
ceef0e41a77f32e473c64111ca3f95bfb31d135ae58f6c4a122f4d5454253988 ./test/PointFarm.spec.ts
203c9c7176d9e5b1e8bb5493bd248c9ee438112a890563d6e96b5f8f0d01627d ./test/UnicStakingV6.spec.ts
acee79178b672b1cdec67d32ea2dc29398d85b9db79390e51f8e5c203eba515a ./test/Converter.spec.ts
```

Changelog

- 2022-04-18 - Initial report
- 2022-05-01 - Revised report based on commit `8d43b00`.

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.