Quantstamp Security Assessment Certificate

October 27th 2022 — Quantstamp Verified

VTVL

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

Executive Summary

Type	Token Vesting
Auditors	Philippe Dumonet, Senior Research Engineer Rabib Islam, Research Engineer Fatemeh Heidari, Security Auditor
Timeline	2022-05-05 through 2022-06-01
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	AccessProtected.md; VTVLVesting.md docs
Documentation Quality	High
Test Quality	High

Source Code

Repository	Commit
vtvl-app	8fe585f
vtvl-app	5345dbd

Total Issues	**5** (4 Resolved)
High Risk Issues	**1** (0 Resolved)
Medium Risk Issues	**1** (1 Resolved)
Low Risk Issues	**1** (1 Resolved)
Informational Risk Issues	**2** (2 Resolved)
Undetermined Risk Issues	0 (0 Resolved)

ALL ISSUES ADDRESSED

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

After reviewing the provided smart contracts we have identified several issues which should be addressed. Despite the discovered issues the quality of the code, documentation and tests speak to a good attention to detail from the developers.
Update: The status of the issues is valid as of the commit `5345dbdf7320669f3212c0d4250ff2a44f31dba9`

ID	Description	Severity	Status
QSP-1	Admins May Revoke Claims And Drain Contract At Any Time	High	Acknowledged
QSP-2	Ownership May Be Renounced	Medium	Fixed
QSP-3	ERC20 Transfer Return Value Not Checked	Low	Fixed
QSP-4	Unlocked Pragma	Informational	Fixed
QSP-5	Lack Of Input Validation	Informational	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 v0.8.2

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Admins May Revoke Claims And Drain Contract At Any Time

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `hardhat/contracts/VTVLVesting.sol`

Description: The `VTVLVesting` contract manages and stores vesting schedules for different accounts configured by the contract's owner address (`owner`). However the owner address has the ability to delete and block created vesting schedules at any time. Any tokens that have vested but were not withdrawn by the beneficiary are lost and returned to the owner's full control. This mechanism strongly diminishes the potential utility of the contract as "claims" within the contract are not immutable and can be revoked by the owner at any time. Beyond the minimal trustlessness that the contract provides any address with the admin or owner roles have the ability to revoke all claims and withdraw the tokens, draining the contract.

Recommendation: It is recommended that the ability to arbitrarily remove vesting schedules is removed from the contract. An alternative approach would be to require the beneficiaries approval in order to be able to revoke a claim. If the ability to arbitrarily remove claims is to be retained it should be clearly documented in a way that is accessible and clear to users.

Update: The client has indicated that they are aware of the potential dangers and will attempt to take the necessary precautions to protect the critical keys. Nevertheless the potential flaws resulting from the access control issue remain.

QSP-2 Ownership May Be Renounced

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `hardhat/contracts/VTVLVesting.sol`

Description: The `VTVLVesting` contract has an `owner` address which is responsible for creating claims, revoking claims and has the ability to withdraw any leftover tokens from the contract. However due the use of OpenZeppelin's `Ownable` contract a `renounceOwnership` method is provided to the contract by default which allows the owner to renounce their ownership over the contract. This may present an issue in the case where the owner either accidentally calls the method or calls the method while unused tokens are still left in the contract, trapping them forever.

Recommendation: If the ability to renounce ownership is required it is recommended the `renounceOwnership` method be extended to transfer any remaining excess tokens (`tokenAddress.balanceOf(address(this)) - numTokensReservedForVesting`) to the ownership upon revoking of the `owner` address. This will ensure that no tokens are accidentally left in the contract.

Update: Recommendation implemented.

QSP-3 ERC20 Transfer Return Value Not Checked

Severity: *Low Risk*

Status: Fixed

File(s) affected: `hardhat/contracts/VTVLVesting.sol`

Description: The `VTVLVesting` contract does not check the token's `transfer` call return value in its `withdraw` (L354) and `withdrawAdmin` (L373) methods. This presents an issue as not all ERC20 tokens revert on failed transfers, instead some simply return a `false` success flag. This is especially problematic for the `withdraw` method as the contract would still update the caller's vesting schedule even if the associated token transfer failed.

Recommendation: It is strongly recommended that the return value of ERC20 transfers calls be checked to ensure that failed transfers are properly caught by the contract. Such logic is already readily available in common smart contract libraries such as e.g. OpenZeppelin Contracts, which provides a `SafeERC20` library.

Update: Recommendation implemented, ERC20 transfers now leverage OpenZeppelin's `SafeERC20` library.

QSP-4 Unlocked Pragma

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/hardhat/*`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.0`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend removing the caret to lock the file onto a specific Solidity version.

Update: Recommendation implemented by fixing the solidity version to `0.8.14`. The `TestERC20Token` contract still does not have a fixed version, despite this, the issue has been marked as 'fixed' due to the only lacking file being named as "`Test{...}`", implying that it is not a production contract.

QSP-5 Lack Of Input Validation

Severity: *Informational*

Status: Fixed

File(s) affected: `hardhat/contracts/VTVLVesting.sol`

Description: The constructor of the `VTVLVesting` contract does not validate whether the initially provided `tokenAddress` is the zero-address or not. This may become an issue if the contract is accidentally deployed with the zero-address as the token; any tokens sent to such a contract would be stuck permanently as the `withdrawAdmin` method only allows for the withdrawal of the configured token.

Recommendation: It is recommended that the provided token address be validated in the constructor and that a method is added to the `VTVLVesting` contract which allows the owner to withdraw any token except for the configured `tokenAddress`. This will ensure that any potential mistakes made by the deployer during setup of the contract can be reversed if necessary.

Update: Recommendation implemented.

Automated Analyses

Slither

The results from Slither have been reviewed, with the false positives removed and any relevant findings added to the report along with the manually identified issues.

Adherence to Specification

1. `hardhat/contracts/VTVLVesting.sol`: While a comment on L32 states that the 40 bits available for the timestamps give the contract a potential range of more than a million years from now, 40 bits is limited to a range of approximately 35 thousand years (`2^40 / (365 * 24 * 60 * 60) ~= 35k`). While this is likely sufficient for the application this discrepancy is nevertheless noted for the sake of completeness.

Adherence to Best Practices

1. (fixed) `hardhat/contracts/VTVLVesting.sol`: L283: Calculation of `allocatedAmount` uses excessive gas. Leveraging the full logic of the `_baseVestedAmount` method is not necessary in the `createClaim` method. Instead the `allocatedAmount` could simply be calculated by adding the `_cliffAmount` to the `_linearVestAmount` value.
2. (fixed) `hardhat/contracts/VTVLVesting.sol`: Repeated access control validation. The `createClaimsBatch` method repeatedly calls the `createClaim` method, both of which check the caller's access via the `onlyAdmin` modifier, this results in the modifier's logic being run $n + 1$ times. Instead it is recommended that the logic of the `createClaim` method be moved to an internal method without access control which is then called from the `createClaimBatch` and `createClaim` methods. This would allow the `createClaimBatch` to run `onlyAdmin` once and then repeatedly call the internal claim creation logic.
3. (fixed)`hardhat/contracts/AccessProtected.sol`: Lack of indexed fields. The `AdminAccessSet` event does not index any of its fields. It is recommended that the `_admin` field of the `AdminAccessSet` event be marked as `indexed` to aid in off-chain querying of the events.
4. `hardhat/contracts/VTVLVesting.sol`: External methods defined as public. The visibility of the `getClaim`, `claimableAmount`, `allVestingRecipients` and `numVestingRecipients` methods could be defined as `external` only rather than `public`

Test Results

Test Suite Results

😉 fails on recipientAddress = 0 (183ms)
😉 fails on startTimestamp = 0 (156ms)
😉 fails on endTimestamp less than startTimestamp (140ms)
😉 fails on invalid interval length (132ms)
😉 fails on vesting duration not a multiple of release interval (134ms)
😉 fails on existing claim (145ms)
😉 fails on invalid cliff 1675209600-0 (152ms)
😉 fails on invalid cliff 0-10000000000000000000 (110ms)
😉 fails on nothing vested (108ms)
😉 fails on insufficient balance on initial allocation (154ms)
😉 fails on insufficient balance after multiple allocations (192ms)
😉 allocates correct amount (linear: 0, cliff: 1) (172ms)
😉 allocates correct amount (linear: 10000000000000000000, cliff: 0) (162ms)
😉 allocates correct amount (linear: 0, cliff: 10000000000000000000) (156ms)
😉 allocates correct amount (linear: 10000000000000000000, cliff: 10000000000000000000) (165ms)
😉 sets the claim (136ms)
😉 sets the vesting recipient (132ms)
😉 emits ClaimCreated (125ms)
😉 calculates the vested amount before the cliff time to be 0
😉 calculates the vested amount at the cliff time to be equal cliff amount
😉 correctly calculates the vested amount after the cliff time, but before the linear start time
😉 vests correctly if cliff and linear vesting begin are at the same time (124ms)
😉 correctly calculates the vested amount at the linear start time
😉 correctly calculates the vested amount after the start
😉 correctly calculates the vested amount at 10 of linear interval
😉 correctly calculates the vested amount at 25 of linear interval
😉 correctly calculates the vested amount at 45 of linear interval
😉 correctly calculates the vested amount at 50 of linear interval
😉 correctly calculates the vested amount at 70 of linear interval
😉 correctly calculates the vested amount at 80 of linear interval
😉 correctly calculates the vested amount at 95 of linear interval
😉 calculates the vested amount at the end of the linear interval to be the full amount allocated
😉 doesn't vest further after the end of the linear interval
😉 calculates the finalVestedAmount to be equal the total amount to be vested
😉 takes the release interval into account (140ms)
😉 delegates the call to createClaim properly (350ms)
😉 fails when called with invalid param length (191ms)
😉 allows withdrawal up to the allowance and fails after the allowance is spent (196ms)
😉 disallows withdrawal for an user without a claim (142ms)
😉 disallows withdrawal for an user with consumed claim (148ms)
😉 disallows withdrawal for an user with revoked claim (133ms)
😉 calculates the claimable amount to be equal to the vested amount if we have no withdrawals (302ms)
😉 takes withdrawals into account when calculating the claimable amount (162ms)
😉 allows admin to revoke a valid claim (134ms)
😉 prohibits a random user from revoking a valid claim (126ms)
😉 fails to revoke an invalid claim (276ms)
😉 fails to revoke an already withdrawn claim (209ms)
😉 takes revocation into account while calculating the vested amount (132ms)

Contract creation

😅 can be created with a ERC20 token address (58ms)
😅 fails if initialized without a valid ERC20 token address
😅 the deployer is the owner (65ms)
😅 not everyone is admin (76ms)
😅 allows the owner to set and unset other user as an admin (86ms)
😅 fails if attempting to set wrong address as an admin (70ms)

Admin withdrawal
😅 allows the admin to withdraw the allocated amount (159ms)
😅 reverts on attempt to withdraw more than available (110ms)
😅 doesn't allow random user to admin withdraw (106ms)

Code Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
hardhat/contracts/	100	100	100	100	
AccessProtected.sol	100	100	100	100	
TestERC20Token.sol	100	100	100	100	
VTVLVesting.sol	100	100	100	100	
All files	**100**	**100**	**100**	**100**	

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

de44f8144e85ab023b6d565ee66ba7a6a46d858d10330a4e56944e88bc85dad3 ./contracts/TestERC20Token.sol

eaeac720d7edfd6993d14b2c5ecd1664913e43d6ec581cf0a1006b5fbb99a542 ./contracts/AccessProtected.sol

3126c709fe568fbb893724004c073f424c49422f262a2b6d768c7a2c8b9631f2 ./contracts/VTVLVesting.sol

Tests

0cd684cd0c6d9a81c8b050670a0e25d4cc45e87275e83b45aeb5686f132e475e ./test/VTVLVesting.ts

Changelog

- 2022-05-06 - Initial report
- 2022-06-01 - Fixes update

About Quantstamp

Quantstamp is a global leader in blockchain security backed by Pantera, Softbank, and Commonwealth among other preeminent investors. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its white glove security and risk assessment services.

The team consists of web3 thought leaders hailing from top organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Many of the auditors hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 250 audits and secured over $200 billion in digital asset risk from hackers. In addition to providing an array of security services, Quantstamp facilitates the adoption of blockchain technology through strategic investments within the ecosystem and acting as a trusted advisor to help projects scale.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT