Quantstamp Security Assessment Certificate

February 24th 2022 — Quantstamp Verified

LiquiFi

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

Executive Summary

Type	Vesting
Auditors	Marius Guggenmos, Senior Research Engineer Poming Lee, Senior Research Engineer Roman Rohleder, Research Engineer
Timeline	2022-01-25 through 2022-01-26
EVM	London
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	gitbook
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
vesting-contracts (initial audit)	13498fb
vesting-contracts (reaudit)	356cd6d

Total Issues	**21** (20 Resolved)
High Risk Issues	**2** (2 Resolved)
Medium Risk Issues	**2** (1 Resolved)
Low Risk Issues	**8** (8 Resolved)
Informational Risk Issues	**7** (7 Resolved)
Undetermined Risk Issues	**2** (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).
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

After initial audit:
Quantstamp has performed an audit of the LiquiFi repository. Overall the code base is relatively small and makes use of a lot of OpenZeppelin libraries, that were not part of the audit. While there is some documentation present, it is fairly basic and still contains some `TODO` items. Additionally, there is no specification, which made it hard to tell whether a lack of some features is intended or not. The audit resulted in a total of 21 findings and an additional 11 best practice violations, described below. We confirm that none of the tests are failing when executed on our end. We recommend that all issues reported in this document be addressed.
After reaudit:
Quantstamp has checked the commit hash `356cd6d` and has determined that 20 issues have been resolved (that is either fixed or mitigated) and 1 issue has been acknowledged by the LiquiFi team. More details regarding each of the issues are provided in the update messages below each issue recommendation.

ID	Description	Severity	Status
QSP-1	Vesting Cannot Be Cancelled	High	Fixed
QSP-2	`LinearModule.createPlan` Missing Input Validation	High	Fixed
QSP-3	Use of Unsafe ERC20 Interface	Medium	Fixed
QSP-4	Privileged Roles and Ownership	Medium	Acknowledged
QSP-5	`createVestPlan` Missing Input Validation	Low	Mitigated
QSP-6	`EmployeeRegistry.isOwner()` Mishandling `address(0)`	Low	Fixed
QSP-7	`EmployeeRegistry.transferOwner` Might Lead to Unexpected State	Low	Fixed
QSP-8	Gas For Loop Concerns	Low	Fixed
QSP-9	`Escrow.deposit()` and `Escrow.withdraw()` Prone to Reentrancy	Low	Fixed
QSP-10	Functions Should Not be Payable	Low	Fixed
QSP-11	Mismatched Interface Signatures	Low	Fixed
QSP-12	`createPlan` Allows Overwriting Plans	Low	Fixed
QSP-13	Events Not Emitted on State Change	Informational	Mitigated
QSP-14	Missing Initializer Calls	Informational	Fixed
QSP-15	SafeMath Library Redundant	Informational	Fixed
QSP-16	Unnecessary `onlyOwner` Modifier	Informational	Fixed
QSP-17	Name Clashes with Ownable	Informational	Fixed
QSP-18	Redundant Storage Variables	Informational	Fixed
QSP-19	Missing Address Input Validation	Informational	Fixed
QSP-20	No Access Control on `EmployeeRegistry.addEmployee()` and `EmployeeRegistry.archiveEmployee()`	Undetermined	Fixed
QSP-21	Incorrect Accounting for `EmployeeRegistry.employeeToCompanyIds`	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 v0.8.2

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 Vesting Cannot Be Cancelled

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/vesting/Vesting.sol`

Description: The contract does not have any functionality to allow a company to stop vesting a former employee. There are two kinds of employees, `activeEmployees` and `inactiveEmployees`. The contract keeps track of them but never utilizes this information in regard to enabling/disabling of vesting.

Recommendation: Clarify whether this is intended. Also, based on the current design, if this mechanism is to be implemented, it is required to also add a function for a company owner to collect and retrieve the funds that were locked and were not given to the former employee.

Update: Code to archive vesting plans and retrieve the funds has been added in this PR.

QSP-2 `LinearModule.createPlan` Missing Input Validation

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/modules/LinearModule.sol`

Description: The function accepts a parameter called `_totalSupply` but does not utilize it properly. It neither checks whether `_cadenceSeconds * _cadenceAmount` is less than the `_totalSupply`, nor whether there are rounding issues that will lead to lost tokens. In case the company owner made a mistake with the inputs, the tokens might vest a lot faster than intended or require manual intervention of the contract owner to retrieve the locked tokens.

Recommendation: Add a require statement that enforces that `_cadenceSeconds * _cadenceAmount` is equal to `_totalSupply`.

Update: The `_totalSupply` parameter has been removed from the list of parameters, making `_duration`, `_cadenceSeconds` and `_cadenceAmount` the single source of truth for the vesting amount of the plan.

QSP-3 Use of Unsafe ERC20 Interface

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/escrow/Escrow.sol`

Description: Functions `Escrow.deposit()` and `Escrow.withdraw()` are using the *unsafe* ERC20 interface functions of OpenZeppelin (`transferFrom()` and `transfer()`). Since some ERC20 tokens do not return a value for these functions these calls might revert.

Recommendation: Replace the `IERC20` interface with the safe variant, also from OpenZeppelin, SafeERC20, to ensure safe behavior, regardless of whether the token call returns a value.

Update: The code now uses the safe versions for transfers. Added in this PR.

QSP-4 Privileged Roles and Ownership

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `contracts/modules/CliffModule.sol`, `contracts/modules/LinearModule.sol`, `contracts/modules/StreamModule.sol`, `contracts/employee/EmployeeRegistry.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 contracts `CliffModule.sol`, `LinearModule.sol` and `StreamModule.sol` contain the following privileged roles:

- `vestingControllerAddress`, as set by `setVestingController()` by the `owner`
  . Assign a new vesting controller or renounce their role by calling `setVestingController()`.
  . Create a plan, by calling `createPlan()`.
  . Update the book-keeping, by calling `release()`.
- `owner`, set to `msg.sender` when `initialize()` is called
  . Same as `vestingControllerAddress`.
  . Assign a new `owner`, by calling `transferOwnership()` or renounce the ownership, by calling `renounceOwnership()`.

Contract `EmployeeRegistry.sol` contains the following privileged roles:

- `owner`, set to `msg.sender` when `initialize()` is called
  . Assign a new `owner`, by calling `transferOwnership()` or renounce the ownership, by calling `renounceOwnership()`.

Contract `Escrow.sol` contains the following privileged roles:

- `owner`, set to `msg.sender` when `initialize()` is called
  . Assign a new `owner`, by calling `transferOwnership()` or renounce the ownership, by calling `renounceOwnership()`.
  . Assign a new `vestingControllerAddress`, by calling `setVestingControllerAddress()`.
  . Retrieve the **public** address `vestingControllerAddress`, by calling `getVestingControllerAddress()`.
  . Deposit, by calling `deposit()`.
  . Withdraw, by calling `withdraw()`, bypassing any vesting schedules in the process.
  . Set or unset the paused state by calling `pause()` and `unpause()` respectively, therefore allowing or preventing calls to `deposit()` and `withdraw()`.
- `vestingControllerAddress`, as initialized to `address(0)` or set by the `owner` when calling `setVestingControllerAddress()`:
  . Deposit, by calling `deposit()`.
  . Withdraw, by calling `withdraw()`.

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

Update: Some documentation about the privileged roles has been added in the form of comments in this PR but we don't consider the roles sufficiently documented.

QSP-5 `createVestPlan` Missing Input Validation

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `contracts/vesting/Vesting.sol`

Description: The `createVestPlan` function is the main entry point to creating a vesting plan. It makes implicit assumptions on the input parameters that are not enforced by the contract, which can lead to mistakes such as accidentally sending additional funds.

Recommendation: The function should validate the following:

- either linear or real-time can be specified but not both
- if the cliff duration is equal to the total duration, neither linear nor real-time can be specified
- if `_cliffDuration` is 0, `_cliffAmount` should be equal to 0 as well

Additional test cases for all of these scenarios should be added.

Update: Overall the function now performs more input validation, however it is still possible to specify linear parameters and also `_isRealtime=true`, which will lead to no real-time plan being created. The test that should assert this use case is broken (`When both realtime and linear plans are specified, should throw an error`). Also, consider removing `_totalAmount` and introduce a `_realtimeAmount` parameter since the linear plan no longer utilizes `_totalAmount`.

QSP-6 `EmployeeRegistry.isOwner()` Mishandling `address(0)`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/employee/EmployeeRegistry.sol`

Description: According to the comments, the function `EmployeeRegistry.isOwner()` should return `true` only if the provided address parameter is the owner of of the provided company id parameter and `false` otherwise. However, as the `exists` return value of `EnumerableMap.tryGet(companyOwners, _companyId)` is not checked, providing `address(0)` to any company id would lead to returning `owner == _employee` equal to `true` and therefore violating the comments assumption.

Recommendation: Add a check on `exists` and return `false`, if it is `false`.

Update: Check has been added in this PR.

QSP-7 `EmployeeRegistry.transferOwner` Might Lead to Unexpected State

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/employee/EmployeeRegistry.sol`

Description: There are two issues:

1. The function `transferOwner` does not check if `_newOwner` is an employee of the `_companyId`. The `_newOwner` could be an inactive employee, or even not an employee of that company. This would violate the assumption that a company owner should be an active employee of that company.
2. The function `transferOwner` does not check if `_newOwner` is not equal to `0x0`. This could lead to a disastrous result: A company has no owner to modify its data.

Recommendation: Add relevant checks to prevent these conditions from happening.

Update: Company ownership transfer now checks that the new owner is an active employee. Fixed in this PR.

QSP-8 Gas For Loop Concerns

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/vesting/Vesting.sol`

Description: When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding. For the loop on `L242`, the code does not restrict the number of vest plans that an employee in a company can have. This function might run out of gas and cannot be recovered from that state when the employee has too many vest plans. This could happen in case the company owner wants to deny an employee from accessing their vested tokens by adding many low-value vest plans.

Recommendation: Add a limit to the number of vest plans that an employee can have under a single company or add a function that accepts an index so employees can vest only a specific plan if required.

Update: The affected function has been removed and replaced by `vestById` in this PR.

QSP-9 `Escrow.deposit()` and `Escrow.withdraw()` Prone to Reentrancy

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/escrow/Escrow.sol`

Description: Although `Escrow.sol` inherits from `ReentrancyGuardUpgradeable`, it does not make use of the `nonReentrant` modifier. Further, functions `deposit()` and `withdraw()` do not comply with the "Checks-Effects-Interactions" pattern, by performing state variable changes **after** external function calls and not before (`escrowBalance[][]` being changed after calling `transferFrom()`, `balanceOf()` and `transfer()`). As these lower level functions are only callable by `owner` or `vestingControllerAddress` the potential impact is limited.

Recommendation: Add the `nonReentrant` modifier to `Escrow.deposit()` and `Escrow.withdraw()` and/or comply with the "Checks-Effects-Interactions" pattern, by updating `escrowBalance[][]` before performing the external calls.

Update: The `nonReentrant` modified has been added and the external call has been moved after the state variable update in this PR.

QSP-10 Functions Should Not be Payable

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/vesting/Vesting.sol`

Description: The functions `createVestPlan` and `vest` are declared `payable`, allowing ether to be sent along the function calls. Since there is no function to retrieve the ether, it will lead to locked funds.

Recommendation: Remove `payable` from the function signatures since receiving ether is not expected.

Update: `payable` has been removed in this PR.

QSP-11 Mismatched Interface Signatures

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/vesting/Vesting.sol`, `contracts/modules/CliffModule.sol`, `contracts/modules/LinearModule.sol`, `contracts/modules/StreamModule.sol`

Description: Some contracts implement the `IVest` or `IModule` interface without explicitly inheriting from them. This can lead to mismatched function signatures as is the case here. Some of the functions are missing the `view` specifier, none of them specify a return type and none of the modules implement a `getPlan` function.

Recommendation: Explicitly implement the interface and fix the interface signatures so they match the implementation.

Update: The interface file has been removed and was replaced by an abstract base class `BaseModule` that the modules implement in this PR.

QSP-12 `createPlan` Allows Overwriting Plans

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/modules/CliffModule.sol`, `contracts/modules/LinearModule.sol`, `contracts/modules/StreamModule.sol`

Description: The `createPlan` function in the modules contracts does not check whether a plan with the provided ID already exists, allowing the owner to overwrite any existing plan.

Recommendation: Either rename the function to clarify that this is the intended functionality (e.g. `createOrUpdatePlan`) or enforce that the plan does not exist already.

Update: It is no longer possible to call the modules' `createPlan` function directly. Therefore, only the incrementing counter from the vesting contract can be used for creating plans, which should no longer allow `createPlan` calls with existing IDs. Added in this PR.

QSP-13 Events Not Emitted on State Change

Severity: *Informational*

Status: Mitigated

File(s) affected: `contracts/modules/CliffModule.sol`, `contracts/modules/LinearModule.sol`, `contracts/modules/StreamModule.sol`, `contracts/employee/EmployeeRegistry.sol`, `contracts/escrow/Escrow.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. `CliffModule.setVestingController()`, after changing `vestingControllerAddress`.
2. `CliffModule.createPlan()`, after changing `cliffPlans[]`.
3. `CliffModule.release()`, after changing `cliffPlans[]`.
4. `LinearModule.setVestingController()`, after changing `vestingControllerAddress`.
5. `LinearModule.createPlan()`, after changing `linearPlans[]`.
6. `LinearModule.release()`, after changing `linearPlans[]`.
7. `StreamModule.setVestingController()`, after changing `vestingControllerAddress`.
8. `StreamModule.createPlan()`, after changing `streamPlans[]`.
9. `StreamModule.release()`, after changing `streamPlans[]`.
10. `EmployeeRegistry.addOwner()`, after changing `companyOwners`, and `companyId`.
11. `EmployeeRegistry.addEmployee()`, after changing `companyIdToEmployees[]` and `employeeToCompanyIds`.
12. `EmployeeRegistry.transferOwner()`, after changing `companyOwners`.
13. `EmployeeRegistry.archiveEmployee()`, after changing `companyIdToEmployees[]`.
14. `Escrow.deposit()`, after changing `escrowBalance[][]`.
15. `Escrow.withdraw()`, after changing `escrowBalance[][]`.

Recommendation: Emit an event in the aforementioned functions.

Update: Many events have been added (PR1, PR2) but none for an important update such as `setVestingController`. We highly recommend adding an event for this.

QSP-14 Missing Initializer Calls

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/employee/EmployeeRegistry.sol`, `contracts/escrow/Escrow.sol`, `contracts/vesting/Vesting.sol`

Description: Not all of the initializers from the inherited contracts are called. While this might not lead to issues with the current version, this can change in the future.

Recommendation: Call all of the initializers from the inherited contracts.

Update: The missing initializer calls have been added as recommended.

QSP-15 SafeMath Library Redundant

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/escrow/Escrow.sol`, `contracts/vesting/Vesting.sol`, `contracts/modules/CliffModule.sol`, `contracts/modules/LinearModule.sol`, `contracts/modules/StreamModule.sol`

Description: Starting from solidity version 0.8, overflows and underflows revert by default (see docs), rendering the use of SafeMath unnecessary.

Recommendation: Remove the library.

Update: The library has been removed in this PR.

QSP-16 Unnecessary `onlyOwner` Modifier

Severity: *Informational*

Status: Fixed

Description: The `getVestingControllerAddress` view function has the `onlyOwner` modifier. Since the `vestingControllerAddress` variable is public anyway, this serves no purpose.

Recommendation: Remove the modifier and potentially remove the function entirely since the variable is declared public and can be accessed already.

Update: The redundant function has been removed in this PR.

QSP-17 Name Clashes with Ownable

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/employee/EmployeeRegistry.sol`

Description: The `EmployeeRegistry` contract has several functions related to company ownership that are simply referred to as `owner`. Since the contract also inherits from `Ownable`, this might lead to confusion.

Recommendation: Rename variables and functions that relate to company ownership to make it clear they are distinct from the contract owner.

Update: The owner of a company is now referred to as `companyOwner` in the code. Changed in this PR.

QSP-18 Redundant Storage Variables

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/vesting/Vesting.sol`

Description: There are several member variables that contain identical values, one typed and one `address`. This is redundant since the address can be accessed by casting it like this: `address(cliffModule)`.
List of redundant variables:

```
address public employeeRegistryAddress;
address public cliffModuleAddress;
address public linearModuleAddress;
address public realtimeModuleAddress;
address public escrowAddress;
```

Recommendation: Remove the `address` variables.

Update: The redundant variables have been removed in this PR.

QSP-19 Missing Address Input Validation

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/escrow/Escrow.sol`, `contracts/vesting/Vesting.sol`, `contracts/modules/CliffModule.sol`, `contracts/modules/LinearModule.sol`, `contracts/modules/StreamModule.sol`

Description: It is important to validate inputs, even if they only come from trusted addresses in order to avoid human error. The following functions do not have proper validation of input parameters:

- `Escrow.initialize`
- `Vesting.initialize`
- `{Escrow,CliffModule,LinearModule,StreamModule}.setVestingControllerAddress`

Recommendation: Add checks that the input addresses are not equal to `address(0)`.

Update: Checks for `address(0)` have been added where recommended (PR1, PR2).

QSP-20 No Access Control on `EmployeeRegistry.addEmployee()` and `EmployeeRegistry.archiveEmployee()`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `contracts/employee/EmployeeRegistry.sol`

Description: The `EmployeeRegistry` contract does not employ any access control for adding or archiving employees, enabling anyone to add arbitrary addresses as *active employees* to arbitrary company ids or mark them as inactive. As a result, the distinction between active and inactive employees is currently pointless.

Recommendation: Clarify if this is an issue and if necessary, consider adding some kind of access controls to these functions.

Update: Only the vesting contract or the company owner can now modify existing companies. Fixed in this commit.

QSP-21 Incorrect Accounting for `EmployeeRegistry.employeeToCompanyIds`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `contracts/employee/EmployeeRegistry.sol`

Description: The function `EmployeeRegistry.addEmployee()` does not check the return value of the call `EnumerableSet.add(companyIdToEmployees[_companyId].activeEmployees, _employee);`, which would indicate whether the added employee was already in the set. Therefore, calling the function multiple times would lead to duplicate company id entries in `employeeToCompanyIds` after executing `employeeToCompanyIds[_employee].push(_companyId);`.

Recommendation: Add a check on the return value of `EnumerableSet.add()` and only execute the rest of the function if it was `true`, indicating the employee was not in the set before. Additionally, check if the employee is in the list of inactive employees already to avoid duplicates.

Update: Checks were added to only ever add an employee to `employeeToCompanyIds` if the employee has not been added before.

Automated Analyses

Slither

Slither reported 240 results, many of which have been identified as false positive. Relevant results have been incorporated in the findings of this report.

Code Documentation

1. Function comment on `Vesting.vest` states that this function checks whether the beneficiary can vest any of their modules when in reality it claims any vested amount.
2. `release` function in `contracts/modules/{CliffModule.sol,LinearModule.sol}` state that it *transfers tokens* when in fact it only returns the vested amount and performs some book-keeping.
3. In the different modules contracts, the comments on several functions are outdated and refer to some beneficiary or user. The current design only uses plan IDs and has no notion of users, as it does not store any addresses.

Adherence to Best Practices

1. Use `object.method()` whenever possible instead of `Contract.method(object)`, e.g. `vestPlanId.current()` instead of `Counters.current(vestPlanId)`.
   Update: Fixed.

2. Don't mix production and testing code (`contracts/lib/helper/Token.sol`).

3. Move struct declarations that are used in external function signatures into the interface files.

4. Remove the reentrancy guard from the modules since none of them perform any external calls and thus cannot be vulnerable to reentrancy.
   Update: Fixed.

5. Use the `+=` operator instead of `value = value + ...` constructs.
   Update: Fixed.

6. Tests contain a lot of duplicate code related to the deployment. We recommend looking into existing hardhat plugins that can help with this issue, e.g. hardhat-deploy.

7. Use the checks-effects-interactions pattern whenever external calls are involved.
   Update: Fixed in some cases but still some instances left.

8. To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the address parameters in `Vesting.CreateCompany()`, `Vesting.CreateVestPlan()`, `Vesting.Vest()`, `Vesting.Transfer()` and `Vesting.RemoveBeneficiary()`.
   Update: Fixed.

9. For improved code quality it is advised to remove unused imports.
   Update: Fixed.

   . Contract `CliffModule.sol` on L10, `Escrow.sol` on L17 and `Vesting.sol` on L17 contain an import to `hardhat/console.sol` which is test-related, unused and should be removed.

   . Contract `LinearModule.sol` on L8 and `StreamModule.sol` on L9 import `OwnableUpgradeable.sol` again, which are duplicates of L4, one of which should be removed.

   . Contracts `CliffModule.sol`, `LinearModule.sol` and `StreamModule.sol` import `IERC20Upgradeable.sol`, `SafeERC20Upgradeable.sol` and use `using SafeERC20Upgradeable for IERC20Upgradeable`, both of which are unused and should therefore be removed.

   . Contract `StreamModule.sol` imports `VestingWalletUpgradeable.sol`, but does not makes use of it, therefore that import should be removed.

   . Contract `EmployeeRegistry.sol` imports and inherits from `PausableUpgradeable`, but none of its functionality is used and should therefore be removed.

   . Contract `Escrow.sol` imports and inherits from `ReentrancyGuardUpgradeable`, but none of its functionality is used and should therefore be removed.

   . Contract `Escrow.sol` imports `CliffModule.sol`, `LinearModule.sol` and `StreamModule.sol` but does not use either of them, therefore these imports should be

. removed.

10. The function `getvesetevents` appears to be unused and should be removed.
    Update: Fixed.

11. For readability and auditability it is recommended to stick to the same naming convention when describing the same things. Contract `StreamModule.sol` always refers to `stream` plans, while `Vesting.sol` sometimes refers to them as `stream` and sometimes as `realtime` plans. This should be harmonized, e.g. by using `stream` plans throughout.
    Update: Partially fixed. There are still some references to real-time plans.

Test Results

Test Suite Results

Overall, the test suite seems fairly extensive. Upon manual inspection we identified that the amount of code duplication related to contract deployments is relatively high. To reduce that, we recommend reading up on hardhat plugins.
Additionally, since we identified several invalid function calls to the `createPlan` function that are not caught in the tests, we recommend adding more tests for that function.

Update:

More tests have been added as recommended.

```
yarn run v1.22.15
$ npx hardhat coverage

Version
=======
> solidity-coverage: v0.7.17

Instrumenting for coverage...
============================

> employee/EmployeeRegistry.sol
> escrow/Escrow.sol
> lib/helper/Token.sol
> lib/Utils.sol
> modules/BaseModule.sol
> modules/CliffModule.sol
> modules/LinearModule.sol
> modules/StreamModule.sol
> vesting/Vesting.sol

Compilation:
============

Compiling 23 files with 0.8.11
Generating typings for: 22 artifacts in dir: typechain-types for target: ethers-v5
Successfully generated 31 typings!
Compilation finished successfully

Network Info
============
> HardhatEVM: v2.8.0
> network:    hardhat

No need to generate any newer typings.


  Deploying Contracts
     ✓ Each contract should be deployed correctly (858ms)

  Upgading Contracts
     ✓ The modules and vesting controllers are upgradeable (326ms)

  Escrow
     ✓ An escrow can be deployed successfully and has the correct owner set (45ms)
     ✓ When depositing to the escrow, it should throw an error if the msg.sender isnt either the owner or the treasury wallet (77ms)
     ✓ When creating a vest plan, an escrow will be successfully funded (46ms)
     ✓ When calling vestById() a vest plan, an escrow will be successfully withdrawn (71ms)
     ✓ Calling from a non authorized address will revert and throw an error
     ✓ Calling deposit will succeed if called from the owner
     ✓ Calling withdraw will succeed if called from the owner (38ms)
     ✓ Calling reclaimUnvestedTokens will succeed if called from the owner (46ms)

  Employee Registry
     ✓ In a blank state, there should be no users or owners
     ✓ After adding an owner, the owner should be returned (41ms)
     ✓ Passing address(0) for a given _companyId to isCompanyOwner() should return false
     ✓ Adding an employee of 0x0 address should throw an error
     ✓ Passing a new owner of address 0x0 should throw an error
     ✓ Adding an employee that is inactive should remove them from the inactiveEmployees list and add them to the activeEmployee list (68ms)
     ✓ Archiving an active owner will throw an error
     ✓ Archiving an inactive employee will
     ✓ Transferring ownership to an inactive employee is not allowed (65ms)
     ✓ Transferring ownership to an non-employee(neither active not inactive) should throw an error
     ✓ Archiving a non-employee should throw an error
     ✓ Can transfer ownership of a company to another user (50ms)
     ✓ After transferring ownership, the previous owner no longer call vest() or archive() for the employees (288ms)
     ✓ An employee cant transfer ownership if they are not an owner
     ✓ Adding an employee to a company that doesnt exist yet will throw an error
     ✓ Archiving a nonexistent employee will throw an error
     ✓ Archiving an employee removes them from the active list & puts them in the inactive list
     ✓ Adding employees to 2 different companies should be tracked separately

  Cliff Module
     ✓ When creating a 1 year cliff, the cliffPlan should be configured correctly
     ✓ When creating a 1 year cliff in the past, the cliffPlan should be configured correctly
     ✓ When creating a cliff with an hour cadence, the cliffPlan should still be configured correctly
     ✓ When creating a 1 year cliff that starts in the future, canVest() should return false
     ✓ When creating a 1 year cliff that is in the past, canVest() should return true
     ✓ When creating a plan with an endTime that is before the startTime, should throw an error
     ✓ When creating a 1 year cliff that is in the past, release() should archive the cliff (54ms)
     ✓ When creating a 1 year cliff that is in the future, release() should not archive the cliffPlan (54ms)
     ✓ When calling createPlan() outside of the vesting controller, an error should be thrown
     ✓ When calling release() outside of the vesting controller, an error should be thrown

Linear Module
     ✓ When creating a 1 year linear plan, the linearPlan should be configured correctly
     ✓ When creating a 1 year linear plan in the past, the linearPlan should be configured correctly
     ✓ When a linear plan with a startTIme greater than the endTime, it should throw an error
     ✓ When a linear plan with a cadence of zero, it should throw an error
     ✓ When creating a 1 year linear plan in the future, canVest() should return false
     ✓ When creating a 1 year linear plan in the past, canVest() should return false if still not past the next cadenceDuration
     ✓ When creating a 1 year linear plan in the past, canVest() should return true if past the next cadenceDuration
     ✓ When creating a 1 year linear plan in the past, getReleasableAmount() should return 1 year worth of the supply
     ✓ When creating a 4 year linear plan that is fully vested in past, getReleasableAmount() should return the totalSupply of the plan
     ✓ When creating a 4 year linear plan that is in the future, getReleasableAmount() should return 0
     ✓ When creating a 1 year linear plan that is 3 months in the future, getReleasableAmount() should return (3 * cadenceAmount)
     ✓ When calling createPlan() as a non-owner, it should throw an error
     ✓ When calling release() as a non-owner, it should throw an error

Stream Module
     ✓ When creating a 1 year realtime plan in the past, the linearPlan should be configured correctly
     ✓ When creating a 1 year realtime plan in the future, the linearPlan should be configured correctly
     ✓ When creating a 1 year realtime plan, where the startTime is after the endTime, it should throw an error
     ✓ When creating a 1 year realtime plan in the future, canVest() should return false
     ✓ When creating a 1 year realtime plan in the past, canVest() should return true
     ✓ When creating a 1 year realtime plan one month in the past, getReleasableAmount() should return one month worth of tokens
     ✓ When creating a 7 day realtime plan 1 day, getReleasableAmount() should return one day worth of tokens
     ✓ When creating a 4 year realtime plan one month in the past, getReleasableAmount() should return one month worth of tokens
     ✓ When creating a 4 year realtime plan more than one month in the past, getReleasableAmount() should return at least more than one month worth of tokens
     ✓ When creating a 1 year realtime plan that is fully vestable, getReleasableAmount() should return the total supply
     ✓ When creating a 1 year realtime plan that starts in the future, getReleasableAmount() should return 0
     ✓ When calling createPlan as a non-owner, it should throw an error
     ✓ When calling release as a non-owner, it should throw an error

When archiving a vest plan
```

```
    ✓ 4 year plan, 1 year cliff, monthly thereafter should transfer tokens to employee if fully vestable and then archive (96ms)
    ✓ 4 year plan, 1 year cliff, monthly thereafter should return all tokens back to companyOwner if nothing is vestable when archiving (90ms)
    ✓ 4 year plan, 1 year cliff, realtime thereafter should return all tokens back to companyOwner if nothing is vestable when archiving (88ms)
    ✓ 1 year cliff should transfer tokens if fully vestable and then be archived (91ms)
    ✓ 4 year plan, no cliff and vesting realtime, should transfer tokens to employee when being archived (94ms)
    ✓ 4 year plan, no cliff and realtime vesting, should transfer tokens to employee when owner archives their plan (101ms)
    ✓ A vest plan that has no vestable tokens should be able to be archived (93ms)
    ✓ If a user has 2 vest plans, archiving 1 vest plan shouldnt archive the other vest plan (181ms)
    ✓ Calling archiveVestPlan() if not an owner or user should throw an error (41ms)
    ✓ 1 year plan that is vesting realtime will still be archived after tokens are trasnferred (99ms)

  When calling canVest()
    ✓ 4 year plan, 1 year cliff, monthly thereafter - canVest() should return false if within cliff (46ms)
    ✓ 4 year plan, 1 year cliff, streaming thereafter - canVest() should return false if within cliff (38ms)
    ✓ 4 year plan, 1 year cliff, monthly thereafter - canVest() should return true if after cliff (42ms)
    ✓ 4 year plan, 1 year cliff, streaming thereafter - canVest() should return true if after cliff
    ✓ 4 year plan, no cliff, monthly thereafter - canVest() should return false if not after first cadence (43ms)
    ✓ 4 year plan, no cliff, streaming thereafter - canVest() should return false if the current time isnt after the startTime (42ms)
    ✓ 4 year plan, no cliff, streaming thereafter - canVest() should return true if the current time is after the startTime (41ms)
    ✓ 4 year plan, no cliff, monthly thereafter - canVest() should return true if after first cadence (45ms)

  When calling canVestById()
    ✓ 4 year plan, 1 year cliff, monthly thereafter - canVestById() should return false if within cliff (54ms)
    ✓ 4 year plan, 1 year cliff, streaming thereafter - canVestById() should return false if within cliff (46ms)
    ✓ 4 year plan, 1 year cliff, monthly thereafter - canVestById() should return true if after cliff (46ms)
    ✓ 4 year plan, 1 year cliff, streaming thereafter - canVestById() should return true if after cliff
    ✓ 4 year plan, no cliff, monthly thereafter - canVestById() should return false if not after first cadence (45ms)
    ✓ 4 year plan, no cliff, streaming thereafter - canVestById() should return false if the current time isnt after the startTime (44ms)
    ✓ 4 year plan, no cliff, streaming thereafter - canVestById() should return true if the current time is after the startTime (43ms)
    ✓ 4 year plan, no cliff, monthly thereafter - canVestById() should return true if after first cadence (41ms)

  When a user has multiple vest plans
    ✓ 4 year plan, 1 year cliff monthly after && 4 year plan, 1 year cliff && monthly after. The two vest plans and company totalSupply should be correctly updated (129ms)
    ✓ 4 year plan 1 year cliff monthly && 4 year plan, 1 year cliff, streaming after. The two vest plans and company totalSupply should be correctly updated (133ms)
    ✓ 1 year plan 1 year cliff && 1 year plan 1 year cliff. The two vest plans and company totalSupply should be correctly updated (114ms)
    ✓ 1 year plan no cliff monthly && 1 year plan no cliff monthly after. The two vest plans and company totalSupply should be correctly updated (117ms)
    ✓ 1 year plan no cliff realtime && 1 year plan no cliff realtime. The two vest plans and company totalSupply should be correctly updated (115ms)
    ✓ 4 year plan 1 year cliff monthly && 1 year plan no cliff monthly. The two vest plans and company totalSupply should be correctly updated (124ms)
    ✓ 4 year plan 1 year cliff monthly && 1 year plan no cliff streaming. The two vest plans and company totalSupply should be correctly updated (122ms)
    ✓ 4 year plan, 1 year cliff monthly after && 4 year plan, 1 year cliff && monthly after. On double call to vest(), should not double vest (180ms)

  When calling vestById()
    ✓ 4 year plan, 1 year cliff, monthly thereafter should create both a cliffPlan and a linearPlan (42ms)
    ✓ 4 year plan, 1 year cliff, realtime thereafter should create both a cliffPlan and a streamPlan (42ms)
    ✓ 4 year plan, 1 year cliff, realtime thereafter in the future, should create both a cliffPlan and a streamPlan (43ms)
    ✓ 1 year plan, 1 year cliff should only create a cliffPlan (39ms)
    ✓ Calling createVestPlan() as a non-owner should throw an error (44ms)
    ✓ 1 year plan, 1 year cliff, monthly thereafter, should throw an error
    ✓ 1 year plan, 1 year cliff, realtime thereafter, should throw an error
    ✓ 1 year plan with 12 tokens total, 1 year cliff for 12 tokens, realtime thereafter, should throw an error
    ✓ 1 year plan with 12 tokens total, 1 year cliff for 12 tokens, nothing thereafter, should not throw an error
    ✓ When both realtime and linear plans are specified, should throw an error
    ✓ When cliff duration is 0, then the cliffAmount should also be 0
    ✓ 1 year plan, 1 year cliff with 1 token total, should create a cliffPlan with 1 token and no linear or stream plan (41ms)
    ✓ 1 year plan, no cliff with 1 token total with daily thereafter, should create a linearPlan with 1 token and no cliff or stream plans (43ms)
    ✓ when transferring ETH to the vesting contract, the contract should throw an error
    ✓ when a cliff length is greater than the plans total duration, it should throw an error
    ✓ for a 1 year plan with a 1 year cliff, the cliffAmount should equal the totalAmount, otherwise it should throw an error
    ✓ For a 4 year plan, 1 year cliff, monthly thereafter should be able to vest both the cliff and linear plan if possible  (54ms)
    ✓ For a 4 year plan, 1 year cliff, monthly thereafter and 2 years have passed, then should have vested exactly half  (68ms)
    ✓ For a 4 year plan, 1 year cliff, monthly after - 4 years have passed, then should have vested everything (65ms)
    ✓ For a 4 year plan, 1 year cliff, realtime thereafter should be able to vest both the cliff and realtime plan if possible  (61ms)
    ✓ For a 4 year plan, 1 year cliff, realtime thereafter - and 2 years have passed, should be able to vest half (64ms)
    ✓ For a 4 year plan, 1 year cliff, realtime thereafter - and 4 years have passed, should be able to vest everyting (64ms)
    ✓ For a 1 year plan, 1 year cliff - and 1 year has passed, should vest the cliff (66ms)
    ✓ can create a company
    ✓ a company struct is updated when a vestPlan is added
    ✓ a company struct is updated when a user vests partially their plan (57ms)
    ✓ a company struct is updated when a user fully vests their plan (63ms)
    ✓ a company struct is not updated when vest() is called but a user cant vest yet (59ms)
    ✓ calling vestById() twice doesnt result in a double vest (78ms)
    ✓ 173 day plan, no cliff, daily thereafter, started 70 days ago - should create only a linearPlan, calling vest() should release 40 tokens (88ms)
    ✓ 173 day plan, no cliff, daily thereafter, started 173 days ago - should create only a linearPlan, calling vest() should release all 173 tokens (89ms)
    ✓ 173 day plan, no cliff, daily thereafter, starting 7 days from now - should create only a linearPlan, calling vest() should not release any tokens (80ms)
    ✓ 173 day plan, no cliff, daily thereafter, started 173 days ago, total of 55555 tokens - should create only a linearPlan, calling vest() should release all 55555 tokens (89ms)


  131 passing (51s)
```

Code Coverage

Code coverage metrics are very good.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
employee/	100	77.78	100	100	
EmployeeRegistry.sol	100	77.78	100	100	
escrow/	100	62.5	100	100	
Escrow.sol	100	62.5	100	100	
lib/	100	100	100	100	
Utils.sol	100	100	100	100	
lib/helper/	100	100	100	100	
Token.sol	100	100	100	100	
modules/	96.08	84.78	100	96.08	
BaseModule.sol	100	75	100	100	
CliffModule.sol	95.65	80	100	95.65	155
LinearModule.sol	95.35	83.33	100	95.35	152,179
StreamModule.sol	96.88	92.86	100	96.77	141
vesting/	97.44	73.68	93.75	96.15	
Vesting.sol	97.44	73.68	93.75	96.15	76,80,245
All files	**97.6**	**78.13**	**98.41**	**97.22**	

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

```
fa09bb239e0b95dd4a736addfb6b8056fb3696ed01dcda7f3c93f9b574bb3580 ./contracts/vesting/Vesting.sol
91f5dbd6599e3abe108946bef9c418fb37cec699605d77cc14a4f49985f2d6f2 ./contracts/modules/BaseModule.sol
41dae1c429b2d7d7772d97b5987cb1a62f268aee267651083e96ec24ea5b6870 ./contracts/modules/CliffModule.sol
ebc1262ef5b0b8f298b192952c2a88b56164f302e172e0adcea3f175b09ba570 ./contracts/modules/LinearModule.sol
7d024653d884a9487c2a3ea3f167430e900aa8dd361ef0aed68953d007eb89e8 ./contracts/modules/StreamModule.sol
d292b8128f37e0875828d507f6d0d3c5d4c6798fcd35f51e9fc04bc4fba1734e ./contracts/lib/Utils.sol
73b4a1525c76ce7dd94778282139b5228b6ee47dd9c20166ad2abcca58850e0f ./contracts/escrow/Escrow.sol
7153c6cc36ce1d9097906da818c816704dc60e1e8c5f008de21658fb7475fe5b ./contracts/employee/EmployeeRegistry.sol
```

Tests

```
13d6a7fdae2d9ead72f7400dba59789f1de4b7a6fa5012728db0b552909f316a ./test/const.ts
dde66af642770b920dc79d8b96dd04b8075b6aa16ec28cfecb0f41b109495d7e ./test/vesting/ArchiveVestPlan.test.ts
9bc33f737d4b76693ad047d78d15b71f13ae569ca1152226e9ed19f66655062b ./test/vesting/CanVest.test.ts
c65dea903c60de322a22fbc1d00802c957ee9349bf9b622a35b5e1da8acc3974 ./test/vesting/CanVestById.test.ts
5b73d191fe86efe42ab5a16c24585cadb550297806dd59b263e78abd62d4905c ./test/vesting/MultipleVestPlans.test.ts
8159f22ae8d944419fcb49f36c0ef9b36c7c0f39147de994c896bcb81676a34e ./test/vesting/VestById.test.ts
09d613f5aba851b4b8bfe91782966ff3589698e85bb404f891f85626da792723 ./test/modules/CliffModule.test.ts
5510195944b473d8fc113a88948e0607f870315f25cc815eafa6462ac75d6d67 ./test/modules/LinearModule.test.ts
e22698bd93b4c9d326de691732975d47b14f6dee018a967b6dd422ae0a60bcb1 ./test/modules/StreamModule.test.ts
97fb5fe82ed25193a5064d9c8dff6e7bdc6690749812d887c846a54bd3f40b90 ./test/employees/EmployeeRegistry.test.ts
2c628f012617be2a9693794d8d493b3e0ce0a2f93fc9f1b5ca57e2b78540e994 ./test/employee/Escrow.test.ts
cb0fbfecff219754b882f7e01b6e9abe40b8f84036664c3a327d940b96cb275a ./test/deployments/Deployments.test.ts
ce93c2c44395260728fcfc8499d1b235b0c336c12cfd1bb935c19b35b5fefefe ./test/deployments/Upgrades.test.ts
```

Changelog

- 2022-01-27 - Initial report
- 2022-02-17 - Reaudit report

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.