Quantstamp Security Assessment Certificate

October 13th 2021 — Quantstamp Verified

Firestarter

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

Executive Summary

Type	Funding Platform
Auditors	Sebastian Banescu, Senior Research Engineer Roman Rohleder, Research Engineer Jake Bunce, Research Engineer
Timeline	2021-09-01 through 2021-09-22
EVM	London
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	GitBook
Documentation Quality	Medium
Test Quality	High

Source Code

Repository	Commit
Firestarter-Finance/contracts	eb96d7b
Firestarter-Finance/contracts	79f72e9

Total Issues	23 (23 Resolved)
High Risk Issues	3 (3 Resolved)
Medium Risk Issues	4 (4 Resolved)
Low Risk Issues	6 (6 Resolved)
Informational Risk Issues	8 (8 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 first audit: Quantstamp has performed a security audit of the Firestarter smart contracts and has identified 23 issues ranging across all severity levels. Additionally, we have found 11 deviations from best practices and several issues regarding specification and code comments. Moreover, the test suite only offers 85% branch coverage. We recommend that branch coverage be 100% to ensure that all code paths are tested at least once. We also strongly recommend addressing all the identified issues before deploying this project in production.

After reaudit: Quantstamp has checked the fixes provided by the project in commit `79f72e9`. The status of all the findings below have been updated accordingly.

ID	Description	Severity	Status
QSP-1	Initialization with multiple owners can be front-run	High	Fixed
QSP-2	Unsafe casts potentially leading to overflows	High	Fixed
QSP-3	KYC is not checked in any of the contracts	High	Fixed
QSP-4	Public presale period can be shortened	Medium	Fixed
QSP-5	Reentrancy possible when (un)locking tokens	Medium	Fixed
QSP-6	Vesting may start before unsold tokens are withdrawn	Medium	Fixed
QSP-7	Multiple owners increases the risk of a hack	Medium	Fixed
QSP-8	Denial-of-service due to integer underflow	Low	Fixed
QSP-9	Missing input validation	Low	Mitigated
QSP-10	Increased loss of precision	Low	Fixed
QSP-11	Local variable shadowing	Low	Fixed
QSP-12	Privileged roles and ownership	Low	Fixed
QSP-13	Contract Could Be Left Without Ownership	Low	Fixed
QSP-14	Uninitialized state variables	Informational	Fixed
QSP-15	Two different versions of `IERC20.sol`	Informational	Fixed
QSP-16	Ignored return value	Informational	Fixed
QSP-17	Events not emitted on state change	Informational	Fixed
QSP-18	Unlocked pragma	Informational	Fixed
QSP-19	Gas concerns / for-loops	Informational	Fixed
QSP-20	Obsolete check	Informational	Fixed
QSP-21	Unclear maximum allocation	Informational	Fixed
QSP-22	Wrong event emitted in `setEarlyWithdrawal`	Undetermined	Fixed
QSP-23	Incorrect accounting for `totalUsers`	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 v0.8.1

Steps taken to run the tools:

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

Findings

QSP-1 Initialization with multiple owners can be front-run

Severity: *High Risk*

Status: Fixed

File(s) affected: `Presale.sol, Whitelist.sol`

Description: The `initialize()` function takes an array of `owners` as an input parameter and gives all these addresses the `DEFAULT_ADMIN_ROLE`, same role given to the `msg.sender` in the same function. Given that the `initialize()` function does not return a particular value to indicate its successful execution and given that it might be executed by a script together with several subsequent contract calls, the legitimate owner who is making the call might be front-run by a malicious entity who inserts itself as one of the owners, alongside all other owners.

Exploit Scenario: Assuming a legitimate user deploys the `Presale` contract.

1. The legitimate user calls the `initialize()` function with a correct list of `owners`
2. An attacker monitors the mempool and notices the call to `initialize` and front-runs the call by making the same call with the address of the legitimate user inserted in the `owners` array and `_addrs.projectOwner` changed to the attacker's wallet.
   . This will lead to the attacker also being added as an owner because they are the `msg.sender`, but also the legitimate user being added.
   . Therefore, even if the call to `initialize` made by the legitimate user fails, all subsequent calls made by the legitimate user's script will succeed because that user is also an owner.

3. After the funds are added to the vesting contract and the sale is finished, the attacker will receive all the funds when `withdrawFunds` and `withdrawUnsoldToken` are called.
4. Alternatively the attacker can also call any of the functions protected by `onlyOwner` in order to disrupt the presale.

Recommendation: Make these contracts `Ownable` such that the role of the contract creator is distinguishable from that of the `DEFAULT_ADMIN_ROLE`. Only the contract creator/deployer should be able to call `initialize()`.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/5

QSP-2 Unsafe casts potentially leading to overflows

Severity: *High Risk*

Status: Fixed

File(s) affected: `Staking.sol`

Related Issue(s): SWC-101

Description: In L154, L158, L178, L202, L244, L259, L263, L288 of `Staking.sol` unsigned integers are cast/converted to signed integers, potentially leading to overflows, where very large unsigned values may become negative values after the conversion.

Recommendation: Replace the primitive cast operations with their safe counterparts from OpenZeppelin's `SafeCast` library, which is already used at some points throughout the contracts.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/2

QSP-3 KYC is not checked in any of the contracts

Severity: *High Risk*

Status: Fixed

File(s) affected: `Whitelist.sol`

Description: The `UserData` structure has a `isKycPassed` attribute, which is never used in the code. The documentation indicates that: "it should always be true but we have some exceptions for the Firestarter campaign". Therefore, we assume that it should be checked for all depositors in both the private and the public presales.

Recommendation: Check that the depositor has passed KYC.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/3

QSP-4 Public presale period can be shortened

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Presale.sol`

Description: The intention behind the `presalePeriod` state variable is to have a fixed size period for the public sale, which cannot be shortened even if the presale is paused. However, it is still possible to shorten the public presale period in case the private sale is not ended (by calling `endPrivateSale()`) before the `block.timestamp` passes the value of `startTime`. If this happens the `setStartTime()` function will no longer work and the public presale period would have effectively started, but the `deposit()` function cannot be called for the entire duration of the public presale. It would only be callable after `endPrivateSale()` is called.

Recommendation: Change the first `require` statement inside the `setStartTime()` function to also check if the `isPrivateSaleOver` flag is set to `true`, i.e.:

```
        require(
            startTime >= block.timestamp || isPrivateSaleOver == false,
            "setStartTime: Presale already started"
        );
```

Also check if the `startTime < block.timestamp` inside of `endPrivateSale()` and if so then set `startTime = block.timestamp` to ensure that the `presalePeriod` is properly enforced.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/16

QSP-5 Reentrancy possible when (un)locking tokens

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `TokenLock.sol`, `Staking.sol`, `Vesting.sol`, `Presale.sol`

Description: The `TokenLock.unlock()` function writes to the `lockInfo.amount` state variable after performing two ERC20 `transfer()` calls on L110 and L117. In the event that the `token` contract would be malicious, the `unlock()` function could be exploited through a reentrancy attack.
A similar issues occurs in the `TokenLock.lock()` function where `lockInfo` and `totalLocked` are changed after the call to the `transferFrom()` function of `token`.
A less severe issue occurs in the following functions because only events are emitted after external contract calls and no state variables are modified:

- `Staking.deposit()` emits `Deposit` after the call to `lpToken.safeTransferFrom()`.
- `Staking.emergencyWithdraw()` emits `EmergencyWithdraw` after the call to `lpToken.safeTransfer`.
- `Staking.harvest()` emits `Harvest` after the calls to the external `FLAME` contract.
- `Staking.withdraw()` emits `Harvest` and `Withdraw` after the calls to the external `FLAME` and `lpToken` contracts.
- `Vesting.withdraw()` emits `Withdraw` after the call to `rewardToken.transfer()`.
- `Presale.withdrawFunds` emits `WithdrawFunds` after the calls to `fundToken`.
- `Presale.withdrawUnsoldToken()` emits `WithdrawUnsoldToken` after the call to `rewardToken.trasferFrom()`.

Recommendation: Employ the "Checks-Effects-Interactions" pattern by moving the interactions with the external contract after all effects such as state variable assignments and emission of events.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/4, with one exception: `Vesting.withdraw()` still emits `Withdraw` after the call to `IERC20(rewardToken).safeTransfer()`. The emit should be done before that call.

QSP-6 Vesting may start before unsold tokens are withdrawn

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Presale.sol`

Description: According to the "Campaign Timeline" in the documentation, vesting should only start after the unsold tokens are withdrawn. Otherwise, one of the vesting users could call the `Vesting.withdraw()` function before one of the owners calls `Presale.withdrawUnsoldToken()`. If this were to happen, the `unsoldAmount` computed by the `Presale.withdrawUnsoldToken()` function would be lower than the actual unsold amount and the `projectOwner` would only receive this lower amount. The token difference would be stuck in the `Vesting` contract.

Recommendation: Add an additional flag inside the `Presale` contract and set it to `true` only when `withdrawUnsoldToken()` is called. The `Presale.startVesting()` contract should only be allowed to be called when the flag is `true`.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/17

QSP-7 Multiple owners increases the risk of a hack

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Presale.sol`, `Whitelist.sol`

Description: The `Presale` and `Whitelist` contracts have multiple addresses which may perform privileged function calls. This design decision increases the likelihood of a hack because if ANY of the addresses is compromised, then an attacker could tamper with the presale or whitelist.

Recommendation: To enable multiple users to be able to perform privileged actions, use a 2-of-N multi-sig address as the sole owner of the contract. Such a multi-sig will prevent a single compromised address from doing any damage. Moreover, the compromised address can be removed and a new address added instead. Addresses can be added or removed at any time, which offers more flexibility than the current version which has a fixed number of owner addresses provided in the `initialize()` function call.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/5. The project developers have indicated that they will use a multi-sig wallet on this page. However, this should be checked by end-users post deployment.

QSP-8 Denial-of-service due to integer underflow

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Staking.sol`

Description: There is a potential integer overflow on L178 in `Staking.totalRewards()`, in case `totalRewardDebt > flamePerSecond * stakingPeriod`, because the result is cast to `unit256`. This underflow is caught by the `SafeCast.toUint256()` function. However, it would cause an effective denial-of-service to all the functions of the `Staking` contract, which are using this function.

Recommendation: Subtract `totalRewardDebt` after adding `total` and `flamePerSecond * stakingPeriod`, i.e. L178 should look like:

```
total = (int256(accTotalRewards + flamePerSecond * stakingPeriod) - totalRewardDebt).toUint256();
```

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/19

QSP-9 Missing input validation

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `TokenLock.sol`, `Vesting.sol`, `Presale.sol`

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

1. `Vesting.init()` does not check if the `presale` parameter is different from `address(0)`.
2. `Vesting.initialize()` does not check if the `_rewardToken` parameter is different from `address(0)`. This function does not check if the `_param` parameter attributes are in the permitted ranges, e.g. `_params.initialUnlock > block.timestamp`.
3. `TokenLock.initialize()` does not check if the `_token` parameter is different from `address(0)`.
4. `Presale.initialize()` does not check if the `AddressParams` are different from `address(0)`. Note that if the `projectOwner` would be `address(0)` by mistake, the `withdrawFunds()` and `withdrawUnsoldToken()` functions would no longer work.
5. `Presale.initialize()` does not check if the `PresaleParams` are greater than `0` and most importantly that `_presale.startTime > block.timestamp`.
6. `Staking.initialize()` does not validate that the `_lpToken` and `_flame` input parameters are different than `address(0)`. This function does not check if the `_startTime > block.timestamp` and `_earlyWithdrawal` is not checked to be shorter than `stakingPeriod`

Recommendation: Add `require` statements to validate the input parameters mentioned above.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/20, with the exception of `Vesting.initialize()`, which does not check that `_params.initialUnlock > block.timestamp`.

QSP-10 Increased loss of precision

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Vesting.sol`

Description: Integer division truncates the result, which leads to a loss of precision. When division is followed by a multiplication, this loss of precision is further increased. Two instances of this issue were observed in the `Vesting.vested()` function:

1. The division on L260: `.div(accuracy)` is performed before the multiplication of the result (i.e. `unlockAmountPerInterval`) on L262.
2. The division on L262: `.div(releaseInterval)` is performed before the multiplication on L262: `.mul(unlockAmountPerInterval)`.

Recommendation: To fix both issues simply change the order of the operations on L262:

```
uint256 vestedAmount = block.timestamp
    .sub(lockEndTime)
    .mul(unlockAmountPerInterval)
    .div(releaseInterval)
    .add(initialUnlockAmount);
```

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/21

QSP-11 Local variable shadowing

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CustomToken.sol`

Description: The `name` and `symbol` input parameters of the `CustomToken.constructor` are shadowing the state variables of the `ERC20` contract that `CustomToken` inherits from. As a result the use of these local variables might be incorrect.

Recommendation: Rename the input parameters to `_name` and `_symbol` respectively.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/22

QSP-12 Privileged roles and ownership

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Presale.sol`, `Vesting.sol`, `Staking.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. Any of the owners of the `Presale` contract may perform the following privileged actions:

1. End the private sale at any point in time.
2. Set the start time of the public sale to any point in the future as long as the public sale has not started yet.
3. Start the public sale at any point in time as long as it has not started yet AND the private sale has ended AND there are sufficient reward tokens deposited in the vesting contract.
4. Pause the public sale at any point in time.
5. Resume the public sale if it is paused.
6. Withdraw the service fee amount, after the public sale has ended, to any address they feed in as the `treasury` parameter.
7. Withdraw the fund tokens and the unsold reward tokens to the `projectOwner` address after the public sale has ended.

The owner of the `Vesting` contract may perform the following privileged actions:

1. Add/update vesting recipients to any amount greater than zero at any time, as long as the `totalVestingAmount <= depositedAmount`. NOTE that this may be done before `Vesting.init()` is called.
2. Set the presale contract to any address at any point in time.
3. Set the start time to any point in the future as long as vesting has not already started and `Vesting.init()` has not yet been called.

The owner of the `Staking` contract may perform the following privileged actions:

1. Set the early withdrawal time to any amount, at any point in time.
2. Set the staking info before staking starts.
3. Set the flame per second to be distributed at any point in time.

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

Update: These privileged roles and actions were documented in publicly available documentation on this page

QSP-13 Contract Could Be Left Without Ownership

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Staking.sol`

Description: `renounceOwnership()` from OpzenZeppelin allows an actor with the `onlyOwner` role to rennounce the ownership of the contract by setting the owner to the zero address. This would prevent any further functions with `onlyOwner` to be callable.

Recommendation: Override `renounceOwnership()` in the contract with a `revert()` so that the OZ library can be used without the renouncement of the ownership.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/23

QSP-14 Uninitialized state variables

Severity: *Informational*

Status: Fixed

File(s) affected: `Presale.sol`, `ProjectPresale.sol`

Description: State variables should always be initialized before they are used, unless they are explicitly stated (in the comments) to be initialized to the default value, e.g. `0x0` for `address`. The following instances of this issue have been observed:

1. `Presale.privateSoldAmount` is never initialized. It is used in `Presale.withdrawUnsoldToken()`
2. `Presale.publicSoldAmount` is never initialized. It is used in `Presale.withdrawUnsoldToken()`
3. `Presale.privateSoldFunds` is never initialized. It is used in `Presale.deposit(uint256)`
4. `Presale.recipients` is never initialized. It is used in `Presale.deposit(uint256)` and `ProjectPresale.depositPrivateSale(uint256)`
5. `Presale.participants` is never initialized. It is used in `Presale.participantCount()` and `Presale.getParticipants(uint256,uint256)`

Recommendation: Either initialize the state variables or document (using code comments) that they are intended to be initialized to the default value.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/24

QSP-15 Two different versions of `IERC20.sol`

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/interfaces/IERC20.sol`, `@openzeppelin/contracts/token/ERC20/IERC20.sol`

Description: The installed project contains 2 different instances of `IERC20.sol`:

- One from the `openzeppelin` contracts package
- One custom version inside the `interfaces/` sub-directory.

The only difference between these 2 versions is that the latter version contains 4 additional functions w.r.t. the former version:

- `function decimals() external view returns (uint8);`
- `function symbol() external view returns (string memory);`
- `function name() external view returns (string memory);`
- `function getOwner() external view returns (address);`

It is not recommended to have any contract or interface with the same name in the same build, because it creates confusion since some of the contracts import one file and some the other. Moreover, the last 3 functions in the enumeration above do not seem to be called by any function in the repository.

Recommendation: Use only one of the two `IERC20.sol` files.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/6

QSP-16 Ignored return value

Severity: *Informational*

Status: Fixed

File(s) affected: `Vesting.sol`

Description: The following instance of this issue was observed: the call to `IERC20(rewardToken).approve()` is ignored on L140 in `Vesting.init()`. This could lead to unexpected errors when calling other functions that are expecting the `presale` contract to be approved for transferring the `rewardToken`.

Recommendation: Use the return value of the call to `approve()` by either reverting the transaction if it returns `false` or add a return value to the `init()` function and return the same value as the call to `approve()`.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/7

QSP-17 Events not emitted on state change

Severity: *Informational*

Status: Fixed

File(s) affected: `Vesting.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 `Vesting.init()` function, which does not emit any event upon a successful change of the `owner` state variable.

Recommendation: Emit an event in the aforementioned function

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/25

QSP-18 Unlocked pragma

Severity: *Informational*

Status: Fixed

File(s) affected: `CustomToken.sol`, `FirestarterPresale.sol`, `Presale.sol`, `ProjectPresale.sol`, `TokenLock.sol`, `Vesting.sol`, `Whitelist.sol`

Description: The version of the Solidity defined in the code is not fixed. This can lead to unintended or unexpected behaviours in the implementation due to different compiler versions between what is intented in the code and the output at compile time.

Recommendation: Statically define the version of the Solidity compiler using `pragma solidity 0.8.0`, where the desired release of the `0.8` train is denoted.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/8

QSP-19 Gas concerns / for-loops

Severity: *Informational*

Status: Fixed

File(s) affected: `Whitelist.sol`, `Presale.sol`

Description: In L54 of `Whitelist.sol` and in L198 of `Presale.sol` for-loops are used to iterate over the length of the user-supplied arrays `owners`. Since the corresponding functions have the `initializer` modifier they can not be called multiple times with smaller sized arrays, effectively denying their service for sufficiently large `owners` arrays.

Functions `addToWhitelist` and `removeFromWhitelist` contain for-loops which could run out of gas in case the input parameter array length is too large.

Recommendation: Add a check on the array length, to prevent arrays larger than a set maximum being processed. This maximum can be determined by performing gas analysis.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/9

QSP-20 Obsolete check

Severity: *Informational*

Status: Fixed

File(s) affected: `Presale.sol`

Description: The `startPresale()` function uses the `whileDeposited` modifier which checks:

```
    modifier whileDeposited() {
        require(
            _getDepositedRewardTokenAmount() >= initialRewardAmount,
            "Deposit enough rewardToken tokens to the vesting contract first!"
        );
        _;
    }
```

Therefore, the check on L276-279 is obsolete (superseded):

```
        require(
            _getDepositedRewardTokenAmount() != 0,
            "startPresale: Please deposit rewardToken tokens to vesting contract first!"
        );
```

Recommendation: Remove the check on L276-279 to save gas and improve code readability.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/10

QSP-21 Unclear maximum allocation

Severity: *Informational*

Status: Fixed

File(s) affected: `Whitelist.sol`, `Presale.sol`

Description: The following check in `Presale.deposit` is unclear:

```
        require(
            maxAlloc + privateSoldFunds[user] >= newFundBalance,
            "Deposit: Can't exceed the maxAlloc!"
        );
```

The comment above the `maxAlloc` declaration in `Whitelist.sol` indicates that it represents: "Max allocation for this user". However, according to the check above it only represents the maximum allocation for the public sale. If it were the absolute maximum allocation, then the check inside `Presale.deposit` should be: `maxAlloc >= newFundBalance`.

Recommendation: Clarify the meaning of `maxAlloc` via clear code comments and if needed adjust the implementation accordingly.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/11

QSP-22 Wrong event emitted in `setEarlyWithdrawal`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `Staking.sol`

Description: In L107 of `Staking.sol` of function `setEarlyWithdrawal` the event `LogFlamePerSecond` is emitted, however event `LogEarlyWithdrawal` exists and seems should've been used instead.

Recommendation: Replace `LogFlamePerSecond` with `LogEarlyWithdrawal` in L107 of `Staking.sol`.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/12

QSP-23 Incorrect accounting for `totalUsers`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `Whitelist.sol`

Description: In L91 of `Whitelist.sol`, if the provided `users` array contains already whitelisted users, the update of `totalUsers` will be incorrect, in that it will incorrectly count them as also being new, resulting in `totalUsers` being larger than the actual number of whitelisted users.

Recommendation: Consider having a separate counter within the if-statement in L84 and then add that variable to `totalUsers` in L91, instead of `users.length`.

Update: Fixed in PR https://github.com/Firestarter-Finance/contracts/pull/13

Automated Analyses

Slither

We have run the latest version of the slither analyzer on this repository's solidity code. The tool has identified 174 issues, most of which were filtered out as false positives. The remaining issues were integrated in the findings of this report.

Adherence to Specification

1. **[Fixed]** The comment on L48 in `Presale.sol` indicates that: "Service Fee : eg 1e5 = 10% default is 5%". However, the default value of 5% is not present in the implementation.
2. Specification is lacking a proper description for `setStakingInfo` of `Staking.sol`.
3. The specification states that the `early withdrawal will be reset to present date value` for `deposit` of `Staking.sol`, but in fact the `lastDepositedAt` is updated.
4. There is a typo in the documentation for `addToWhitelist` of `Whitelist`, speaking of a `waitlist`, which should've been `whitelist`.
5. The method description for `startVesting` of `Presale` is inconsistent with the implementation. The specification describes that the presale contract is set as the owner for the vesting contract, however, the function just sets the vesting start time to `block.timestamp + 1`.

Code Documentation

1. **[Fixed]** The README.md file is empty. It should at the minimum include:
   . the directory structure and a description of the files each sub-folder contains.
   . the system requirements, e.g. node.js version.
   . the steps needed to install and run the (coverage) tests.
2. **[Fixed]** In L133 of `Presale.sol` the event below is commented with `presale is started` (a copy-and-pase from L127), however the event is called `PresaleResumed`.
3. **[Fixed]** In L299 of `Presale.sol` the comment states `Start presale`, while more precisely the function `resumePresale` resumes an already started but currently paused presale.
4. **[Fixed]** In L415 of `Presale.sol` the comment states `Check if Presale is in progress` (a copy-and-paste from L423), however, the corresponding function `startVesting` rather starts vesting, by calling `setStartTime` of the vesting contract.

Adherence to Best Practices

1. **[Fixed]** Starting with Solidity version 0.8.0 the `SafeMath` and `SignedSafeMath` checks have been integrated into the compiler and therefore, it is recommended to avoid using the openzeppelin `SafeMath` and `SignedSafeMath` libraries to improve readability and save gas. Solidity 0.8.0 provides the same security guarantees.
2. **[Unresolved]** `Vesting` should inherit from `IVesting`.
3. **[Fixed]** The code does not conform to solidity naming conventions.
4. **[Fixed]** Commented code should be removed from production code. The following instances were observed:
   . L77 in `Presale.sol`: `// uint256 public goalFunds;`
   . L212 in `Presale.sol`: `// goalFunds = _presale.goalFunds;`
   . L184-185 inside `Vesting.setStartTime()`.
5. **[Unresolved]** Code cloning should be avoided and code reuse should be favored. The following code clone instances were observed:
   . A large part of the `Staking.withdraw()` and `Staking.harvest()` functions are identical. And there is one check missing in the diff, which shows why code cloning is bad.
   . A large part of the `Staking.pendingFlame()` and `Staking.updatePool()` functions are identical.
6. **[Fixed]** The ABI decoder statement is valid but deprecated and therefore has no effect. It should be removed from all contracts that use it. Change the ABI coder

statements from `pragma experimental ABIEncoderV2;` -> `pragma abicoder v2;`.

7. **[Fixed]** Address parameters in events should be indexed. For example, the following events have address parameters that are not indexed: L33 and L36 of `TokenLock.sol`, L91 and L94 of `Vesting.sol`, L43 of `Whitelist.sol` and L20 of `interfaces/IWhitelist.sol`.

8. **[Fixed]** For better disambiguation the struct variable `maxAlloc` in L24 of `Whitelist.sol` should be renamed to `publicMaxAlloc` (including all its uses), since there also already exists a `privateMaxAlloc`.

9. **[Fixed]** For improved readability it is recommended to have a maximum line length of 79 or 99. Therefore L141 of `Whitelist.sol`, L23, L27, L42, L56, L65, L68, L129, L185 and L262 of `Vesting.sol`, L154, L171, L200, L202, L213, L223, L244 and L263 of `Staking.sol`, L33 and L37 of `ProjectPresale.sol` and L56 and L228 of `Presale.sol`, which exceed these limits, should be shortened accordingly.

10. **[Fixed]** In accordance with best practices constant variables should be written in all capital letters. Variable `accuracy` in L43 of `Vesting.sol` and L57 in `Presale.sol`, which is a constant, should therefore be renamed to `ACCURACY` and correspondingly all its uses.

11. **[Fixed]** It is recommended to remove unused variables. As variable `goalFunds` of the `PresaleParams` struct in L51 of `Presale.sol` is unused, it should be removed.

Test Results

Test Suite Results

After first audit: We confirm that all 131 tests are passing. However, we recommend implementing Mainnet forking for tests to be executed using popular tokens as deployed on Mainnet, e.g. USDC and USDT, which are notorious for bringing issues to the surface.

After reaudit: 21 new tests were added and we confirm that all 152 tests are passing on our end.

```
Firestarter Presale
  depositPrivateSale
     ✓ Only owners can do this operation (137ms)
     ✓ Can do this only when enough amount is deposited (67ms)
     ✓ Can't deposit if private sale is over (68ms)
     ✓ Recipient info is updated (86ms)
     ✓ Can deposit full allocation amount in private and public sale (146ms)
     ✓ Vested event is emmitted with correct params (59ms)
  analysis support
     ✓ participants list - private and public (232ms)
     ✓ participants list - no duplication (266ms)
     ✓ participants list - if duplicated, no update to the list (89ms)
     ✓ participants list - pagination (376ms)

Locking
  initialize
     ✓ Validiation of initilize params (1226ms)
  lock
     ✓ Correct amount is locked
     ✓ lock amount is stacked (47ms)
     ✓ Locked event is emitted with correct params
  getPenalty
     ✓ Revert if none locked
     ✓ Penalty is correct per passed days (47ms)
  unlock
     ✓ Revert if nothing locked
     ✓ Cannot unlock more than locked amount
     ✓ Before 10 days(ensure correct amount burned, transferred, subtracted) (66ms)
     ✓ Before 20 days(ensure correct amount burned, transferred, subtracted) (55ms)
     ✓ Before 30 days(ensure correct amount burned, transferred, subtracted) (62ms)
     ✓ After 30 days(Ensure correct amount burned, transferred, subtracted) (56ms)
     ✓ Unlocked event is emitted with correct params (38ms)

Presale
  initialize
     ✓ Validiation of initilize params (1351ms)
  endPrivateSale
     ✓ Only owners can do this operation (43ms)
     ✓ Set startTime to now if presale is already started
     ✓ PrivateSaleDone event is emitted with correct params
  setStartTime
     ✓ Only owners can do this operation (59ms)
     ✓ Cannot set if private sale is over and presale alredy started
     ✓ Can set if private sale is over and presale is not started
     ✓ Can set if private sale is not over, though presale is already started
     ✓ Must set future time
     ✓ Time is set/event emitted
  startPresale
     ✓ Only owners can do this operation (76ms)
     ✓ Enough amount should be deposited (50ms)
     ✓ Private presale must have ended
     ✓ Can't be called if already started (51ms)
     ✓ PresaleManuallyStarted event is emitted with correct params (39ms)
     ✓ startTime is reset to that timestamp (44ms)
  Pause
     ✓ Only owners can do this operation (99ms)
     ✓ Can only be called while on going (82ms)
     ✓ Status variables are correctly set (55ms)
     ✓ PresalePaused event is emitted with correct params (51ms)
  Resume
     ✓ Only owners can do this operation (166ms)
     ✓ Can only be called while paused (58ms)
     ✓ Status variables are correctly set (67ms)
     ✓ PresaleResumed event is emitted with correct params (64ms)
  isPresaleOnGoing
     ✓ Ongoing by manual start (74ms)
     ✓ Ongoing auto start and end - should be false if private sale not ended (55ms)
     ✓ Private sale ended, but not enough reward token deposited (54ms)
     ✓ Pause and resume (75ms)
  Deposit
     ✓ Can only be called when ongoing (42ms)
     ✓ Must be whitelisted user (57ms)
     ✓ Must be kyc passed user (57ms)
     ✓ Can't exceed publicMaxAlloc (231ms)
     ✓ Deposit updates correct states (94ms)
     ✓ deposit amount is stacked (131ms)
     ✓ Vested event is emitted with correct params (79ms)
  Start vesting
     ✓ Only owners can do this operation (100ms)
     ✓ Can only be called when finished and unsold token is withdrawn (163ms)
     ✓ Vesting starts correctly (71ms)
  Withdraw funds
     ✓ Only owners can do this operation (91ms)
     ✓ Can only be called when finished (61ms)
     ✓ Cannot withdraw to zero address (49ms)
     ✓ Correct amount is withdrawn (212ms)
     ✓ WithdrawFunds event emitted with correct params (394ms)
  Withdraw Unsold tokens
     ✓ Only owners can do this operation (80ms)
     ✓ Can only be called when finished (60ms)
     ✓ Set unsoldTokenWithdrawnFlag to true (60ms)
     ✓ Correct amount is withdrawn (216ms)
     ✓ WithdrawUnsoldToken event emitted with correct params (211ms)

Project Presale
  Deposit PrivateSale
     ✓ Can do this only when enough amount is deposited (63ms)
     ✓ Can't deposit if private sale is over (63ms)
     ✓ Must be whitelisted user
     ✓ Must be kyc passed user
     ✓ Must be private sale allowed user (45ms)
     ✓ Can't exceed publicMaxAlloc (61ms)
     ✓ Deposit updates correct states (67ms)
     ✓ Can deposit full allocation amount in private and public sale (119ms)
     ✓ deposit amount is stacked (106ms)
     ✓ Vested event is emitted with correct params (48ms)
  analysis support
     ✓ participants list - private and public (302ms)
     ✓ participants list - no duplication (351ms)
     ✓ participants list - pagination (417ms)

Staking Pool
  initialize
```

```
    ✓ Validiation of initilize params (1898ms)
  Deposit/withdraw reward token
    ✓ Only owner can do these operation (52ms)
  Set penalty period
    ✓ Only owner can do these operation
    ✓ Cannot exceed staking period
    ✓ It correctly updates information
  Set Flame per second
    ✓ Only owner can do these operation
    ✓ It correctly updates information
  Set staking info
    ✓ Only owner can do these operation
    ✓ Fails if staking is in progress (42ms)
    ✓ New startTime must be in the future
    ✓ It correctly updates information
    ✓ Must update lastRewardTime always (41ms)
  Deposit
    ✓ Pool should be open (42ms)
    ✓ Deposit 0 amount
    ✓ Staking amount increases (46ms)
  Total rewards
    ✓ Should involve several periods (118ms)
    ✓ FlamePerSecond is updated serveral times in one staking period (68ms)
    ✓ Period 1 -> setStakingInfo -> setFlamePerSecond -> Period 2 (84ms)
    ✓ Period 1 -> setFlamePerSecond -> setStakingInfo -> Period 2 (77ms)
  PendingFlame
    ✓ Should be zero when lp supply is zero (40ms)
    ✓ Should be zero always before staking starts
    ✓ PendingFlame should equal ExpectedFlame
    ✓ Deposit while staking is in progress (72ms)
    ✓ Staking is finished
    ✓ New staking is set
    ✓ New staking is started (52ms)
  Update Pool
    ✓ LogUpdatePool event is emitted (45ms)
  Harvest
    ✓ Should give back the correct amount of FLAME (58ms)
    ✓ Should receive half in case of penalty (54ms)
    ✓ Harvest with empty user balance
  Withdraw
    ✓ Should give back the correct amount of lp token and harvest rewards(withdraw whole amount) (63ms)
    ✓ Should receive half in case of penalty(withdraw 1/4 amount) (150ms)
    ✓ Withraw 0
  EmergencyWithdraw
    ✓ Should emit event EmergencyWithdraw
  Renoucne Ownership
    ✓ Should revert when call renoucne ownership

Vesting
  initialize
    ✓ Validiation of initilize params (1267ms)
  init
    ✓ Only owner can call this function
    ✓ Cannot set zero address
    ✓ Init updates the owner (45ms)
  updateRecipient
    ✓ Only owner can call this function (38ms)
    ✓ Cannot vest 0
    ✓ Cannot update the recipient after started
    ✓ Cannot set more than total amount (43ms)
    ✓ Recipient amount is updated.
    ✓ VestingInfoUpdated event is emitted.
  setStartTime
    ✓ Cannot set if alredy started
    ✓ Must set future time (126ms)
    ✓ Time is set/event emitted
  vested
    ✓ Should be zero if not started
    ✓ Should be zero if in lockPeriod
    ✓ Correct amount should be vested during the lockPeriod (4879ms)
    ✓ Full amount should be released after vesting period
  withdraw
    ✓ If zero, nothing happens
    ✓ Correct amount is withdrawn/event is emitted (90ms)
    ✓ withdrawable amound decrease / amountWithdrawn is updated (55ms)
    ✓ withdrawable amound decrease / amountWithdrawn is updated (69ms)
  analysis support
    ✓ participants list (130ms)
    ✓ pagination (100ms)

Whitelist
  addToWhitelist
    ✓ Security (146ms)
    ✓ Input length shouldn't exceed MAX_ARRAY_LENGTH (227ms)
    ✓ Attempt to add one user. AddedOrRemoved event is emitted.
    ✓ Attempt to add multiple users. AddedOrRemoved event is emitted. (193ms)
  removeFromWhitelist
    ✓ Security (42ms)
    ✓ Input length shouldn't exceed MAX_ARRAY_LENGTH (78ms)
    ✓ Attempt to remove one user. AddedOrRemoved event is emitted. (45ms)
    ✓ Attempt to remove multiple users. AddedOrRemoved event is emitted. (178ms)
  analysis support
    ✓ users list (135ms)
    ✓ users list - remove (197ms)


152 passing (13m)
```

Code Coverage

After first audit: Even though line coverage is high, we noticed that the branch coverage is significantly lower (~85%). This means that not all branches of the code are tested. We recommend increasing the branch coverage to 100%.

After reaudit: The branch coverage has been increased to 100%, which is in accordance to our recommendation.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
CustomToken.sol	100	100	100	100	
FirestarterPresale.sol	100	100	100	100	
Presale.sol	100	100	100	100	
ProjectPresale.sol	100	100	100	100	
Staking.sol	100	100	100	100	
TokenLock.sol	100	100	100	100	
Vesting.sol	100	100	100	100	
Whitelist.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
IERC20Extended.sol	100	100	100	100	
IVesting.sol	100	100	100	100	
IWhitelist.sol	100	100	100	100	
contracts/libraries/	100	100	100	100	
AddressPagination.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

```
364502e38088d5482ef7130e243a1a54722ebb0dcf213d4bf6a5a8e9cc3a5e7f ./contracts/TokenLock.sol
41ff368391a3d96443ef24b1ea09399899c7bf1c1af3394cf80c0fdea9058530 ./contracts/FirestarterPresale.sol
456da22838c22d7df7fa00366a77c8619f1b5acbd9f26f0be8a53ddcf4ad1aad ./contracts/Whitelist.sol
4c2074a5fbfb447b497aeb970d3fe1fdfe158a891c0a3942bba64b67661b34e5 ./contracts/Presale.sol
1a49b866bf90b5788fafb05fd83e7f39af9a731636ae6b246f0e7fb285ad6ce5 ./contracts/CustomToken.sol
e0b061b834661ff9446a33f8e360f90d550f305efdd2643eacfdb993775d02b1 ./contracts/ProjectPresale.sol
8ba28b2c61b00d98f93194858632b2c50483b9c81c29db007fd5b21a3628ea6e ./contracts/Vesting.sol
7ad9a2d5b0c258e0b7feb32a07b6b49f584d8773f2330ffd0881c33f20f7bb72 ./contracts/Staking.sol
3fd9e081343cdbb8bef42f6a41fc1eac8dba131b632f453956dc28c5287e0a0d ./contracts/interfaces/IWhitelist.sol
082e7e73e460dca4ae0ce3318a5b65eb2f701447bcce9e2793933be32faa3ac3 ./contracts/interfaces/IERC20Extended.sol
fc1410261fe77807a8745785ec103a0595afa757a530df195874bb3c5e436d38 ./contracts/interfaces/IVesting.sol
28eb042b6c4029c6cd2c7c0358be40a5e1f5428171ec3dc0dbdee76e4cd4fb3d ./contracts/libraries/AddressPagination.sol
```

Tests

```
d613da4dc346a24afca354ade8c3e28cfc6661b716feb01fc3e3b2c6b4d46264 ./test/firestarterPresale.test.ts
ed82c51cf29ccaab36db9a14ef5fdc23d9580a8c208bb8f18fe0c9168fcc2aea ./test/staking.test.ts
6054ed713ef4eed6de69b24fc3fbd5600f8ab27e9c99092079d5f4eb911db0cc ./test/vesting.test.ts
6b9b1b3175a0bc2175f190baafb961d7aa2c14292221e41760fb2bcfac9fa2d8 ./test/locking.test.ts
83fbce99c05cbc6e435e28b0e92b6b5a211c6720a34f1bb470c88e869c7b9a7a ./test/projectPresale.test.ts
3e8210c4f7ce4b6b39274ae5cde281d05f9015108cdfe16b3c680c1497321064 ./test/presale.test.ts
cb91707378e0c60bebf91f24d7b5a14f9d3f0fba532aea80b35f3831cf4ffcb4 ./test/whitelist.test.ts
```

Changelog

- 2021-09-11 - Initial 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.