Quantstamp

Subscription Token - Fabric

Executive Summary

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

Type	Subscription protocol
Timeline	2023-10-09 through 2023-10-16
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Public Documentation
Source Code	• https://github.com/withfabricxyz/contracts #504dd06
Auditors	• Guillermo Escobero Auditing Engineer • Danny Aksenov Senior Auditing Engineer • Mostafa Yassin Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	9 Fixed: 5 Acknowledged: 4
High severity findings	1 Fixed: 1
Medium severity findings	4 Fixed: 2 Acknowledged: 2
Low severity findings	1 Fixed: 1
Undetermined severity findings	1 Acknowledged: 1
Informational findings	2 Fixed: 1 Acknowledged: 1

Summary of Findings

Quantstamp audited Subscription Token smart contract from Fabric. Subscription tokens give creators the ability to offer subscriptions to their fans on-chain. The protocol handles payments to the creator in exchange for an NFT with an expiration. Over time, subscribers add time to their subscription for continued access.

All issues and recommendations are discussed in the *Findings* section of this document. After that, recommendations about documentation and best practices are discussed. We strongly recommend addressing all the issues before deployment.

A high-security issue was found. SUB-1 explains how the reward redemption formula may inaccurately distribute rewards when total reward points decrease, leading to locking funds in the contract.

The documentation quality is high. Public documentation was provided by the Fabric team. It is recommended to add detailed and updated public documentation focusing on critical parts of the protocol, such as protocol fees, rewards, privileged accounts, and a list of addresses of the smart contracts deployed. Refer to the "Adherence to Specification" section for more details.

Regarding testing, all tests passed, and the project implements code coverage metrics (reaching `100%` in the audited contract). We recommend adding new tests to cover the proposed fixes (especially the fixes for SUB-1).

Fix review

The Fabric team provided a set of commits containing fixes for the issues found. All the issues were fixed or acknowledged.

ID	DESCRIPTION	SEVERITY	STATUS
SUB-1	**Funds Allocated for Rewards Can Be Locked in the Contract**	High	Fixed
SUB-2	**Users Can Get Discounted Subscription**	Medium	Acknowledged
SUB-3	**Checks-Effects-Interactions Pattern Violation**	Medium	Fixed
SUB-4	**Privileged Roles and Ownership**	Medium	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
SUB-5	**Missing Input Validation**	● Low ⓘ	Fixed
SUB-6	**Funds Sent Can Bypass the `receive()` Function**	● Informational ⓘ	Fixed
SUB-7	**Cannot Transfer to Other Accounts with Expired Subscriptions**	● Informational ⓘ	Acknowledged
SUB-8	**Fee Recipient Cannot Be Added if Not Initially Set**	● Undetermined ⓘ	Acknowledged
SUB-9	**Loss of Private Keys Leads to Rewards and Fees Allocation Lock**	● Medium ⓘ	Fixed

Assessment Breakdown

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

Disclaimer
Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

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

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. 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:
   1. 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.
   2. 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, clarity, 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.

Scope

Files Included

Files:

- `src/subscriptions/SubscriptionTokenV1.sol`
- `test/subscriptions/*`

Findings

SUB-1 Funds Allocated for Rewards Can Be Locked in the Contract

● High ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `9cce282fd2f97c3974fe88d4b54e93d600ec558c`, `a4f49058e07b31a9ebec6b5fda1adb992fd357f5`, and `bb507d0f553939b205422a464b7ea4b019b5c44b`.

Alert

The auditing team found a rounding issue in the fix proposed by the Fabric team. Due to integer division, it is possible to have a slashed value of zero reward tokens and zero tokens to be slashed from the already withdrawn variable:

```
uint256 slashed = (sub.rewardPoints * bps) / _MAX_BIPS; <=== rewardPoints can be greater than 0
but if bps is not big enough, slashed will be 0.

uint256 slashedValue = (sub.rewardsWithdrawn * bps) / _MAX_BIPS; <=== another instance where
`rewardsWithdrawn` might also be a positive value but not large enough, this can result in a 0
value.
```

This can result in several undesirable outcomes:

1. `slashed` is `0`, but if `slashedValue` ends up being greater than `0`, the result will be the updating of `_rewardPoolSlashed` and `sub.rewardsWithdrawn`, without affecting `_totalRewardPoints` or `sub.rewardPoints`. This will result in the incorrect distribution of rewards.
2. `slashed` is greater than `0`, but `slashedValue` ends up being `0` leading to the complete opposite effect. This will have the effect of updating the points but not the slashed value accordingly.

Update

Marked as "Fixed" by the client. Addressed in: `ffff4a4176a7a54842d6bce0acc6f89798dd5ccf` and `b0265087d099278be7e029d73792b7ca318ae65f`.

File(s) affected: `SubscriptionTokenV1.sol`

Description: A security issue has been identified within the reward redemption and total reward points management system of the platform. This issue is critical because it can result in unfair reward distribution, where users who have accumulated reward points may not receive the correct amount of tokens from the reward pool. The issue arises from the formula used for redemption and the dynamics of total reward points.

Analysis:

1. Reward Redemption Formula:
    - The platform uses a redemption formula to calculate the number of tokens a user receives from the reward pool when redeeming their reward points.
    - The formula is: `rewardPoolTotal * (userRewardPoints / totalRewardPoints)`.
    - `rewardPoolTotal` represents the total amount of tokens allocated for rewards (not the current reward pool balance).
    - `totalRewardPoints` is the sum of reward points in circulation.
2. Total Reward Points Dynamics:
    - `totalRewardPoints` increases when users purchase subscription time.
    - `totalRewardPoints` decreases when a user's subscription expires, and their reward points are slashed by other users.
    - The formula for reward points redemption can potentially fail to accurately distribute rewards when `totalRewardPoints` decreases, leading to funds blocked in the contract.

When a user subscription expires, other users can slash and burn those reward points, so `totalRewardPoints` will decrease and users will get a higher share per reward token on the reward pool. `rewardPoolBalance` will not have enough funds for the users, and some withdrawals will revert, locking the funds in the contract.

Exploit Scenario: In a scenario involving three users (A, B, C) with varying reward points and a reward pool allocation:

1. `totalRewardPoints` initially amounts to `600` (A: 100 + B: 200 + C: 300).
2. `rewardPoolTotal` is set at `$100,000`.
3. User `C` withdraws rewards: the reward pool balance is now `$50,000`.
4. User `C` subscription expires, and user `A` slashes `C`'s reward points, which results in a `totalRewardPoints` decrease to `300`.
5. Subsequently, when user `B` attempts to withdraw rewards, the redemption formula may fail, as it doesn't correctly account for the reduced `totalRewardPoints`: B tries to withdraw `$66,666.66`, but the `rewardPoolBalance` is `$50,000`, always reverting the transaction.

Here is a PoC:

Add the following setup functions to the contract:

```
    function setSub(address account, uint256 rewardsPoints) external {
        Subscription storage sub = _subscriptions[account];
        sub.rewardPoints = rewardsPoints;
    }

    function setRewards(uint256 amount1, uint256 amount2, uint256 amount3) external {
        _rewardPoolTotal = amount1;
        _rewardPoolBalance = amount2;
        _totalRewardPoints = amount3;
    }

    function slash(address account, uint256 amount) external {
        Subscription storage sub = _subscriptions[account];
        sub.rewardPoints -= amount;
        _totalRewardPoints -= amount;
    }
```

Then add the following test:

```
 function testInvalidRewards() public {
        vm.prank(alice);
        stp.mintFor{value: 1e18}(alice, 1e18);
        stp.setSub(alice, 100);

        vm.prank(bob);
        stp.mintFor{value: 1e18}(bob, 1e18);
        stp.setSub(bob, 200);

        vm.prank(charlie);
        stp.mintFor{value: 1e18}(charlie, 1e18);
        stp.setSub(charlie, 300);

        stp.setRewards(100000, 100000, 600);

        vm.prank(charlie);
        stp.withdrawRewards();

        stp.slash(charlie, 300);

        vm.prank(bob);
        stp.withdrawRewards();
    }
```

The test will fail with an arithmetic underflow :

```
Running 1 test for test/subscriptions/SubscriptionTokenV1.t.sol:SubscriptionTokenV1Test
[FAIL. Reason: Arithmetic over/underflow] testInvalidRewards() (gas: 780038)
Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 19.61ms

Failing tests:
Encountered 1 failing test in test/subscriptions/SubscriptionTokenV1.t.sol:SubscriptionTokenV1Test
[FAIL. Reason: Arithmetic over/underflow] testInvalidRewards() (gas: 780038)
```

Recommendation: Carefully evaluate the redemption formula and assess its suitability in scenarios where `totalRewardPoints` may decrease. Consider alternative formulas or modifications that can accurately account for changes in `totalRewardPoints`.

SUB-2 Users Can Get Discounted Subscription

● Medium ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Having referral codes which anyone could use to earn rewards was desired. We knew this meant referrals could be used as discounts, and ultimately decided this is a feature not a bug, given that the creator still earns revenue. Clients may present

these as discounts, but this effect documented now.

File(s) affected: `SubscriptionTokenV1.sol`

Description: The function `mintWithReferralFor()` takes both the `referralCode` and the `referrer` as user input. This means the user is free to choose whatever value they would like for `referrer`, including their own address.

Take the following scenario as an example:
1. Alice provides Bob with her referral code.
2. Bob purchases a subscription for himself, but instead of specifying Alice as the `referrer`, he specifies himself.
3. Bob gets a discount on his purchase, while Alice does not receive any payout.

Recommendation: Consider using a data structure that associates `referrer` and `referralBps` with the `referralCode` as the mapping, while also removing the user's ability to input the `referrer`.

SUB-3 Checks-Effects-Interactions Pattern Violation

● Medium Fixed

Update
Marked as "Fixed" by the client. Addressed in: `88b955a75e0b3de2838dff83113a215d4406778e`.

File(s) affected: `SubscriptionTokenV1.sol`

Related Issue(s): SWC-107

Description: The Checks-Effects-Interactions coding pattern is meant to mitigate any chance of other contracts manipulating the state of the blockchain in unexpected and possibly malicious ways before control is returned to the original contract. As the name implied, only after checking whether appropriate conditions are met and acting internally on those conditions should any external calls to, or interactions with, other contracts be done.

The following instance(s) have been identified:

- `_refund()`

```
uint256 tokens = balance * _tokensPerSecond;

if (balance > 0) {
    sub.secondsPurchased -= balance;
    _transferOut(account, tokens);  <== External Call
}

 _subscriptions[account] = sub; <== State Change
```

- `_fetchSubscription()`

```
if (sub.tokenId == 0) {
    require(_supplyCap == 0 || _tokenCounter < _supplyCap, "Supply cap reached");
    _tokenCounter += 1;
    sub = Subscription(_tokenCounter, 0, 0, block.timestamp, block.timestamp, 0, 0, 0); <== This state
gets updated in `purchaseTime()`
    _safeMint(account, sub.tokenId); <== External Call
}
```

Recommendation: We recommend refactoring the code so that it conforms to the Checks-Effects-Interactions pattern.

SUB-4 Privileged Roles and Ownership

● Medium Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

As discussed with QS, this is more about documentation. We added a section to docs around Roles and what each role can do.

File(s) affected: `SubscriptionTokenV1.sol`

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

`Owner` can:
- set/update the transfer recipient
- set/update the supply cap
- pause/unpause the contract
- grant time to any accounts
- update contract metadata
- refund remaining purchase time to any account
- withdraw contract funds to any account
- create/delete referral codes

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

SUB-5 Missing Input Validation

● Low ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `345f1170758dd7cb54f3450392a9e2d9aac3e0e4`.

File(s) affected: `SubscriptionTokenV1.sol`

Related Issue(s): SWC-123

Description: It is essential to validate inputs, even if they only come from trusted addresses, to avoid human error.

The following missing input validations have been identified:
- `initialize()`
  - `params.name` : validate to be a non-empty string.
  - `params.symbol` : validate to be a non-empty string.
  - `params.contractUri` : validate to be a non-empty string.
  - `params.tokenUri` : validate to be a non-empty string.
- `mintWithReferralFor()`
  - `account` : validate to be a non-zero address.
  - `referrer` : validate to be a non-zero address.
- `withdrawTo()`
  - `account` : validate to be a non-zero address.
- `refund()`
  - `accounts` : validate to be a non-empty list.
- `updateMetadata()`
  - `_contractURI` : validate to be a non-empty string.
  - `_tokenURI` : validate to be a non-empty string.
- `_grantTime()`
  - `numSeconds` : validate to be a non-zero value.
- `createReferralCode()` relies on verifying that `_referralCodes[code].bps` is zero to know if a referral code exists or not. However, it does not revert if `bps` input parameter is zero. This can confuse external systems listening to `ReferralCreated` event. Validate `bps` to be a non-zero value.

Recommendation: We recommend adding the relevant checks.

SUB-6 Funds Sent Can Bypass the `receive()` Function

● Informational ⓘ Fixed

Update
Functions `recoverNativeTokens()` and `reconcileNativeBalance()` were implemented to withdraw or account for unexpected native tokens.

Update

Marked as "Fixed" by the client. Addressed in: `9597b5f1ed81ae5722c84c01d8c0e5a3d9deb7ff` and `6717d6985f96e79f6aae6f1f44ec4f509e3371e5`. The client provided the following explanation:

> Added some functions to recover/reconcile native tokens.

File(s) affected: `SubscriptionTokenV1.sol`

Description: Funds can be sent to the contract using the function `selfDestruct`, funds sent this way will not invoke the code inside the `receive()` function.

Recommendation: The only issue here is that funds received this way will not be accounted for in the contract. It will also not be possible to withdraw them.

SUB-7 Cannot Transfer to Other Accounts with Expired Subscriptions

● Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> This is by design. There can only be one token per account. The token itself can be renewed. Transfers are intended for end users to move their asset to another account of theirs.

File(s) affected: `SubscriptionTokenV1.sol`

Description: The hook `_beforeTokenTransfer()` checks if the `to` address has a subscription, whether it is active or not. If that is the case, it prevents the transfer.

This will prevent transferring to users who have expired subscriptions.

Recommendation: Confirm that this is intended, if not, then consider changing the logic to check if the `to` address has an active subscription.

SUB-8 Fee Recipient Cannot Be Added if Not Initially Set

● Undetermined ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> This is by design. Enabling fees should be transparent at time of deployment. It seems to most fair. We do however support disabling fees (if the fee collector so chooses) to relinquish the right to fees. Reasons for that could be usage which does not align with the fee collectors ideals.

File(s) affected: `SubscriptionTokenV1.sol`

Description: To update the `_feeCollector`, the call to `updateFeeRecipient()` must come from the current `_feeCollector`. If the `_feeCollector` was not initially set to a non zero address during initialization, then the `_feeCollector` can't be updated from the duration of the contract's life cycle.

Recommendation: Determine whether this behavior aligns with the team's business logic and if not make the appropriate changes.

SUB-9 Loss of Private Keys Leads to Rewards and Fees Allocation Lock

● Medium ⓘ Fixed

Update
Fees and rewards are being allocated at purchase time via `_allocateFeesAndRewards()`.

Update
Marked as "Fixed" by the client. Addressed in: `9cce282fd2f97c3974fe88d4b54e93d600ec558c`.

File(s) affected: `SubscriptionTokenV1.sol`

Description: A scenario where a content creator has lost access to their private keys, and did not set a designated transfer recipient (`_transferRecipient`) will lead to funds locked in the contract. This oversight results in rewards (if activated) and fees not being correctly allocated, causing these funds to remain locked within the contract indefinitely.

Rewards and fees are allocated when `withdrawTo()` is executed. Note that this function can only be called by the contract creator. Users will not be able to claim any reward tokens by redeeming their reward points.

Although this can be seen as a contract creator's responsibility, final user funds will be locked, potentially harming the protocol's reputation.

Recommendation: Consider modifying the design of fees and reward allocation. One possible approach would be allocating them at minting time (users purchasing subscription time).

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – 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.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **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).

Code Documentation

1. Fixed `refund()` documentation should cover that `refund()` can be used to clear grants (e.g. passing zero as `numTokensIn`).

Adherence to Best Practices

1. Each repository's `README.md` file should contain basic instructions on how to run the test suite and any prerequisites for running tests. For each of these prerequisites also specify the versions supported/tested.

Adherence to Specification

1. Fixed Documentation states that 30% of slashed reward points will be transferred to the slasher. This is not found in the code.
2. Fixed Documentation should mention that rewards and fees will be allocated once the owner (creator) withdraws the earnings.

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

- `abb...ec1 ./src/subscriptions/SubscriptionTokenV1.sol`

Tests

- `916...683 ./test/subscriptions/SubscriptionTokenV1Grants.sol`

- `b31...6f6 ./test/subscriptions/SubscriptionTokenV1.t.sol`
- `422...2d8 ./test/subscriptions/SubscriptionTokenV1Rewards.t.sol`
- `948...a6e ./test/subscriptions/SubscriptionTokenV1Referral.t.sol`
- `da2...913 ./test/subscriptions/SubscriptionTokenV1Factory.t.sol`
- `670...3d1 ./test/subscriptions/BaseTest.t.sol`
- `bb6...3b6 ./test/subscriptions/SubscriptionTokenV1Fee.t.sol`

Toolset

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

Setup

Tool Setup:
- Slither v0.9.3
- SuMo v2.4.0

Steps taken to run the tools:
1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Slither was used to get a static analysis of the repository. All the issues and recommendations are discussed in this report or classified as false positives.

SuMo

SuMo is a mutation testing tool for Solidity Smart Contracts. It features 25 Solidity-specific mutation operators and 19 traditional operators.

`SubscriptionTokenV1` contract obtained a score of 75.03%. We shared the detailed mutants with the Fabric team to improve the test suite further and recommended reaching a 100% score. This will help to cover bugs in future changes.

Details:

```
Mutation Testing completed in 89.40 minutes 👋

SuMo generated 1007 mutants:
- 244 live;
- 733 killed;
- 30 stillborn;
- 0 equivalent;
- 0 redundant;
- 0 timed-out.

Mutation Score: 75.03 %
```

Test Suite Results

All tests passed.

```
Running 9 tests for test/finance/CrowdFinancingV1Factory.t.sol:CrowdFinancingV1FactoryTest
[PASS] testDeployFeeCapture() (gas: 303875)
[PASS] testDeployFeeCollectNone() (gas: 12879)
[PASS] testDeployFeeTooLow() (gas: 37554)
[PASS] testDeployFeeTransfer() (gas: 323016)
[PASS] testDeployFeeTransferBadReceiver() (gas: 316101)
[PASS] testDeployFeeTransferNonOwner() (gas: 310214)
[PASS] testDeployFeeUpdate() (gas: 42580)
[PASS] testDeployment() (gas: 292136)
[PASS] testFeeUpdate() (gas: 44465)
```

```
Test result: ok. 9 passed; 0 failed; finished in 6.92ms

Running 3 tests for test/finance/DataQuiltRegistryV1.t.sol:DataQuiltRegistryV1Test
[PASS] testInvalidAddress() (gas: 15937)
[PASS] testMintWithContribution() (gas: 2686955)
[PASS] testMintWithoutContribution() (gas: 2540623)
Test result: ok. 3 passed; 0 failed; finished in 6.93ms

Running 18 tests for test/finance/CrowdFinancingV1/InitTests.t.sol:InitTests
[PASS] testBadRange() (gas: 36767)
[PASS] testBadrecipient() (gas: 36470)
[PASS] testFeeCollectorNoFees() (gas: 39371)
[PASS] testImpossibleDepositRange() (gas: 39089)
[PASS] testImpossibleGoalRange() (gas: 38974)
[PASS] testInitialDeployment() (gas: 255674)
[PASS] testLargePayoutFee() (gas: 39210)
[PASS] testLargeUpfrontFee() (gas: 39178)
[PASS] testMinDepositGoalRelation() (gas: 39212)
[PASS] testMinGoalRelationWith1() (gas: 226330)
[PASS] testNoFeeCollectorFees() (gas: 41508)
[PASS] testPastStart() (gas: 39354)
[PASS] testReinit() (gas: 231884)
[PASS] testTooLong() (gas: 36902)
[PASS] testValid() (gas: 226316)
[PASS] testZeroDeposit() (gas: 38997)
[PASS] testZeroGoal() (gas: 38938)
[PASS] testtransferFeeBips() (gas: 823096)
Test result: ok. 18 passed; 0 failed; finished in 7.03ms

Running 5 tests for test/subscriptions/SubscriptionTokenV1Grants.sol:SubscriptionTokenV1GrantsTest
[PASS] testGrant() (gas: 202012)
[PASS] testGrantDouble() (gas: 202302)
[PASS] testGrantMixed() (gas: 316484)
[PASS] testGrantRefund() (gas: 192578)
[PASS] testGrantRefundMixed() (gas: 316096)
Test result: ok. 5 passed; 0 failed; finished in 6.87ms

Running 17 tests for test/subscriptions/SubscriptionTokenV1Rewards.t.sol:SubscriptionTokenV1RewardsTest
[PASS] testDecay() (gas: 43530)
[PASS] testDisabledWithdraw() (gas: 5158575)
[PASS] testDoubleSlash() (gas: 542465)
[PASS] testNoPointAllocation() (gas: 279737)
[PASS] testRewardPointAllocation() (gas: 349250)
[PASS] testRewardPointWithdraw() (gas: 819197)
[PASS] testRewardPointWithdrawStepped() (gas: 1016999)
[PASS] testRewardPoolToCreator() (gas: 486159)
[PASS] testSingleHalving() (gas: 4881215)
[PASS] testSlashPostWithdraw() (gas: 748784)
[PASS] testSlashPostWithdrawDistanceFuture() (gas: 748561)
[PASS] testSlashResub() (gas: 623472)
[PASS] testSlashing() (gas: 507322)
[PASS] testSlashingActive() (gas: 515345)
[PASS] testSlashingNoRewards() (gas: 497937)
[PASS] testSlashingWithdraws() (gas: 571620)
[PASS] testWithdrawExpired() (gas: 366321)
Test result: ok. 17 passed; 0 failed; finished in 10.00ms

Running 4 tests for test/finance/CrowdFinancingV1/YieldTests.t.sol:YieldTests
[PASS] testMulti() (gas: 6447686)
[PASS] testProfit() (gas: 6279707)
[PASS] testReturns() (gas: 6384613)
[PASS] testYieldFees() (gas: 6520860)
Test result: ok. 4 passed; 0 failed; finished in 10.30ms

Running 8 tests for test/finance/CrowdFinancingV1/WithdrawTests.t.sol:WithdrawTests
[PASS] testDoubleWithdraw() (gas: 5738205)
[PASS] testDoubleWithdrawYieldBalance() (gas: 6429961)
[PASS] testEarlyWithdraw() (gas: 5806627)
[PASS] testGoalNotMet() (gas: 5757233)
[PASS] testWithdrawContributeERC20Fail() (gas: 6444543)
```

```
[PASS] testWithdrawContributionEthFail() (gas: 2639615)
[PASS] testWithdrawYieldBalanceERC20Fail() (gas: 6720718)
[PASS] testWithdrawYieldEthFail() (gas: 2845621)
Test result: ok. 8 passed; 0 failed; finished in 13.02ms

Running 12 tests for test/finance/CrowdFinancingV1/TransferTests.t.sol:TransferTests
[PASS] testAllFees() (gas: 6330705)
[PASS] testBadOutcome() (gas: 5941271)
[PASS] testEarlySuccess() (gas: 6341323)
[PASS] testProcessFailedFeeTransfer() (gas: 6668294)
[PASS] testProcessFailedTransfer() (gas: 5784356)
[PASS] testReprocess() (gas: 6165132)
[PASS] testSuccess() (gas: 6176388)
[PASS] testTooEarly() (gas: 5623054)
[PASS] testTransferEthFailedFeeTransfer() (gas: 5236834)
[PASS] testTransferFailedTransfer() (gas: 6602720)
[PASS] testUpfrontFees() (gas: 6285542)
[PASS] testYieldFees() (gas: 6267716)
Test result: ok. 12 passed; 0 failed; finished in 12.98ms

Running 7 tests for test/subscriptions/SubscriptionTokenV1Referral.t.sol:SubscriptionTokenV1ReferralTest
[PASS] testCreate() (gas: 41828)
[PASS] testCreateInvalid() (gas: 42482)
[PASS] testDelete() (gas: 29559)
[PASS] testInvalidReferralCode() (gas: 281372)
[PASS] testRewards() (gas: 382034)
[PASS] testRewardsErc20() (gas: 5838529)
[PASS] testRewardsMintFor() (gas: 286196)
Test result: ok. 7 passed; 0 failed; finished in 1.62ms

Running 7 tests for test/subscriptions/SubscriptionTokenV1Fee.t.sol:SubscriptionTokenV1FeeTest
[PASS] testAllocation() (gas: 5214421)
[PASS] testFeeCollectorRelinquish() (gas: 5178753)
[PASS] testFeeCollectorUpdate() (gas: 4909266)
[PASS] testFeeTransfer() (gas: 5207695)
[PASS] testRenounce() (gas: 5238269)
[PASS] testTransferAll() (gas: 5410632)
[PASS] testWithdrawWithFees() (gas: 5389958)
Test result: ok. 7 passed; 0 failed; finished in 2.13ms

Running 5 tests for test/finance/CrowdFinancingV1/UnlockTests.t.sol:UnlockTests
[PASS] testEvents() (gas: 6121990)
[PASS] testInProgress() (gas: 5625481)
[PASS] testPostFail() (gas: 5873976)
[PASS] testPostSuccess() (gas: 6166340)
[PASS] testSuccess() (gas: 6141001)
Test result: ok. 5 passed; 0 failed; finished in 21.48ms

Running 20 tests for test/finance/CrowdFinancingV1/ContributionTests.t.sol:ContributionTests
[PASS] testBadAllowance() (gas: 3128531)
[PASS] testBadOutcome() (gas: 2617611)
[PASS] testBigThenSmallContribution() (gas: 5845101)
[PASS] testContributionMaxOnFeeTokens() (gas: 6374929)
[PASS] testContributionMinOnFeeTokens() (gas: 6316861)
[PASS] testDepositEmit() (gas: 2594091)
[PASS] testDepositRange() (gas: 2608165)
[PASS] testDepositRangeAdvanced() (gas: 2685638)
[PASS] testEarlyGoal() (gas: 2802740)
[PASS] testEarlycontributeERC20() (gas: 3196771)
[PASS] testEarlycontributeEth() (gas: 2545396)
[PASS] testFailBadBalance() (gas: 3218571)
[PASS] testHappyPath() (gas: 5804178)
[PASS] testHugeDeposit() (gas: 2543787)
[PASS] testInvalidERC20Deposit() (gas: 2535315)
[PASS] testInvalidETHDeposit() (gas: 3160409)
[PASS] testLateDeposit() (gas: 2540997)
[PASS] testSmallContribution() (gas: 2543708)
[PASS] testTransferFalseReturn() (gas: 6294294)
[PASS] testTransferredCampaign() (gas: 2729932)
Test result: ok. 20 passed; 0 failed; finished in 31.52ms
```

```
Running 13 tests for test/subscriptions/SubscriptionTokenV1Factory.t.sol:SubscriptionTokenV1FactoryTest
[PASS] testDeployFeeCapture() (gas: 376942)
[PASS] testDeployFeeCollectNone() (gas: 12890)
[PASS] testDeployFeeTooLow() (gas: 36974)
[PASS] testDeployFeeTransfer() (gas: 396193)
[PASS] testDeployFeeTransferBadReceiver() (gas: 389146)
[PASS] testDeployFeeTransferNonOwner() (gas: 383299)
[PASS] testDeployFeeUpdate() (gas: 42573)
[PASS] testDeployment() (gas: 385542)
[PASS] testDeploymentWithReferral() (gas: 402006)
[PASS] testFeeCreate() (gas: 43173)
[PASS] testFeeCreateInvalid() (gas: 52324)
[PASS] testFeeDestroy() (gas: 28655)
[PASS] testInvalidReferral() (gas: 404303)
Test result: ok. 13 passed; 0 failed; finished in 14.61ms

Running 8 tests for test/finance/CrowdFinancingV1/TokenizationTests.t.sol:TokenizationTests
[PASS] testIncreaseDecreaseAllowance() (gas: 2737772)
[PASS] testInsufficientAllowance() (gas: 2750872)
[PASS] testInvalidAllowanceAddresses() (gas: 2725218)
[PASS] testInvalidTransferAddresses() (gas: 2725394)
[PASS] testPayoutRecalculation() (gas: 2886306)
[PASS] testTokenAllowanceAndApproval() (gas: 2733280)
[PASS] testTokenTransfer() (gas: 2758933)
[PASS] testtransferFeeBipss() (gas: 2725073)
Test result: ok. 8 passed; 0 failed; finished in 13.68ms

Running 39 tests for test/subscriptions/SubscriptionTokenV1.t.sol:SubscriptionTokenV1Test
[PASS] testCreatorEarnings() (gas: 561527)
[PASS] testERC20FeeTakingToken() (gas: 5722543)
[PASS] testERC20Mint() (gas: 5790141)
[PASS] testInit() (gas: 112032)
[PASS] testInvalidRefund() (gas: 246403)
[PASS] testMint() (gas: 245781)
[PASS] testMintDecay() (gas: 234352)
[PASS] testMintExpire() (gas: 236003)
[PASS] testMintFor() (gas: 299852)
[PASS] testMintForErc20() (gas: 5806341)
[PASS] testMintInvalid() (gas: 29747)
[PASS] testMintInvalidERC20() (gas: 5556468)
[PASS] testMintSpaced() (gas: 257741)
[PASS] testMintViaFallback() (gas: 228641)
[PASS] testMintViaFallbackERC20() (gas: 5549721)
[PASS] testNonSub() (gas: 31838)
[PASS] testPartialRefund() (gas: 287041)
[PASS] testPausing() (gas: 251596)
[PASS] testReconcile() (gas: 5608808)
[PASS] testReconcileEth() (gas: 15632)
[PASS] testReconcileNative() (gas: 129167)
[PASS] testRecoverERC20() (gas: 603014)
[PASS] testRecoverERC20Self() (gas: 5544268)
[PASS] testRecoverNative() (gas: 5644769)
[PASS] testRefund() (gas: 271934)
[PASS] testRefundCalc() (gas: 404675)
[PASS] testRefundERC20() (gas: 5787092)
[PASS] testRefundERC20AfterWithdraw() (gas: 5800582)
[PASS] testRefundNoBalance() (gas: 298232)
[PASS] testRefundNoPurchase() (gas: 267936)
[PASS] testRenounce() (gas: 328317)
[PASS] testSupplyCap() (gas: 434535)
[PASS] testTransfer() (gas: 311652)
[PASS] testTransferAll() (gas: 471110)
[PASS] testTransferEarnings() (gas: 322760)
[PASS] testTransferToExistingHolder() (gas: 424932)
[PASS] testUpdateMetadata() (gas: 272320)
[PASS] testWithdraw() (gas: 443478)
[PASS] testWithdrawERC20() (gas: 5977436)
Test result: ok. 39 passed; 0 failed; finished in 14.15ms
```

Code Coverage

The test suite shows good coverage metrics, reaching 100% in the audited contract.

File	% Lines	% Statements	% Branches	% Funcs
src/finance/CrowdFinancingV1.sol	100.00% (**181/**181)	100.00% (**193/**193)	98.94% (**93/**94)	100.00% (**53/**53)
src/finance/CrowdFinancingV1Factory.sol	100.00% (**18/**18)	100.00% (**22/**22)	100.00% (**4/**4)	100.00% (**5/**5)
src/finance/DataQuiltRegistryV1.sol	100.00% (**9/**9)	100.00% (**9/**9)	100.00% (**4/**4)	100.00% (**3/**3)
src/subscriptions/SubscriptionTokenV1.sol	100.00% (**287/**287)	100.00% (**325/**325)	100.00% (**156/**156)	100.00% (**71/**71)
src/subscriptions/SubscriptionTokenV1Factory.sol	100.00% (**25/**25)	100.00% (**27/**27)	100.00% (**16/**16)	100.00% (**6/**6)
test/finance/CrowdFinancingV1/mocks/MockFeeToken.sol	100.00% (**4/**4)	100.00% (**4/**4)	100.00% (**0/**0)	100.00% (**2/**2)
test/finance/CrowdFinancingV1/mocks/MockToken.sol	100.00% (**5/**5)	100.00% (**5/**5)	100.00% (**0/**0)	100.00% (**3/**3)
test/mocks/SelfDestruct.sol	100.00% (**1/**1)	100.00% (**1/**1)	100.00% (**0/**0)	100.00% (**1/**1)
Total	100.00% (**530/**530)	100.00% (**586/**586)	99.64% (**273/**274)	100.00% (**144/**144)

Changelog

- 2023-10-16 - Initial report
- 2023-10-25 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

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

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

Timeliness of content

Quantstamp