Quantstamp

Sequence - Clawback

Executive Summary

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

Type	Token Wrapper
Timeline	2024-07-01 through 2024-07-05
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Internal technical specifications
Source Code	• 0xsequence/contracts-library #5fde93d
Auditors	• Nikita Belenkov Auditing Engineer • Rabib Islam Senior Auditing Engineer • Pavel Shabarkin Senior Auditing Engineer

Documentation quality	High
Test quality	Medium
Total Findings	2 Fixed: 2
High severity findings	0
Medium severity findings	1 Fixed: 1
Low severity findings	1 Fixed: 1
Undetermined severity findings	0
Informational findings	0

Summary of Findings

Sequence has developed this clawback contract that will work alongside any ERC20, ERC720 or ERC1155 contract to wrap the tokens with additional logic that governs when the underlying token can be transferred. This model will allow minters to block any transfers until a fiat purchase clears, so in the case of a chargeback, the token can be "clawed back."

During the review of this project, we found two issues: a low-severity one that allows wrapping an already wrapped token and a medium-severity one that makes the clawback process more expensive for the operator via multiple transfers. Some other concerns are also highlighted in the operational considerations and should also be taken into consideration.

The project has 43 tests, of which all pass with a branch coverage of 77.80%. The branch coverage can be further improved to ideally above 90%.

Fix Review Update

All issues have been fixed. The test suite has been updated to cover the added fixes.

ID	DESCRIPTION	SEVERITY	STATUS
SEQ-1	**Transferring Wrapped Tokens to Multiple Addresses Can Prevent Clawbacks**	● Medium	Fixed
SEQ-2	**Clawback ERC-1155 Allows Nested Wrappings**	● Low	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

Repo: https://github.com/0xsequence/contracts-library/(5fde93d731f28fcc2239290aedddc5cfc674c45e) Files: src/tokens/wrappers/clawback/*

Operational Considerations

1. There is no official support for non-standard tokens such as fee-on-transfer or rebasing tokens. This leads to unexpected behavior and potential loss of funds if such tokens are used.
2. It is required to approve the contract an allowance in order to execute certain operations. In general, approvals should be as small and as short as possible in order to limit damage in the case of exploitation of funds.
3. Updating a template will cause changes to the behavior of every wrapped token using that template. Notably, the template admin thereby has the ability to immediately unlock the underlying tokens of every wrapped token holder by decreasing the template's duration to zero. Therefore, wrapped token minters should make sure that they trust the admins of the templates they use to mint wrapped tokens.
4. While template operators can be both added and removed, template transferers can only be added, not removed.

Key Actors And Their Capabilities

The contract inherits from OpenZeppelin's `Ownable` contract. As such, it has an `owner` address with privileged access to certain functions. Also, each template possesses an admin with privileged access.

1. `owner`
   1. `updateMetadataProvider()`
   2. `transferOwnership()`
   3. `renounceOwnership()`
2. `_templates[templateId].admin`
   1. `updateTemplate()`
   2. `updateTemplateAdmin()`
   3. `addTemplateTransferer()`
   4. `updateTemplateOperator()`
3. `templateTransferers[][]`

   1. If the wrapped token's template has the `transferOpen` field set to `false`, then in order to transfer the wrapped token, one must be a designated address in the `templateTransferers` mapping.
4. `templateOperators[][]`
   1. Template operators are allowed to both clawback underlying tokens before the lock duration of the wrapped tokens has expired and unwrap tokens to the wrapped token owners once the lock duration has expired.

Findings

SEQ-1
Transferring Wrapped Tokens to Multiple Addresses Can Prevent Clawbacks

● Medium ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `bfae6bbcc2668ac224d0a0262f1cc20468c01f2c`.

File(s) affected: `src/tokens/wrappers/clawback/Clawback.sol`

Description: Once wrapped tokens are sent to a recipient, this recipient may transfer these tokens to whomever they wish. In particular, they may send these tokens to as many addresses as there are tokens. However, this can make clawing back the underlying tokens extremely impractical, as the corresponding wrapped tokens *have* to be burned for the completion of the clawback, and each specific wrapped token owner has to be identified when calling `clawback()`. Meanwhile, the recipient can unwrap the tokens once the lock duration has expired. Even if the tokens are not ultimately unwrapped by the recipient, the underlying tokens will be deprived from who could have been the owner after clawing back.

Exploit Scenario:
1. The project owner wraps 1000000 tokens and sends the wrapped tokens to a user.
2. The user sends the 1000000 wrapped tokens to a contract that sends 1000 each to a different address.
3. The project owner decides not to call `clawback()` 1000 times in order to save gas.
4. Once the lock duration expires, the user transfers all the wrapped tokens to one address and calls `unwrap()` in order to retrieve the underlying.

Recommendation: Create an emergency clawback function that claws back all the underlying tokens without having to burn any. Although this will impact all wrapped token holders, a new wrapped token may be prepared for users whose tokens were not intended to be clawed back.

SEQ-2 Clawback ERC-1155 Allows Nested Wrappings

● Low ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `8494e125b3b75630fa8b1e973ab77a1625e13a76`.

File(s) affected: `src/tokens/wrappers/clawback/Clawback.sol`

Description: The `Clawback` contract inherits the ERC1155 standard and accepts three types of tokens: ERC20, ERC721, and ERC1155. It mints wrapped versions of these underlying tokens. Since the `Clawback` contract inherits the ERC1155 standard, it is possible to wrap an already wrapped token. Although we have not identified any direct exploitation of this side effect, the protocol may still be abused in the following ways:

- If the `Clawback` contract address is set in the `templateTransferers[details.templateId][address(clawback)]` mapping (there is no practical use-case for this), an attacker could bypass transfer restrictions by wrapping the wrapped token, transferring it to the desired user, who can then unwrap it.
- When a user unwraps a wrapped-wrapped token, the wrapped token is held by the `Clawback` smart contract address. If the operator seizes funds via the `clawback()` function (which should seize from the `Clawback` contract itself), the contract balance of the wrapped token would decrease (effectively losing its "balance"). Thus, during the unwrapping of the wrapped-wrapped token, the contract will not have enough "balance" to complete the transfer in the `_transferFromOther()` function, resulting in a revert unless someone sends their wrapped token to the `Clawback` contract or decides to wrap the wrapped token in the same way.

These two cases do not pose a direct threat or vulnerability to the contract but certainly violate the implied business logic. This might lead to direct security issues in the future due to contract updates, ecosystem upgrades, or unknown vulnerabilities.

Recommendation: Remove the ability to wrap wrapped tokens to explicitly define the protocol's business logic and eliminate any potential violations.

Auditor Suggestions

SEQ-S3 Gas Optimization

Fixed

Update

Marked as "Fixed" by the client.

Addressed in: `784237e41014ed981de79be18b56e8e06ac32d41`.

Description: There are a few places where gas optimization is possible:

1. The following functions can be marked `external` for lower gas costs:
    1. `wrap()`
    2. `addToWrap()`
    3. `unwrap()`
    4. `clawback()`
    5. `addTemplate()`
    6. `updateTemplate()`
    7. `updateTemplateAdmin()`
    8. `addTemplateTransferer()`
    9. `updateTemplateOperator()`
    10. `updateMetadataProvider()`
    11. `uri()`
2. When looping through a `for` loop, use `unchecked { ++i }` to increment the index.
3. When looping through an array, cache the length of the array. Opportunities exist at `Clawback.sol#L245` and `ClawbackMetadata.sol#L45,186`.

Recommendation: Consider adding these gas optimizations to the codebase.

SEQ-S4 Ownership Can Be Renounced

Acknowledged

Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

```
Renouncing ownership is an intentional feature
```

File(s) affected: `src/tokens/wrappers/clawback/Clawback.sol`

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed.

Recommendation: Confirm that this is the intended behavior. If not, override and disable the `renounceOwnership()` function in the affected contracts. For extra security, consider using a two-step process when transferring the ownership of the contract (e.g. `Ownable2Step` from OpenZeppelin).

SEQ-S5 Unlocked Pragma

Acknowledged

Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

```
Unlocked pragma is intentional so that these contracts can be integrated more easily in downstream
contracts. Compiler version is fixed in the foundry.toml for consistent compilation in this repository.
```

File(s) affected: `src/tokens/wrappers/clawback/Clawback.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

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

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).

Automated Analysis

N/A

Test Suite Results

All of the 47 tests successfully pass.

Fix Review Update

7 tests have been added to cover the added functionality. The total test count now stands at 54 tests, of which all successfully pass.

```
Ran 5 tests for test/tokens/wrappers/clawback/ClawbackMetadata.t.sol:ClawbackMetadataTest
[PASS] testDurationAndUnlocksAt() (gas: 500421)
[PASS] testMetadataPropertiesERC1155((uint8,uint32,uint56,uint256),(bool,bool,uint56,address)) (runs:
1024, μ: 156222, ~: 145344)
[PASS] testMetadataPropertiesERC20((uint8,uint32,uint56,uint256),(bool,bool,uint56,address)) (runs: 1024,
μ: 161903, ~: 161918)
[PASS] testMetadataPropertiesERC721((uint8,uint32,uint56,uint256),(bool,bool,uint56,address)) (runs:
1024, μ: 259439, ~: 255304)
[PASS] testSupportsInterface() (gas: 9592)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 11.17s (7.75s CPU time)

Ran 42 tests for test/tokens/wrappers/clawback/Clawback.t.sol:ClawbackTest
[PASS] testAddTemplate(address,uint56,bool,bool) (runs: 1024, μ: 46866, ~: 46866)
[PASS] testAddTemplateTransferer(address,address) (runs: 1024, μ: 76988, ~: 76988)
[PASS] testAddTemplateTransfererInvalidCaller(address,address,bool,address) (runs: 1024, μ: 62486, ~:
49852)
[PASS] testAddToWrap(address,uint8,uint256,uint256,uint56,address,uint64) (runs: 1024, μ: 284052, ~:
260181)
[PASS] testAddToWrapInvalidWrappedId(uint256,uint256,address) (runs: 1024, μ: 13844, ~: 13844)
[PASS] testClawback(uint8,uint256,uint256,uint56,address,address) (runs: 1024, μ: 277780, ~: 299967)
[PASS] testClawbackAfterTransfer(uint8,uint256,uint256,uint56,address,address,address) (runs: 1024, μ:
301172, ~: 319223)
[PASS] testClawbackDestructionOnly(uint8,uint256,uint256,uint56,address) (runs: 1024, μ: 283205, ~:
308317)
[PASS] testClawbackDestructionOnlyInvalidReceiver(uint8,uint256,uint256,uint56,address,address) (runs:
1024, μ: 266990, ~: 292316)
[PASS] testClawbackInvalidCaller(uint8,uint256,uint256,uint56,address,address) (runs: 1024, μ: 231881, ~:
258551)
[PASS] testClawbackInvalidUnlocked(uint8,uint256,uint256,uint56,address,address) (runs: 1024, μ: 261503,
~: 286189)
[PASS] testPreventsOnERC1155Received(uint256,uint256) (runs: 1024, μ: 100123, ~: 100035)
[PASS] testPreventsOnERC1155Received(uint256,uint256,uint256,uint256) (runs: 1024, μ: 158859, ~: 158856)
[PASS] testPreventsOnERC721Received(uint256) (runs: 1024, μ: 118187, ~: 118187)
[PASS] testSupportsInterface() (gas: 13393)
[PASS] testTransferByTransfererAsFrom(uint8,uint256,uint256,uint56,uint64,bool,address,bool) (runs: 1024,
µ: 277540, ~: 302552)
[PASS] testTransferByTransfererAsOperator(uint8,uint256,uint256,uint56,uint64,bool,address,address,bool)
(runs: 1024, µ: 303230, ~: 326681)
[PASS] testTransferByTransfererAsTo(uint8,uint256,uint256,uint56,uint64,bool,address,bool) (runs: 1024,
µ: 278444, ~: 303081)
[PASS] testTransferByTransfererNotApproved(uint8,uint256,uint256,uint56,uint64,bool,address,address,bool)
(runs: 1024, µ: 270010, ~: 294609)
[PASS] testTransferInvalidOperator(uint8,uint256,uint256,uint56,address,address,bool) (runs: 1024, µ:
267418, ~: 293994)
[PASS] testTransferOpen(uint8,uint256,uint256,uint56,uint64,address) (runs: 1024, µ: 250478, ~: 274710)
[PASS] testUnwrap(address,uint8,uint256,uint256,uint56,uint64) (runs: 1024, µ: 235726, ~: 242779)
[PASS] testUnwrapAfterTransfer(address,uint8,uint256,uint256,uint56,uint64,address) (runs: 1024, µ:
271427, ~: 288198)
[PASS] testUnwrapByInvalidOperator(address,address,uint8,uint256,uint256,uint56) (runs: 1024, µ: 233046,
~: 258935)
[PASS] testUnwrapByOperator(address,address,uint8,uint256,uint256,uint56,uint64) (runs: 1024, µ: 268114,
~: 275759)
[PASS] testUnwrapInvalidAmount(address,uint8,uint256,uint256,uint256,uint56) (runs: 1024, µ: 230856, ~:
257858)
[PASS] testUnwrapInvalidToken(address,uint8,uint256,uint256,uint256,uint56) (runs: 1024, µ: 237642, ~:
263222)
[PASS] testUnwrapTokenLocked(address,uint8,uint256,uint256,uint56) (runs: 1024, µ: 227853, ~: 255248)
[PASS] testUpdateTemplateAdmin(address,address,uint56,bool,bool) (runs: 1024, µ: 75995, ~: 75995)
[PASS] testUpdateTemplateAdminInvalidAdmin() (gas: 49646)
[PASS] testUpdateTemplateAdminInvalidCaller(address,address,bool) (runs: 1024, µ: 63809, ~: 75966)
[PASS] testUpdateTemplateInvalidCaller(address,address,bool,uint56,bool,bool) (runs: 1024, µ: 63321, ~:
75803)
[PASS] testUpdateTemplateInvalidDestructionOnly() (gas: 49683)
[PASS] testUpdateTemplateInvalidDuration(address,uint56,bool,bool,uint56) (runs: 1024, µ: 51191, ~:
51191)
[PASS] testUpdateTemplateInvalidTransferOpen() (gas: 50066)
[PASS] testUpdateTemplateOperator(address,address,bool) (runs: 1024, µ: 67061, ~: 76856)
[PASS] testUpdateTemplateOperatorInvalidCaller(address,address,bool,address,bool) (runs: 1024, µ: 62504,
~: 50098)
[PASS] testUpdateTemplateValid(address,uint56,bool,bool,uint56,bool,bool) (runs: 1024, µ: 59125, ~:
59250)
[PASS] testWrap(address,uint8,uint256,uint256,address) (runs: 1024, µ: 252717, ~: 280848)
[PASS] testWrapInvalidAmount(address,uint8,uint256,uint256,address) (runs: 1024, µ: 208697, ~: 221647)
[PASS] testWrapInvalidTemplate(uint32,uint8,uint256,uint256,address) (runs: 1024, µ: 105661, ~: 114994)
[PASS] testWrapInvalidTokenType(address,uint8,uint256,uint256) (runs: 1024, µ: 144411, ~: 152209)
Suite result: ok. 42 passed; 0 failed; 0 skipped; finished in 14.67s (18.08s CPU time)

** Fix Review Update**

Ran 5 tests for test/tokens/wrappers/clawback/ClawbackMetadata.t.sol:ClawbackMetadataTest
[PASS] testDurationAndUnlocksAt() (gas: 500421)
[PASS] testMetadataPropertiesERC1155((uint8,uint32,uint56,uint256),(bool,bool,uint56,address)) (runs:
1024, µ: 155449, ~: 143255)
[PASS] testMetadataPropertiesERC20((uint8,uint32,uint56,uint256),(bool,bool,uint56,address)) (runs: 1024,
µ: 160255, ~: 160174)
[PASS] testMetadataPropertiesERC721((uint8,uint32,uint56,uint256),(bool,bool,uint56,address)) (runs:
1024, µ: 257600, ~: 254338)
[PASS] testSupportsInterface() (gas: 9592)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 557.34ms (1.46s CPU time)

Ran 49 tests for test/tokens/wrappers/clawback/Clawback.t.sol:ClawbackTest
[PASS] testAddTemplate(address,uint56,bool,bool) (runs: 1024, µ: 46910, ~: 46910)
[PASS] testAddTemplateTransferer(address,address) (runs: 1024, µ: 77227, ~: 77227)
[PASS] testAddTemplateTransfererInvalidCaller(address,address,bool,address) (runs: 1024, µ: 63504, ~:
75750)
[PASS] testAddToWrap(address,uint8,uint256,uint256,uint56,address,uint64) (runs: 1024, µ: 285307, ~:
260535)
[PASS] testAddToWrapInvalidWrappedId(uint256,uint256,address) (runs: 1024, µ: 13962, ~: 13962)
[PASS] testClawback(uint8,uint256,uint256,uint56,address,address) (runs: 1024, µ: 277875, ~: 300591)
[PASS] testClawbackAfterTransfer(uint8,uint256,uint256,uint56,address,address,address) (runs: 1024, µ:
306133, ~: 324569)
[PASS] testClawbackDestructionOnly(uint8,uint256,uint256,uint56,address) (runs: 1024, µ: 280416, ~:
305386)
[PASS] testClawbackDestructionOnlyInvalidReceiver(uint8,uint256,uint256,uint56,address,address) (runs:
1024, µ: 265706, ~: 292743)
[PASS] testClawbackInvalidCaller(uint8,uint256,uint256,uint56,address,address) (runs: 1024, µ: 233755, ~:
259026)
```

```
[PASS] testClawbackInvalidUnlocked(uint8,uint256,uint256,uint56,address,address) (runs: 1024, μ: 259327,
~: 286546)
[PASS] testEmergencyClawback(uint8,uint256,uint256,uint56,address,address) (runs: 1024, μ: 289553, ~:
317480)
[PASS] testEmergencyClawbackAfterTransfer(uint8,uint256,uint256,uint56,address,address,address) (runs:
1024, μ: 316318, ~: 340954)
[PASS] testEmergencyClawbackDestructionOnly(uint8,uint256,uint256,uint56,address) (runs: 1024, μ: 290688,
~: 321415)
[PASS] testEmergencyClawbackDestructionOnlyInvalidReceiver(uint8,uint256,uint256,uint56,address,address)
(runs: 1024, μ: 266158, ~: 293263)
[PASS] testEmergencyClawbackInvalidCaller(uint8,uint256,uint256,uint56,address,address) (runs: 1024, μ:
232086, ~: 258094)
[PASS] testEmergencyClawbackInvalidUnlocked(uint8,uint256,uint256,uint56,address,address) (runs: 1024, μ:
262103, ~: 286956)
[PASS] testPreventsOnERC1155Received(uint256,uint256) (runs: 1024, μ: 100407, ~: 100332)
[PASS] testPreventsOnERC1155Received(uint256,uint256,uint256,uint256) (runs: 1024, μ: 159323, ~: 159326)
[PASS] testPreventsOnERC721Received(uint256) (runs: 1024, μ: 118353, ~: 118352)
[PASS] testSupportsInterface() (gas: 13448)
[PASS] testTransferByTransfererAsFrom(uint8,uint256,uint256,uint56,uint64,bool,address,bool) (runs: 1024,
μ: 278904, ~: 303231)
[PASS] testTransferByTransfererAsOperator(uint8,uint256,uint256,uint56,uint64,bool,address,address,bool)
(runs: 1024, μ: 301022, ~: 326897)
[PASS] testTransferByTransfererAsTo(uint8,uint256,uint256,uint56,uint64,bool,address,bool) (runs: 1024,
μ: 280017, ~: 303782)
[PASS] testTransferByTransfererNotApproved(uint8,uint256,uint256,uint56,uint64,bool,address,address,bool)
(runs: 1024, μ: 270827, ~: 294956)
[PASS] testTransferInvalidOperator(uint8,uint256,uint256,uint56,address,address,bool) (runs: 1024, μ:
269180, ~: 294441)
[PASS] testTransferOpen(uint8,uint256,uint256,uint56,uint64,address) (runs: 1024, μ: 250485, ~: 275161)
[PASS] testUnwrap(address,uint8,uint256,uint256,uint56,uint64) (runs: 1024, μ: 236173, ~: 243143)
[PASS] testUnwrapAfterTransfer(address,uint8,uint256,uint256,uint56,uint64,address) (runs: 1024, μ:
273158, ~: 288725)
[PASS] testUnwrapByInvalidOperator(address,address,uint8,uint256,uint256,uint56) (runs: 1024, μ: 232852,
~: 259371)
[PASS] testUnwrapByOperator(address,address,uint8,uint256,uint256,uint56,uint64) (runs: 1024, μ: 268524,
~: 276227)
[PASS] testUnwrapInvalidAmount(address,uint8,uint256,uint256,uint256,uint56) (runs: 1024, μ: 231763, ~:
258219)
[PASS] testUnwrapInvalidToken(address,uint8,uint256,uint256,uint256,uint56) (runs: 1024, μ: 238643, ~:
263517)
[PASS] testUnwrapTokenLocked(address,uint8,uint256,uint256,uint56) (runs: 1024, μ: 228704, ~: 255574)
[PASS] testUpdateTemplateAdmin(address,address,uint56,bool,bool) (runs: 1024, μ: 76260, ~: 76260)
[PASS] testUpdateTemplateAdminInvalidAdmin() (gas: 49830)
[PASS] testUpdateTemplateAdminInvalidCaller(address,address,bool) (runs: 1024, μ: 62393, ~: 50400)
[PASS] testUpdateTemplateInvalidCaller(address,address,bool,uint56,bool,bool) (runs: 1024, μ: 63489, ~:
75910)
[PASS] testUpdateTemplateInvalidDestructionOnly() (gas: 49760)
[PASS] testUpdateTemplateInvalidDuration(address,uint56,bool,bool,uint56) (runs: 1024, μ: 51268, ~:
51268)
[PASS] testUpdateTemplateInvalidTransferOpen() (gas: 50208)
[PASS] testUpdateTemplateOperator(address,address,bool) (runs: 1024, μ: 67210, ~: 76985)
[PASS] testUpdateTemplateOperatorInvalidCaller(address,address,bool,address,bool) (runs: 1024, μ: 63184,
~: 63184)
[PASS] testUpdateTemplateValid(address,uint56,bool,bool,uint56,bool,bool) (runs: 1024, μ: 59332, ~:
59185)
[PASS] testWrap(address,uint8,uint256,uint256,address) (runs: 1024, μ: 254546, ~: 281166)
[PASS] testWrapInvalidAmount(address,uint8,uint256,uint256,address) (runs: 1024, μ: 209303, ~: 221861)
[PASS] testWrapInvalidRewrapping(uint8,uint256,uint256,uint56,address) (runs: 1024, μ: 229647, ~: 254783)
[PASS] testWrapInvalidTemplate(uint32,uint8,uint256,uint256,address) (runs: 1024, μ: 105799, ~: 115140)
[PASS] testWrapInvalidTokenType(address,uint8,uint256,uint256) (runs: 1024, μ: 144812, ~: 152283)
Suite result: ok. 49 passed; 0 failed; 0 skipped; finished in 732.34ms (6.38s CPU time)
```

Code Coverage

The branch coverage for all files stands at 77.80%, which can be improved.

Overall coverage rate: lines......: 86.6% (531 of 613 lines) functions......: 81.4% (149 of 183 functions) branches......: 77.8% (140 of 180 branches)

Changelog

- 2024-07-08 - Initial report
- 2024-07-24 - 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

Quantstamp

Sequence - Clawback