Quantstamp

TokenBeamer

Executive Summary

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

Type	Batch Token Transfer Utility
Timeline	2024-05-30 through 2024-05-31
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Source Code	• https://github.com/Merit-Circle/tokenbeamer #5b4f92d
Auditors	• Jeffrey Kam Auditing Engineer • Valerian Callens Senior Auditing Engineer • Gereon Mendler Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	3 Fixed: 2 Acknowledged: 1
High severity findings	0
Medium severity findings	0
Low severity findings	2 Fixed: 2
Undetermined severity findings	0
Informational findings	1 Acknowledged: 1

Summary of Findings

TokenBeamer is a simple contract that helps facilitate the process of transferring multiple tokens in one transaction. It supports various token standards, namely Native, ERC20, ERC721, and ERC1155.

Our team found two low-severity issues and one info-severity issue related to minor inconsistencies and nuanced behaviors of token approvals in different token standards. Overall, the contract is well-written and easy to understand, supported by a robust test suite.

Update: The team has addressed all the issues by either fixing or acknowledging them. We appreciate the team's responsiveness and commitment to security.

ID	DESCRIPTION	SEVERITY	STATUS
TB-1	**Token Type Defaults to `ERC1155` when Getting Approvals for Tokens without Id**	Low	Fixed
TB-2	**`ERC721` Approval Is Not Checked Properly**	Low	Fixed
TB-3	**Tip for `_tipRecipient` Is Not Guaranteed**	Informational	Acknowledged

Operational Considerations

The protocol is upgradable. We assume that the protocol will only go through well-planned, audited protocol upgrades.

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/Merit-Circle/tokenbeamer(5b4f92d97dff162ec3804e3d5db9ada4236c8106)

Files: contracts/TokenBeamer.sol

Files Excluded

Repo: https://github.com/Merit-Circle/tokenbeamer(5b4f92d97dff162ec3804e3d5db9ada4236c8106)

Files: Everything else

Findings

TB-1
Token Type Defaults to `ERC1155` when Getting Approvals for Tokens without Id

Low | Fixed

> **Update**
> Fixed in `29e79cd9ba197936eff9deb12a28e05bc8c92fe5`.
> The team added code comments to describe the changed behavior:
>
> `Type defaults to ERC-721 if ids are provided, ERC-20 if values are provided, and ERC-1155 otherwise.`

File(s) affected: `TokenBeamer.sol`

Description: In `_getApprovals()`, the token type defaults to `1155` when `hasTypes && hasIds` is false. However, if `hasIds` is false, it is unclear why the token type should be 1155 because `ERC1155` tokens are identified by unique `ID`s (see here).

Recommendation: Confirm whether this is intended. If not, adjust the token type accordingly and document the expected behaviour.

TB-2 `ERC721` Approval Is Not Checked Properly

Low Fixed

Update

Fixed in `26199b48102b140ac9573996ee62af79a98ba77c`.
The team provided the following comment:

`Added the suggested method and added an addition check on ownership of the id for ERC721.`

File(s) affected: `TokenBeamer.sol`

Description: In `_getApproval()`, the approval of an `ERC721` token is checked the following way:

```
IERC721(token).getApproved(id) == operator;
```

However, `ERC721` has another way of checking approval similar to `ERC1155`, namely `isApprovedForAll()`, which is a different mechanism from `getApproved()`. In OpenZeppelin `ERC721` implementation, it uses `_isAuthorized()` to check if a `spender` is approved to spend the `owner`'s tokens (see below or here), which uses both `isApprovedForAll()` and `_getApproved()`.

```
function _isAuthorized(address owner, address spender, uint256 tokenId) internal view virtual returns
(bool) {
    return
        spender != address(0) &&
        (owner == spender || isApprovedForAll(owner, spender) || _getApproved(tokenId) == spender);
}
```

Recommendation: Consider following the implementation of `_isAuthorized()` to check if the `operator` is approved to use the `owner`'s tokens.

TB-3 Tip for `_tipRecipient` Is Not Guaranteed

Informational Acknowledged

Update

The team provided the following comment:

`This is intended.`

File(s) affected: `TokenBeamer.sol`

Description: In the current implementation, we cannot guarantee there will be tips for the recipient as it simply transfers what remains in the contract to the `_tipRecipient`. A user can simply transfer all the native ETH to an address in the list `to`, without leaving any ETH behind.

Recommendation: Confirm if this is intended. If not, ensure a certain percentage of the funds will be transferred to `_tipRecipient` as fees.

Auditor Suggestions

TB-S-1 Events Emitted by the Contracts Could Be Improved

Fixed

Update

Fixed in `e5cc79c5fc22e06ab661417eb582e5d59c3ffb98`.
The team added the relevant event emission.

File(s) affected: `TokenBeamer.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks. Below we present a non-exhaustive list of events that could be emitted to improve application management:

1. `_tipRecipient` being set in the function `initialize()` and updated in the function `setTipRecipient()`.
2. `_upgradesDisabled` being set in the function `disableUpgrades()`.

Recommendation: Consider emitting the events.

TB-S-2 Risks Around Unlocked Solidity Version

Fixed

> **Update**
> Fixed in `e49c814ebccc8199622e4ab4b89645898fd06318`.

File(s) affected: `TokenBeamer.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". Currently, the contract uses version `^0.8.25`.

It should also be noted that newer versions of solc > `0.8.18` added the `PUSH0` opcode and contracts compiled at these versions may be incompatible with L2 chains that cannot interpret this opcode correctly.

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend removing the caret to lock the file onto a specific Solidity version. Additionally, if deployment on L2s without the support of `PUSH0` is intended, use `pragma 0.8.18` instead.

TB-S-3 Ownership Can Be Renounced

Acknowledged

> **Update**
> Addressed in `e49c814ebccc8199622e4ab4b89645898fd06318`.
> The team acknowledged the issue and provided the following comment:
>
> `This is intended. Added Ownable2Step for extra security.`

File(s) affected: `TokenBeamer.sol`

Description: If the owner renounces their ownership, the contract 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).

Key Actors And Their Capabilities

The contract has an owner role, which allows it to remove upgradeability, set the tip recipient, and retrieve lost tokens. It should be noted that it does not have the ability to steal funds from users who transfer tokens using this tool.

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).

Adherence to Best Practices

1. Fixed In `_processTransfer()`, `IERC721(token).safeTransferFrom(from, to, id, "");` can be replaced with `IERC721(token).safeTransferFrom(from, to, id);`
2. Fixed The `getApprovals()` function is missing NatSpec comments for the `ids` parameter.
3. The keyword `public` can be used for the state variables `_tipRecipient` and `_upgradesDisabled` to automatically generate a public view function.
4. The team could consider the integration of the function `safeBatchTransferFrom()` of the `ERC1155` standard.

Adherence to Specification

The different cases and shortcuts expected by the function `_getApprovals()` could be exhaustively clarified to help external observers make sure all cases are implemented as expected, especially for the parameter `ids`.

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.

Files

- `2f5...51d ./contracts/TokenBeamer.sol`

Tests

- `e72...81f ./test/MockERC1155.sol`
- `13a...1d9 ./test/MockERC20.sol`
- `a46...2f5 ./test/types.ts`
- `e6f...005 ./test/MockERC721.sol`
- `f05...50a ./test/TokenBeamer.t.sol`
- `ffe...ebe ./test/greeter/Greeter.ts`
- `697...a8d ./test/greeter/Greeter.fixture.ts`
- `46f...42c ./test/greeter/Greeter.behavior.ts`

Toolset

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

Setup

Tool Setup:

- Slither v0.10.0

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

All issues have either been incorporated into the report or disregarded as false positives.

Test Suite Results

All tests passed.

```
Ran 28 tests for test/TokenBeamer.t.sol:TokenBeamerTest
[PASS] testFuzz_BeamTokens_AllTokens(uint256,uint256) (runs: 256, μ: 8406658, ~: 5173019)
[PASS] testFuzz_BeamTokens_ERC1155_MultipleERC1155_MultipleReceivers(uint256,uint256) (runs: 256, μ:
4445036, ~: 2147651)
[PASS] testFuzz_BeamTokens_ERC1155_MultipleERC1155_OneReceiver(uint256,uint256) (runs: 256, μ: 6196113,
~: 2142657)
[PASS] testFuzz_BeamTokens_ERC1155_SingleERC1155_MultipleReceivers(uint256,uint256) (runs: 256, μ:
1420986, ~: 1131705)
[PASS] testFuzz_BeamTokens_ERC1155_SingleERC1155_OneReceiver(uint256,uint256) (runs: 256, μ: 1284714, ~:
1126797)
[PASS] testFuzz_BeamTokens_ERC20_MultipleReceivers(uint256,uint256) (runs: 256, μ: 2995576, ~: 1080391)
[PASS] testFuzz_BeamTokens_ERC20_OneReceiver(uint256,uint256) (runs: 256, μ: 3055694, ~: 1078040)
[PASS] testFuzz_BeamTokens_ERC721_MultipleERC721_MultipleReceivers(uint256) (runs: 256, μ: 6953556, ~:
1905363)
[PASS] testFuzz_BeamTokens_ERC721_MultipleERC721_OneReceiver(uint256) (runs: 256, μ: 5509346, ~: 1900348)
[PASS] testFuzz_BeamTokens_ERC721_SingleERC721_MultipleReceivers(uint256) (runs: 256, μ: 1294625, ~:
1030058)
[PASS] testFuzz_BeamTokens_ERC721_SingleERC721_OneReceiver(uint256) (runs: 256, μ: 1136869, ~: 1005186)
[PASS] testFuzz_BeamTokens_ETH_MultipleReceivers(uint256,uint256) (runs: 256, μ: 260176, ~: 103689)
[PASS] testFuzz_BeamTokens_ETH_OneReceiver(uint256,uint256) (runs: 256, μ: 116120, ~: 75095)
[PASS] testFuzz_BeamTokens_RevertWhen_BadInput(uint256,uint256) (runs: 256, μ: 41604, ~: 37211)
[PASS] testFuzz_BeamTokens_TransferTips(uint256,uint256,uint256) (runs: 256, μ: 408293, ~: 156189)
[PASS] testFuzz_GetApprovals_AllTokens(uint256,uint256) (runs: 256, μ: 7917998, ~: 5114770)
[PASS] testFuzz_GetApprovals_NFTsWithIds(uint256,uint256) (runs: 256, μ: 7343330, ~: 4024145)
[PASS] testFuzz_GetApprovals_NFTsWithoutIds(uint256,uint256) (runs: 256, μ: 7311893, ~: 4024038)
[PASS] testFuzz_GetApprovals_NoValue(uint256,uint256) (runs: 256, μ: 9614083, ~: 5107618)
[PASS] testFuzz_ProcessTransfer_RevertWhen_BadInput(uint256,uint256) (runs: 256, μ: 41779, ~: 35319)
[PASS] testFuzz_ProcessTransfer_RevertWhen_UnsupportedTokenType(uint256,uint256) (runs: 256, μ: 44380, ~:
38337)
[PASS] testFuzz_RecoverFunds() (gas: 36295)
[PASS] testFuzz_setTipRecipient(address) (runs: 256, μ: 26847, ~: 26847)
[PASS] testFuzz_setTipRecipient_RevertWhen_BadInput(address) (runs: 256, μ: 27542, ~: 27542)
[PASS] test_DisableUpgrades() (gas: 26439)
[PASS] test_GetApprovals_RevertWhen_BadInput() (gas: 20385)
[PASS] test_GetApprovals_RevertWhen_UnsupportedTokenType() (gas: 23062)
[PASS] test_Receive_RevertWhen_EthSent() (gas: 23934)
Suite result: ok. 28 passed; 0 failed; 0 skipped; finished in 491.63ms (3.59s CPU time)
```

Code Coverage

The branch coverage of TokenBeamer is at around 80%. We recommend improving the test suite to achieve 90% or higher of branch coverage.

File	% Lines	% Statements	% Branches	% Funcs
contracts/TokenBeamer.sol	100.00% (**52/**52)	100.00% (**84/**84)	80.77% (**21/**26)	90.91% (**10/**11)
test/MockERC1155.sol	100.00% (**1/**1)	100.00% (**1/**1)	100.00% (**0/**0)	100.00% (**1/**1)
test/MockERC20.sol	100.00% (**1/**1)	100.00% (**1/**1)	100.00% (**0/**0)	100.00% (**1/**1)
test/MockERC721.sol	100.00% (**1/**1)	100.00% (**1/**1)	100.00% (**0/**0)	100.00% (**1/**1)
Total	100.00% (**55/**55)	100.00% (**87/**87)	80.77% (**21/**26)	92.86% (**13/**14)

Changelog

- 2024-05-31 - Initial report
- 2024-06-13 - 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