Synthetix Vest Tool Smart Contract Audit

SYNTHETIX

Vest Tool
Smart Contract Audit

iosiro

1. Introduction

iosiro was commissioned by Synthetix to conduct a smart contract audit of their Tokenvest Tool. The audit was performed by 1 auditor between 17 and 20 January 2022, using 2 resource days. Changes to the contract made in response to user testing and internal code review were reviewed on 7 and 28 March 2022, using 1 resource day.

This report is organized into the following sections.

- **Section 2 - Executive summary:** A high-level description of the findings of the audit.
- **Section 3 - Audit details:** A description of the scope and methodology of the audit.
- **Section 4 - Design specification:** An outline of the intended functionality of the smart contracts.

- **Section 5 - Detailed findings:** Detailed descriptions of the findings of the audit.

The information in this report should be used to understand the smart contracts' risk exposure better and as a guide to improving the security posture of the smart contracts by remediating issues identified. The results of this audit reflect the in-scope source code reviewed at the time of the audit.

The purpose of this audit was to achieve the following:

- Identify potential security flaws.
- Ensure that the smart contracts function according to the documentation provided.

Assessing the off-chain functionality associated with the contracts, for example, backend web application code, was outside of the scope of this audit.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered high risk from cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. Strategies that should be used to encourage secure code development include:

- Security should be integrated into the development lifecycle, and the level of perceived security should not be limited to a single code audit.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed where possible.

2. Executive summary

This report presents the findings of an audit performed by iosiro on the Synthetix Tokenvest Tool, a smart contract written to manage token grants with vesting schedules.

Audit findings

The audit discovered several informational issues related to best practice. All findings were addressed during the audit.

User concerns

Users of the contract should note that the contract owner has significant power over grants issued through this contract. The owner can, at any time:

- Cancel any issued grant.
- Replace any issued grant with another grant with different details. Details include the grant token, start and cliff time, the total amount, the vesting amount, and the vesting interval.
- Withdraw all ERC-20 tokens deposited in the contract. Users with redeemable funds will need to wait for the contract to be resupplied before they will be able to redeem their funds.

The ownership of this contract is a multi-sig wallet that also owns the grant funds.

3. Audit details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from all other files was considered to be out-of-scope. Out-of-scope code that interacts with in-scope code was assumed to function as intended and not introduce any functional or security vulnerabilities for the purposes of this audit.

3.1.1 Smart contracts

- **Project name:** Tokenvest
- **Inital audit commit:** 36573f2
- **Final review commit:** 76b11f9
- **Files:** Vester.sol

3.2 Methodology

The audit was conducted using a variety of techniques described below.

3.2.1 Code review

The source code was manually inspected to identify potential security flaws. Code review is a useful approach for detecting security flaws, discrepancies between the specification and implementation, design improvements, and high-risk areas of the system.

3.2.2 Dynamic analysis

The contracts were compiled, deployed, and tested in a test environment, both manually and through the test suite provided. Manual analysis was used to confirm that the code was functional and discover security issues that could be exploited. The coverage report of the provided tests as on the final day of the audit is given below.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.21	88.46	100	98.28	
Vester.sol	98.21	88.46	100	98.28	123
contracts/test-helpers/	100	100	100	100	
SampleToken.sol	100	100	100	100	
All files	98.25	88.46	100	98.31	

3.2.3 Automated analysis

Tools were used to automatically detect the presence of several types of security vulnerabilities, including reentrancy, timestamp dependency bugs, and transaction-ordering dependency bugs. Static analysis results were reviewed manually and any false positives were removed. Any true positive results are included in this report.

Static analysis tools commonly used include Slither, Securify, and MythX. Tools such as the Remix IDE, compilation output, and linters could also be used to identify potential areas of concern.

3.3 Risk ratings

Each issue identified during the audit has been assigned a risk rating. The rating is determined based on the criteria outlined below.

- **High risk** - The issue could result in a loss of funds for the contract owner or system users.
- **Medium risk** - The issue resulted in the code specification being implemented incorrectly.
- **Low risk** - A best practice or design issue that could affect the security of the contract.
- **Informational** - A lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.
- **Closed** - The issue was identified during the audit and has since been satisfactorily addressed, removing the risk it posed.

4. Design specification

The specification for the Tokenvest Tool was based on the Technical Specification of the repository readme at commit 76b11f9.

5. Detailed findings

The following section details the findings of the audit.

5.1 High risk

No identified high-risk issues were open at the conclusion of the review.

5.2 Medium risk

No identified medium-risk issues were open at the conclusion of the review.

5.3 Low risk

No identified low-risk issues were open at the conclusion of the review.

5.4 Informational

No identified informational issues were open at the conclusion of the review.

5.5 Closed

No issues were closed during the audit.

5.5.1 Use of unsafe ERC-20 transfer functions (informational)

Vester.sol#L43,54,93

Description

The contracts did not use the `safeTransfer` and `safeTransferFrom` functions from the OpenZeppelin SafeERC20 library. `safeTransfer(...)` and `safeTransferFrom(...)` abstract the standard ERC-20 `transfer(...)` and `transferFrom(...)` functions and throws an exception if the transfer returns `false`. This adds an additional layer of validation in case the ERC-20 token returns `false` on failed transfers instead of reverting.

Recommendation

The OpenZeppelin SafeERC20 library should be used to wrap all important ERC-20 functions as a defense-in-depth measure.

Update

Addressed in b467c5d.

5.5.2 Re-entrancy (informational)

Vester.sol#L51,127

Description

Statements in the `redeemWithTransfer` and `mint` functions were ordered in a manner that could lead to a re-entrancy bug, as external contract calls were made before internal state changes.

Recommendation

This issue could be mitigated in both instances by following the checks-effect pattern and moving external contract calls to the end of the function.

Update

Addressed in b467c5d and dfb52a2.

5.5.3 Struct not cleared on token burn (informational)

Vester.sol#L136

Description

The `burn` function did not clear the associated `Grant` struct when burning a grant token. While burned `tokenId` s cannot be reused, it is considered best practice to delete data no longer in use.

Recommendation

The `burn` function should clear `grant[tokenId]` .

Update

Addressed in dfb52a2.

5.5.4 Design comments

Actions to improve the functionality and readability of the codebase are outlined below.

Gas optimization

In the `amountVested(..)` function, loading `storage grants[tokenId]` once at the start of the function will result in a ~300 gas saving when calling `redeem(...)`.

Update

Addressed in b467c5d.

Storage considerations

Vester.sol#L23

The state variable `tokenAddress` could be made immutable. Immutable state variables save storage slots, which in turn reduce gas fees for reading.

Update

Addressed in b467c5d.

Comment and error message clarity

1. The comment on Vester.sol#L60 reads "Calculate the amount of tokens currently available for redemption for a given *grantee*". "Calculate the amount of tokens currently available for redemption for a given *grant*" would be more accurate.
2. The comment on Vester.sol#107 reads "Calculate the amount that has vested for a given *address*". "Caculate the amount that has been vested for a given *grant*" would be more accurate.
3. The require message on Vester.sol#L66 could be expanded to "No tokens available for redemption in grant". This would provide a clearer error when a the redemption of a single grant fails during a call to `redeemMultiple(...)` than the current message, "No tokens available for redemption".

Update

1. Addressed in dfb52a2.
2. Addressed in 76b11f9.
3. Addressed in 76b11f9.

Secure your system.

Request a service

START NOW →

ABOUT

CONTACT US

SMART CONTRACT AUDITING

PENETRATION TESTING

AUDIT REPORTS

PRIVACY POLICY

TERMS OF SERVICE