HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Hedgey
Date: April 6, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Hedgey
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	Vesting; ERC721
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://hedgey.finance/
Changelog	08.03.2023 - Initial Review 22.03.2023 - Second Review 06.04.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Hedgey (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository	https://github.com/hedgey-finance/StreamVestingNFT
Commit	05028ffa5d1c024d43ce68054b7df29fb546287b
Whitepaper	Not provided
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/BatchStreamer.sol SHA3: 47b3f6fde1499ef944ff81dc4a76b4558aca0c82281b91f7846c1dbfd1f38329 File: ./contracts/BatchVesting.sol SHA3: 5ea4fb7ca51c9a7b59c3a162873992fac74ebc6fd68f7e04df381b59f862b7b6 File: ./contracts/ERC721Delegate/ERC721Delegate.sol SHA3: d7ef641d6dd52c2b226e30773e68bf0ee82844aefcd3816b0b1468d81eb189f8 File: ./contracts/interfaces/IStreamNFT.sol SHA3: 1b26312a58385af58e59beea86f4960946a05c08cee8a0abee041832b3690f38 File: ./contracts/interfaces/IVestingNFT.sol SHA3: b74163b1beb5b49b337aa8822b38b91926c376d76497331aedeeab2cf6380e25 File: ./contracts/libraries/StreamLibrary.sol SHA3: 462f96cda4b37f549e402b5648fd0586c0981bed0122d32cc6c1825c375f97d8 File: ./contracts/libraries/TransferHelper.sol SHA3: a93cf2a9bfec370bdae0209dcd7a85c6820303e01b5d5fc839da5ceb0b714eed File: ./contracts/StreamingBoundNFT.sol SHA3: c80b3126ecf859136085144ceec9976a0026ff1bdc506d23270ba8eef3872f13 File: ./contracts/StreamingNFT.sol SHA3: 320899e8ef83297d63689a8ed09778e55c1dbad6d49630138ba79b5030118076 File: ./contracts/StreamVestingNFT.sol SHA3: 033a5f4688f033c36b6c3824ff7acb8cc49e2c50272cbe28bfa8f1692454e510

Second review scope

Repository	https://github.com/hedgey-finance/StreamVestingNFT
Commit	d1a143420b972d8363f5f81769b907d532b4964a
Whitepaper	Not provided
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/BatchStreamer.sol SHA3: 48d7d5602bd355cb9156f28cfeeb1bece9d934d35ed7cdbe3baff84be1dd01a8 File: ./contracts/BatchVester.sol SHA3: af4058cb491c40eda512de34cd3bfae9056dd1f1c5e57443b448c1c6a509b9b5 File: ./contracts/ERC721Delegate/ERC721Delegate.sol SHA3: 9854cebd59f7c16bd369bcdeeeceb9fbdf408b96932f5f40f8e023bd93fee9f0 File: ./contracts/ERC721Delegate/IERC721Delegate.sol SHA3: 258a284dc2d59cf92f95dd90d9c9684018e5b5796cb5cd1c8e6b7a897a0e3195 File: ./contracts/interfaces/IStreamNFT.sol SHA3: 7246a69fc2dec9c12f7c7e523f55d3827c80f1a0a02210303af4b9a926099ed3 File: ./contracts/interfaces/IVestingNFT.sol SHA3: 214152c799612fada6103c5361286a11f4f13840cd83386c531e6db187d5b78b File: ./contracts/libraries/StreamLibrary.sol SHA3: 93141a748cabf3c1717056906dd6e77756e5bdb7401f9b9a7c9ee98db686905a File: ./contracts/libraries/TransferHelper.sol SHA3: b632fdc4188f94839e9829c38e4b6da498d61fefa36b1dadc9eed389575b1b1d File: ./contracts/StreamingBoundHedgeys.sol SHA3: a732903374483b59b3d2ef1897335c9ca13c02d54d27b647abd25c81e1d402e9 File: ./contracts/StreamingHedgeys.sol SHA3: ac13781718924900744793157bcd99b792962169f728da515aaf8fdcf45f9868 File: ./contracts/StreamingNFT.sol SHA3: 05fc1f891b12d113e85ac38481fb202ce025f27257229dfa2fa26adef867cae7 File: ./contracts/StreamVestingNFT.sol SHA3: a85f515e10f6c14e7209a0c63e3ac4122cc3e72adec74f39001c4e8930bdf463

Third review scope

Repository	https://github.com/hedgey-finance/StreamVestingNFT
Commit	59e62e79f5ab2f94c8ecab75f14c7af7d851b9f0
Whitepaper	Not provided

Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/BatchStreamer.sol SHA3: 48d7d5602bd355cb9156f28cfeeb1bece9d934d35ed7cdbe3baff84be1dd01a8 File: ./contracts/BatchVester.sol SHA3: af4058cb491c40eda512de34cd3bfae9056dd1f1c5e57443b448c1c6a509b9b5 File: ./contracts/ERC721Delegate/ERC721Delegate.sol SHA3: faeabaf1749b1325685a0241e7a44e47ac1e08f0d394e79f1d4f698f17280398 File: ./contracts/ERC721Delegate/IERC721Delegate.sol SHA3: 258a284dc2d59cf92f95dd90d9c9684018e5b5796cb5cd1c8e6b7a897a0e3195 File: ./contracts/interfaces/IStreamNFT.sol SHA3: 7246a69fc2dec9c12f7c7e523f55d3827c80f1a0a02210303af4b9a926099ed3 File: ./contracts/interfaces/IVestingNFT.sol SHA3: 214152c799612fada6103c5361286a11f4f13840cd83386c531e6db187d5b78b File: ./contracts/libraries/StreamLibrary.sol SHA3: 93141a748cabf3c1717056906dd6e77756e5bdb7401f9b9a7c9ee98db686905a File: ./contracts/libraries/TransferHelper.sol SHA3: b632fdc4188f94839e9829c38e4b6da498d61fefa36b1dadc9eed389575b1b1d File: ./contracts/StreamingBoundHedgeys.sol SHA3: 4320d05b6d9712f6cf8a8f4c4cd0060caa1d844819498dd1f7e709ec93b0e27b File: ./contracts/StreamingHedgeys.sol SHA3: bf16cf2f1faaa31b93d29be34d01577c362e3512d7d374f0e72a9982e3927ad4 File: ./contracts/StreamingNFT.sol SHA3: 3a3e7298c6373642cdcd32274876e27ed31cb6a968f657389e93d32a49085965 File: ./contracts/StreamVestingNFT.sol SHA3: 06d1989fcf9c884d5503a8b5d2675a5927ea0327c2a34234014dd53b8e6558a9

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- The functional and technical requirements are provided and comprehensive.
- The code is covered with the NatSpec comments.

Code quality

The total Code Quality score is **9** out of **10**.

- Mixed order of functions and variable declaration.

Test coverage

Code coverage of the project is **97.4%** (branch coverage).

- Negative cases and interactions by several users testing are partially missed.

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

The system users should acknowledge all the risks summed up in the risks section of the report.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.7**.

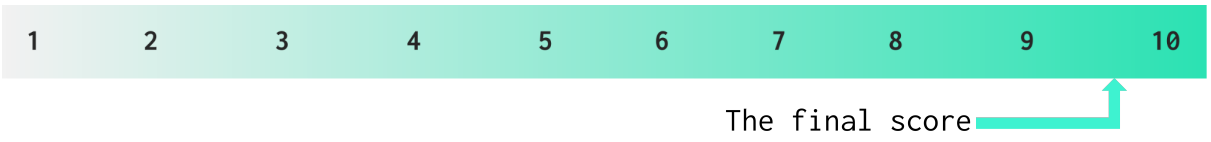

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
8 March 2023	14	7	1	0
22 March 2023	3	0	0	0
6 April 2023	0	0	0	0

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

The provided project is a vesting system with the following contracts:

- *ERC721Delegate* – is an abstract ERC-721 token contract (inherits *OpenZeppelin ERC721* contract) with enumerable functionality. The contract contains the delegating functionality: it stores the tokens' delegates. When the token is minted, its delegate is set to the token owner, when the token is transferred, its delegate is being set to the token receiver, when it is burnt the token's delegate record is removed. The contract allows to obtain each address delegated balance, each token's delegate, and the delegated token id by the delegate address and the index.
- *StreamingNFT* – is a contract that inherits *ERC721Delegate* contract with the vesting functionality.

  The token's name, symbol and admin are defined when the contract deployment. The admin can set the base token URI and remove the admin from the contract.

  The contract allows the creation of NFT, it includes the vesting: the creator defines the recipient, ERC-20 token address, amount of the defined ERC-20 token, vesting start date, cliff date, rate (amount of tokens vested per each second). The ERC-20 tokens are transferred from the creator, the NFT token is minted to the recipient and the corresponding vesting stream record is created.

  The contract allows NFT receivers to redeem their vesting streams. The amount that can be redeemed is calculated according to the vesting start date, vesting rate (amount of tokens vested per each second), current time and the vesting cliff time (the tokens can not be redeemed until it is reached). The vestings can be partially redeemed, when they are fully redeemed, the appropriate NFT is burnt from its owner and the vesting record is deleted.

  The contract provides functions that allow to redeem the defined by the caller NFTs' vestings or to all the NFTs' that are owned by the caller.

  There is a functionality that allows to delegate the owned NFTs (described in the *ERC721Delegate* contract functionality): the function that delegated the defined by the caller NFTs to the defined delegate address, and the function that allows to delegate all the NFTs owned by the caller to the defined delegate address.

  The contract allows to obtain the defined NFT vesting amounts (unlocked and locked), each NFT vesting end time, the not withdrawn balance of the defined user of the defined tokens (obtained from all the vestings held by the user), the delegated balance of the defined delegate of the defined tokens (obtained from all the vestings delegated to them).

- *StreamingHedgeys* – is a contract that repeats the *StreamingNFT* contract functionality, but provides additional function: to redeem the defined by the caller NFT and transfer it to the define by the caller address (this is only possible if the redeeming is not full and the NFT is not burned).
- *StreamingBoundHedgeys* – is a contract that repeats the *StreamingNFT* contract functionality, but is not transferable.
- *BatchStreamer* – is a contract that allows to create batches of NFT vesting streams (*StreamingHedgeys* / *StreamingBoundHedgeys*). The caller defines the streamer address (*StreamingHedgeys* / *StreamingBoundHedgeys*), ERC-20 token to be locked under NFTs, recipients, amounts of ERC-20 token, starts, cliffs and rates.
- *StreamVestingNFT* – is a contract that repeats the *StreamingHedgeys* contract functionality, but is not transferable, has the locking and revoking functionality.

  Locking functionality: when the contract deployment, the addresses of transferable and non-transferable NFTs for locking are set (*StreamingHedgeys* / *StreamingBoundHedgeys*). The locked and usual NFTs can be created, when creating the locked NFT, the caller defines the unlock date and if the locking NFT is transferable or not (to be minted when revoking). When redeeming tokens, there is an additional check that verifies if the current time has reached the unlock date.

  Redeeming functionality: when creating the NFT vesting, the vesting admin is set (it can not be equal to the recipient). The vesting admin may revoke it if it not fully vested: the NFT is burned, the vesting record is deleted, the not vested tokens are transferred to the vesting admin; if there are vested but not withdrawn tokens, they are processed accordingly to the vesting unlocking data: if the unlock date has reached or the vesting is not lockable, the vested balance is transferred to the beneficiary, otherwise, the NFT vesting is created to the beneficiary (*StreamingHedgeys* / *StreamingBoundHedgeys*) according to the defined while the vesting indicator if the token should be transferred or not, the parameters of the NFT to be created: the vested amount as an amount and the rate, unlock data as a start date and the cliff.
- *BatchVester* – is a contract that allows to create batches of NFT vesting streams (*StreamVestingNFT*). The caller defines the vester address (*StreamVestingNFT*), ERC-20 token to be locked under NFTs, recipients, amounts of ERC-20 token, starts, cliffs and rates, vesting admin, additional oner for the lockable vesting: unlock date and if the locking NFT is transferable or not.
- *StreamLibrary* – is a library that provides functions for the vesting amounts and end date calculations, used in the vesting contracts.

- *TransferHelper* – is a library that provides functions for the transferring token to the contract and from it, used in the vesting contracts.
- *IStreamNFT* – is an interface for the *StreamingHedgeys* / *StreamingBoundHedgeys* contracts.
- *IVestingNFT* – is an interface for the *StreamVestingNFT* contracts.
- *IERC721Delegate* – is an interface for the *ERC721Delegate* contracts.

Privileged roles

- The *StreamingHedgeys*, *StreamingBoundHedgeys*, *StreamVestingNFT* contracts have the privileged role of the admin:
    - The admin can set the base URI and remove the admin.

Risks

- Usage of *ERC721Enumerable*, *ERC721Delegate* significantly increases the minting Gas for NFTs (2-3x)
- If the base URIs of the NFTs contracts were set incorrectly and the admin was deleted, it would be impossible to set the correct base URI.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Failed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

No critical severity issues were found.

High

H01. Undocumented Behavior

When tokens are transferred, their delegates are changed to the token receiver. (*_transferDelegate* in the *_beforeTokenTransfer* function)

Such behavior is not described in the documentation and may lead to the incorrect assumption of the system logic.

Path:
./contracts/ERC721Delegate/ERC721Delegate.sol : _transferDelegate(), _beforeTokenTransfer()

Recommendation: clarify the delegating behavior while tokens transferring in the documentation and ensure that the implementation matches it.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434) (added to the documentation: “When a token is burned or transferred, the current delegate is removed via token being removed from the delegates index. It is important to change the delegate during a token transfer, because should the NFT be sold or transferred to an unrelated party, the new owner should not still have their token delegated to the previous owners delegate, or unwanted voting may occur.”)

Medium

M01. Best Practice Violation

The *cliff* parameter value is not validated in the *createNFT* functions.

Therefore, if it is mistakenly set to the too big value, the tokens may be locked in the contract for a large amount of time.

Paths:
./contracts/StreamingNFT.sol: createNFT()
./contracts/StreamingBoundNFT.sol : createNFT()
./contracts/StreamVestingNFT.sol : _createNFT()

Recommendation: ensure the *cliff* parameter value does not exceed the vesting end time.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

M02. Contradiction

The documentation states about the *deleteAdmin* function: “This function can be called by the admin after updating the baseURI”.

However, the functionality allows calling this function when the base URI is not set. Therefore, if the admin removes itself before the URI setting, it would be impossible to set the base URI.

Paths:
./contracts/StreamVestingNFT.sol : deleteAdmin()
./contracts/StreamingNFT.sol : deleteAdmin()
./contracts/StreamingBoundNFT.sol : deleteAdmin()

Recommendation: ensure the admin can be deleted only after the base URI is set.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

M03. Denial of Service Vulnerability

The *redeemAllNFTs* and *delegateAllNFTs* functions loop over all the user's NFTs and delegated balances.

If the numbers of elements in the arrays are large enough to increase the Gas required for executing the loop over the block Gas limit, the mentioned functionality may become inoperable.

Paths:
./contracts/StreamingNFT.sol : delegateAllNFTs(), redeemAllNFTs()
./contracts/StreamingBoundNFT.sol: delegateAllNFTs(), redeemAllNFTs()
./contracts/StreamVestingNFT.sol : delegateAllNFTs(), redeemAllNFTs()

Recommendation: consider updating the Gas model:

create the variable that stores the *balanceOf(msg.sender)* and use in in the loops conditions,

introduce the internal *_tokenOfOwnerByIndex* without the index check (as the loop operates only with the *msg.sender*`s tokens and the check is redundant) and use it instead of the *tokenOfOwnerByIndex*.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

M04. Undocumented Behavior

The contracts for the NFTs batch creation contain the duplicated functions that have the *mintType* parameter. However, the functionalities of the functions with this parameter and without it do not differ and this parameter is used only for the event emitting.

This may indicate that the requirements are violated or the code is not finalized. The usage of this parameter and the event emitting may lead to the incorrect assumption of the transaction execution way.

Paths:
./contracts/BatchStreamer.sol : createBatch()
./contracts/BatchVesting.sol : createBatch(), createLockedBatch()

Recommendation: clarify the requirements and implement the code according to them, ensure that the code does not contradict itself.

Found in: 05028ff

Status: Mitigated (the documentation: “This is an internal field that is called when using the Hedgey front end application. The field is picked up in an event and stored in the Hedgey database, and used for certain special visualizations on the front end. It is simply an identifier that Hedgey maintains internally for this extra usage, and has no technical or mechanical impact to the functionality of the smart contract itself.”)

M05. Inconsistent Data

Critical state changes should emit events for tracking things off-chain.

The following functions do not emit events on change of important values.

Paths:
./contracts/StreamingNFT.sol : deleteAdmin()
./contracts/StreamingBoundNFT.sol : deleteAdmin()
./contracts/StreamVestingNFT.sol : deleteAdmin()
./contracts/ERC721Delegate/ERC721Delegate.sol : _removeDelegate()

Recommendation: emit the event whenever the corresponding action happens.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

M06. Best Practice Violation

The `unlockDate` parameter value is not validated in the `createLockedNFT` function.

Therefore, if it is mistakenly set to the too big value, the minted locked tokens may be locked in the contract for a large amount of time.

Path:
./contracts/StreamVestingNFT.sol : createLockedNFT()

Recommendation: limit the maximal unlock date and ensure that the *unlockDate* parameter value is valid.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

M07. Inconsistent Data

The vesting contracts have the mapping that stores the *Stream* structure of vesting details data. *Stream* structure has the *start* field, which stores the data of the Vesting start date, but according to the current logic, the value may be changed during the "redeem" process. This makes the start date field unreliable for use on the frontend, because it may contain an irrelevant start date.

This leads to the inability to rely on the start date as a source of data on the frontend and on other contracts.

Paths:
./contracts/StreamingBoundNFT.sol : _redeemNFT()
./contracts/StreamingNFT.sol : _redeemNFT()
./contracts/StreamVestingNFT.sol : _redeemNFT()

Recommendation: rework the logic not to update the start date field and add the new field to use it for the vesting calculations.

Found in: 05028ff

Status: Mitigated (the documentation: "The internal function will also update the storage struct to reset the amount to the remaining tokens that are still vesting, and it will reset the start date to the current block timestamp.")

Low

L01. Redundant Parameter

The *from* parameter in the *_transferDelegate* function is not used.

The redundant code decreases the code readability.

Path:
./contracts/ERC721Delegate/ERC721Delegate.sol : _transferDelegate.from

Recommendation: remove the redundant code.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

L02. Insufficient Contract Description

The *ERC721Delegate* contract contains the contract description, it mentions the enumerability, but it does not mention the contract delegation functionality.

This may lead to an incorrect understanding of the contract functionality.

Path:
./contracts/ERC721Delegate/ERC721Delegate.sol

Recommendation: ensure that the contract description is comprehensive, contains the delegating functionality.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

L03. Incorrect Functions Titles

The functions' titles indicate that they handle one object, but they handle multiple ones.

Incorrect titles may lead to incorrect assumptions about the functions' purposes.

Paths:
./contracts/StreamingNFT.sol : delegateToken(), redeemNFT()
./contracts/StreamingBoundNFT.sol : delegateToken(), redeemNFT()
./contracts/StreamVestingNFT.sol : delegateToken(), revokeNFT(), redeemNFT()

Recommendation: ensure that the functions' titles match their functionality.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

L04. Public Functions That Could Be Declared External

There are *public* functions in the contracts that are not called inside the system.

Functions with *external* visibility use less Gas.

Paths:
./contracts/StreamingNFT.sol : delegatedBalances(), lockedBalances(), getStreamEnd()
./contracts/StreamingBoundNFT.sol : delegatedBalances(), lockedBalances(), getStreamEnd()
./contracts/StreamVestingNFT.sol : delegatedBalances(), lockedBalances(), getStreamEnd()

Recommendation: use *external* visibility for the functions that are never used inside the contracts.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

L05. Best Practice Violation

There are *uint* values in the contracts whose sizes are not set explicitly.

This decreases the code readability. The explicit type definition helps to understand the data size to proceed and detect errors.

Paths:
./contracts/StreamingNFT.sol : delegateToken() (uint[] tokenIds), redeemAllNFTs() (uint bal)
./contracts/StreamingBoundNFT.sol : redeemAllNFTs() (uint bal)

Recommendation: replace the uint type with the *uint256*.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

L06. Redundant Check

The *stream.unlockDate <= block.timestamp* check is redundant, as this verification is already performed in the *_redeemNFTs* function.

Redundant code decreases the code readability and leads to the redundant Gas consumption.

Path:
./contracts/StreamVestingNFT.sol : _redeemNFT()

Recommendation: remove the redundant check.

Found in: 05028ff

Status: Fixed (Revised commit: 59e62e7)

L07. Contradiction

The *lockedBalances* functions' title may indicate that it returns the balance that is not already unlocked in the vestings and cannot be redeemed. However, it returns the amount of tokens that have not been withdrawn yet (locked + unlocked).

This may lead to the incorrect assumption on the functions' purposes.

Paths:
./contracts/StreamVestingNFT.sol : lockedBalances()
./contracts/StreamingBoundNFT.sol : lockedBalances()
./contracts/StreamVestingNFT.sol : lockedBalances()

Recommendation: ensure that the function titles match their functionality.

Found in: 05028ff

Status: Mitigated (the documentation: “This function will aggregate all of the vesting token balances (both vested and unvested) across all NFTs that share the same underlying token address, and return the sum of all of them.”)

L08. Best Practice Violation

The contracts are missing interface inheritance and redefine the events which are already defined in the interfaces.

Paths:
./contracts/StreamVestingNFT.sol : NFTCreated, NFTRevoked, NFTRedeemed, URISet
./contracts/StreamingBoundNFT.soll : NFTCreated, NFTRedeemed, URISet
./contracts/StreamingNFT.sol : NFTCreated, NFTRedeemed, URISet

Recommendation: inherit the proper interface.

Found in: 05028ff

Status: Fixed (Duplicated events are removed) (Revised commit: d1a1434)

L09. Best Practice Violation

Not all the main parameters in the events are marked with the *indexed* keyword.

It is a good practice to mark important parameters for the filtering.

Paths:
./contracts/interfaces/IVestingNFT.sol : NFTCreated()
./contracts/StreamVestingNFT.sol : NFTCreated()

Recommendation: add the *indexed* keyword to the important parameters in the events.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

L10. Floating Pragma

Contracts with unlocked pragmas may be deployed by the latest compiler, which may have higher risks of undiscovered bugs.

Contracts should be deployed with the same compiler version they have been tested thoroughly.

Paths:
./contracts/libraries/StreamLibrary.sol
./contracts/libraries/TransferHelper.sol

Recommendation: lock the pragma.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

L11. NatSpec Missing

The NatSpec comments are missed in the *BatchStreamer* and *BatchVesting* contracts.

It is recommended that Solidity contracts are fully annotated using NatSpec.

Paths:
./contracts/BatchStreamer.sol
./contracts/BatchVesting.sol
./contracts/libraries/StreamLibrary.sol

Recommendation: annotate the contracts with NatSpec.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

L12. Contradiction

Contract names are not the same as the file names.

The best practice is to keep the same name for the file and the contract it contains.

Paths:
./contracts/BatchVesting.sol
./contracts/StreamingBoundNFT.sol
./contracts/StreamingNFT.sol

Recommendation: ensure that the contract names are not the same as the file names.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

L13. Typos in Comments

There are typos in the contracts.

They decrease the code readability.

Paths:
./contracts/ERC721Delegate/ERC721Delegate.sol - *puprose*, _transferDelegate() - *transfering*,

./contracts/StreamingBoundNFT.sol - *puprose*, delegateToken() - *wallt*, constructor() - *constructur*, lockedBalances(), delegatedBalances() - *enumarate*, *delagate*, createNFT() - *onesself*, *futured*, *paramater*,

./contracts/StreamingNFT.sol - *puprose*, delegateToken() - *wallt*, lockedBalances(), delegatedBalances() - *enumarate*, *delagate*, constructor() - *constructur*, createNFT() - *onesself*, *futured*, *paramater*,

./contracts/StreamVestingNFT.sol - *puprose*, delegateToken() - *wallt*, *delagate*, lockedBalances(), delegatedBalances() - *enumarate*, constructor() - *constructur*, createNFT(), createLockedNFT() - *onesself*, *futured*, *paramater*

Recommendation: fix the typos in the comments.

Found in: 05028ff

Status: Mitigated (Most of the reported typos are fixed. StreamingNFT: createNFT() - *paramater* is not fixed. New typos added in NatSpec comments. Consider using code spell checker extension.)

L14. Code Duplication

The *StreamingBoundHedgeys* contract duplicates the *StreamingHedgeys* contract except for the transferring functionality.

Path:
./contracts/StreamingNFT.sol

Recommendation: inherit the *StreamingHedgeys* contract in the *StreamingBoundHedgeys* contract and override the *_transfer* function.

Found in: 05028ff

Status: Fixed (Revised commit: d1a1434)

L15. Redundant Code

The *admin* is defined in the *StreamingBoundHedgeys* and *StreamingHedgeys* contracts` constructors, though it is defined in the *StreamingNFT* contract`s constructor, which is inherited by the mentioned contracts.

Therefore, this code is redundant.

Paths:
./contracts/StreamingBoundHedgeys.sol - constructor()
./contracts/StreamingHedgeys.sol - constructor()

Recommendation: remove the redundant code.

Found in: d1a1434

Status: Fixed (Revised commit: 59e62e7)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.