SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: StreamPay
Date: 31 May, 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 StreamPay
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	ERC777 token; Staking;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.streamablefinance.com/
Changelog	16.05.2023 - Initial Review 29.05.2023 - Second review 31.05.2023 - Third Review

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

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by StreamPay (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.

System Overview

The audit scope consists of a staking and reward distribution system. Users can deposit ERC20 LP tokens to various LP pools determined by the owner of the system. In return, they can acquire StreamableFinanceToken, which is an ERC777 standard token. The rewards are not directly sent to the user but are locked with a timelock. If the users wish to withdraw rewards before the timelock expires, they can do it with a penalty of 50%.

StreamPay is a staking protocol with the following contracts:

- *TokenStaker* – a contract that allows users to stake their tokens in pools to earn rewards. The rewards are distributed in STRF tokens, which are locked in a separate contract until the staking period is over. The contract also includes a schedule of future reward rates that depend on the elapsed time since the start of staking, as well as a mechanism for distributing the last stage of rewards at the end of the staking period.
- STRFTokenLocker – an STRF token locking contract that is used by the TokenStaker. Implementation includes timelock mechanism that handles reward distribution and penalties. TokenStaker contract is the LOCKER_ROLE of the STRFTokenLocker contract.
- ERC777Capped – an ERC-777 contract that is customly modified to cap the total supply. Initially no token is minted. Additional minting is allowed.
  It has the following attributes:
  - Name: given as a constructor parameter
  - Symbol: given as a constructor parameter
  - Decimals: 18
  - Total supply: given as a constructor parameter

Privileged roles

- The owner of the *TokenStaker* contract can:
  - set STRF Locker
  - add pools
  - set allocation points of pools
- The LOCKER_ROLE of the STRFTokenLocker contract can:
  - call the lock function
- The MINTER_ROLE of the ERC777Capped contract can:
  - mint tokens

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 **8** out of **10**.

- Functional requirements and technical description were provided.
- NatSpec format was not followed.
- The development environment instructions were provided.

Code quality

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

- The development environment was configured.
- The code is well-designed and follows best practices.

Test coverage

Code coverage of the project is **91.67%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions by several users are tested.

Security score

As a result of the audit, the code contains **1** low severity issue. The security score is **10** out of **10**.

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

Summary

According to the assessment, the Customer's smart contract has the following score: **9.5**. The system users should acknowledge all the risks summed up in the risks section of the report.

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
16 May 2023	5	3	2	1
29 May 2023	1	0	0	0

31 May 2023	1	0	0	0

Risks

- All privileged roles can affect the system; there is no documentation regarding how those roles will be protected or if multi-sig wallets will be used.

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	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	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	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Passed	
DoS (Denial of Service)	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	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Passed	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed	
Signature Unique Id	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	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	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	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

Flashloan Attack	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	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	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	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	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	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	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.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

■■■■ Critical

C01. Invalid Comparison

Impact	High
Likelihood	High

The *stakingStarted* modifier should check if the stake information is provided and the stake system has been started for the given pool. In the code, it is implemented incorrectly.

The default value of the *poolInfo[pid].startTime* is always zero, and when the pool has not been added and started yet, its value will be zero. Therefore, comparing it with block.timestamp is meaningless since the current time is always greater than zero.

It will allow users to execute pool transactions even though it has not yet been started.

Path: ./contracts/lp-token-staker/TokenStaker.sol : stakingStarted()

Recommendation: Add the require statement condition as below to the *stakingStarted* modifier.

_poolInfo[pid].startTime != 0

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

■■■ High

H01. Missing Validation

Impact	High
Likelihood	Medium

When adding a new pool, adding two pools of the same token can disrupt the logic of the contracts. This is stated as a warning in the *addPool*() function comment, but no prevention check is made.

This may result in unexpected behavior, as the total balance of the same token will be used as a divisor for multiple pool's reward calculations, resulting in lower rewards than expected.

Path: ./contracts/lp-token-staker/TokenStaker.sol : addPool()

Recommendation: Add a check to see if a token has already been added to a pool. This can be done by introducing a new mapping that tracks boolean values indicating whether tokens have been added to the pools or not.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

H02. Data Inconsistency

Impact	High
Likelihood	Medium

In the *setAllocPoint()* function, the allocation points are updated without updating the associated pool information simultaneously through the *_updatePool()* function.

Consequently, until the *_updatePool()* function is manually triggered by other functions, a period of time will elapse. During this time, the rewards generated will be calculated based on the new allocation points.

As a result of the allocation points being updated without updating the pool, the rewards between the "last reward time" and the "current time" are being manipulated.

This can lead to potential increases or decreases in reward amounts.

Path: ./contracts/lp-token-staker/TokenStaker.sol : setAllocPoints()

Recommendation: Update the pool right before changing the allocation points in the *setAllocPoint()* function.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

Medium

M01. Inefficient Gas Model

Impact	High
Likelihood	Medium

All token locks of a user are stored in an array and never removed, even after release.

The *withdraw()* and *withdrawExpiredLocks()* functions iterate over all the array elements. If the *_userLocks[user]* reaches a size big enough, transactions can revert due to exceeding Gas.

This leads to a situation where the Gas cost for calling the functions will constantly increase with each new lock.

Path: ./contracts/lp-token-staker/STRFTokenLocker.sol : withdraw(), withdrawExpiredLocks()

Recommendation: Consider removing the released lock elements from the locks array, or introducing an index mechanism that points to the last released lock element.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

M02. Undocumented Functionality

Impact	Low
Likelihood	High

The reason for manually updating the pool with a given time offset is not clear, as there is already a way to update it to the most current one.

The time offset functionality has no significant meaning in the implementation, so it is best to always set it to zero as a parameter in order to update the rewards to the latest block timestamp.

Path: ./contracts/lp-token-staker/TokenStaker.sol: manualUpdate()

Recommendation: Explain the logic behind this implementation or remove this redundant functionality.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f) (Customer stated that this function is used in case of a pool is not updated for a long time and trying to update with default offset would cause DOS. This way, the pool can be updated to its current state in a few transactions by using different offsets.)

M03. Undocumented Functionality

Impact	Medium
Likelihood	Medium

Calculations in the *_getPoolChanges()* function to update the pool info variables were not documented.

Complex function logic should be documented to ensure that the system runs as intended.

Path: ./contracts/lp-token-staker/TokenStaker.sol : _getPoolChanges()

Recommendation: Document all the important calculations.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

Low

L01. Missing Zero Address Validation

Impact	Low
Likelihood	Medium

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0 or can lead to saving information in mappings as 0x0.

Path: ./contracts/lp-token-staker/STRFTokenLocker.sol: constructor(), setTreasury(), lock()

Recommendation: Implement zero address checks.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Reported (*setTreasury()* and *lock()* functions have zero address validations implemented; however, the constructor is lacking zero address validation.)

L02. Check-Effects-Interaction Violation

Impact	Medium
Likelihood	Low

In the STRFTokenLocker.sol contracts *withdraw()* function, there is an external call made before modifying state variables.

The *_STRFToken.safeTransfer()* call made in the *if (!penaltyFlag)* if statement violates the Check-Effects_Interaction pattern and is against best practices.

This may result in reentrancy vulnerabilities and unexpected behavior.

Path: ./contracts/lp-token-staker/STRFTokenLocker.sol : withdraw()

Recommendation: Follow the Check-Effects_Interaction pattern by moving the penalty functionality logic after doing state changes, or use ReentrancyGuard.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

L03. Variable Shadowing

Impact	Low
Likelihood	Low

There is state variable shadowing in the constructor of the *StreamableFinanceToken*, all parameters *name*, *symbol*, *defaultOperators*, *cap* are shadowing the getter functions *name()*, *symbol()*, *defaultOperators()*, *cap()* from the child contracts.

Path: ./contracts/token/StreamableFinanceToken.sol : constructor()

Recommendation: Rename related parameter names.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

L04. Missing Events

Impact	Low
Likelihood	Low

The STRFTokenLocker contracts constructor is missing the *SetTreasury* event.

Events for critical state changes should be emitted for tracking things off-chain.

Path: ./contracts/lp-token-staker/STRFTokenLocker.sol : constructor

Recommendation: Emit related events.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

L05. Possible Denial Of Service

Impact	Medium
Likelihood	Low

In the *setSTRFLocker()* function, there is a potential vulnerability where the address zero can be set for the *_STRFLocker* variable.

This can result in a temporary Denial of Service (DoS) for users when they attempt to deposit or withdraw funds from the pool.

If a user already made a deposit and the *_STRFLocker* variable got set to address zero accidentally, the user will not be able to execute the *withdraw* function to get the deposited assets.

This may lead to temporarily locking the funds in the contract.

Path: ./contracts/token/TokenStaker.sol : setSTRFLocker()

Recommendation: Implement zero check for the setSTRFLocker function.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

Informational

I01. Redundant Contract & Function

The contract *Time* has only one function and it returns the block.timestamp. Block variables can be called directly.

Redundant declarations cause unnecessary Gas consumption and reduce the code readability.

Path: ./contracts/utils/Time.sol

Recommendation: Remove the contract from the project and bring block variables directly without using an intermediary contract.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Mitigated (Removing *Time.sol* causes many of the tests to fail.)

I02. Floating Pragma

The project uses floating pragma *^0.8.9*.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Path: ./contracts/token/extensions/ERC777Capped.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

I03. Explicit Size Of The Uint

In the for loops in the STRFTokenLockers contracts' the uint is used without explicit size.

It is best practice to explicitly state the size of the uint, for example, uint256.

Path: ./contracts/lp-token-staker/STRFTokenLocker.sol

Recommendation: Consider using uint with explicit size.

Found in: 999fc5e1034231fa1ad3362bf8d6c44da8b36088

Status: Fixed (Revised commit: 168e72f)

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.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	Medium
Medium Likelihood	**High**	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

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.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

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

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/streamable-finance/streampay-staking
Commit	999fc5e1034231fa1ad3362bf8d6c44da8b36088
Whitepaper	-
Requirements	-
Technical Requirements	-
Contracts	File: contracts/lp-token-staker/STRFTokenLocker.sol SHA3: c1173b9061e744c3ac8e7f569f22156468f77e81a542dd8963bdf1359e6488dc File: contracts/lp-token-staker/TokenStaker.sol SHA3: 2c7814653fd52bd22fcebc73d8b8b203ae88ed8de8b4df15a7af413bb24221dc File: contracts/token/StreamableFinanceToken.sol SHA3: a653c6389096fee9b055061cfd11e80b81ad5fde17bbef7bd491c25d6f0fe685 File: contracts/token/extensions/ERC777Capped.sol SHA3: ab1279c4f5369eafd50c3948c07a94d166962ecd281b5086db6eadbcfac18485 File: contracts/utils/Time.sol SHA3: a89ec8540dad9ceb058c4122fc4bfbdde8b1a8631c664565c56464e09bf925a7

Second review scope

Repository	https://github.com/streamable-finance/streampay-staking
Commit	168e72fb24689b92e18cd077c8f4943c911b12f5
Whitepaper	-
Requirements	LP staking security audit docs SHA3 : 5ef6211d2dcede9b494c2eb47d588109869ed2196f264e2e203e3a0b191e4ed0
Technical Requirements	LP staking security audit docs SHA3 : 5ef6211d2dcede9b494c2eb47d588109869ed2196f264e2e203e3a0b191e4ed0
Contracts	File: contracts/lp-token-staker/STRFTokenLocker.sol SHA3: 1b7e390df14ac17b8f9e2ec76e5a900c587f24c294fac518cf57d649093eefff File: contracts/lp-token-staker/TokenStaker.sol SHA3: 6c56ff7af75daf93a798cea31bbe4b0fd44ce6b97f9edd9b0d95dcfd525fa83c File: contracts/token/StreamableFinanceToken.sol SHA3: 4c880e77c80eeabf2d7521ffa2d42c8b9081ea29172853dacb39d9f98dc6337e

	File: contracts/token/extensions/ERC777Capped.sol SHA3: 76b892abe7840989680ab129a2f197dc6ccb4ae35657be5ded1302e85cdc5112

Third review scope

Repository	https://github.com/streamable-finance/streampay-staking
Commit	c5b6f12ede90352047a87d960b0318e0a94d714a
Whitepaper	-
Requirements	LP staking security audit docs SHA3 : 5ef6211d2dcede9b494c2eb47d588109869ed2196f264e2e203e3a0b191e4ed0
Technical Requirements	LP staking security audit docs SHA3 : 5ef6211d2dcede9b494c2eb47d588109869ed2196f264e2e203e3a0b191e4ed0
Contracts	File: contracts/lp-token-staker/STRFTokenLocker.sol SHA3: 53c64d65e25ae021a73e9fbf64756d255e239d292034faa4f9bd7de9ec8a2ff2 File: contracts/lp-token-staker/TokenStaker.sol SHA3: 75fea3b33acc1e3653471fae1229f919d8cdceee28bd7e1296e349a35f84ae89 File: contracts/token/StreamableFinanceToken.sol SHA3: 4c880e77c80eeabf2d7521ffa2d42c8b9081ea29172853dacb39d9f98dc6337e File: contracts/token/extensions/ERC777Capped.sol SHA3: 76b892abe7840989680ab129a2f197dc6ccb4ae35657be5ded1302e85cdc5112