verichains

SECURITY AUDIT OF

HEROFI SMART CONTRACTS

Public Report

Dec 09, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Dec 09, 2021. We would like to thank the HeroFi for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the HeroFi Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified one vulnerable issue in the contract code. HeroFi team has resolved and updated all the recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About HeroFi Smart Contracts

HeroFi is a mobile aRPG game in which players can earn tokens through PvP/PvE battles between Heroes.

Each Hero is unique and equally accessible to anyone. There is no initial investment barrier in HeroFi. HeroFi is completely free to play, and truly play to earn. The gap between normal games and NFT games is eliminated. HeroFi allows players to collect and trade NFT heroes. Then for the first time, heroes can get married and give birth to a new NFT hero.

This game has "Free To Earn" mechanisms so that users can invest and play or play for free

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the token vesting and staking part of the HeroFi Smart Contracts. It was conducted on the source code provided by the HeroFi team.

1.3. ROFIToken contract

The token contract in the HeroFi Smart Contracts deployed on Binance Smart Chain Mainnet at address 0x3244b3b6030f374bafa5f8f80ec2f06aaf104b64. The details of the deployed smart contract are listed in Table 1.

FIELD	VALUE
Contract Name	ROFIToken
Contract Address	0x3244b3b6030f374bafa5f8f80ec2f06aaf104b64
Compiler Version	v0.6.12+commit.27d51765
Optimization Enabled	Yes with 200 runs
Explorer	*https://bscscan.com/address/0x3244b3b6030f374bafa5f8f80ec2f06aaf104b64*

Table 1. The deployed smart contract details

1.4. Reward contract

The reward contract (*reward.sol*) was conducted on commit 1a3dcbfffe8117aefc29e0d1b8e13a21ad23aced from git repository *https://github.com/HeroFi/smart-contracts*.

1.5. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.

SEVERITY LEVEL	DESCRIPTION
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 2. Severity levels

1.6. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

2.1.1. Token contract

This table lists some properties of the audited HeroFi Smart Contracts (as of the report writing time).

PROPERTY	VALUE
Name	HeroFi
Symbol	ROFI
Decimals	18
Max Supply	100,000,000 ($x10^{18}$) Note: the number of decimals is 18, so the total representation token will be 100,000,000 or 100 million.

Table 3. The HeroFi Smart Contracts properties

ROFIToken contract extends BEP20, Pausable and ReentrancyGuard contracts. With Ownable, by default, Token Owner is contract deployer but he can transfer ownership to another address at any time.

Token Owner can pause/unpause contract using Pausable contract, user can only transfer unlocked tokens and only when contract is not paused.

The contract supports mint unlocked/locked tokens which can only be used by one who have permission. Token Owner can use setAuthorizedUnlockedMintCaller and setAuthorizedLockedMintCaller to allow addresses for minting unlocked and locked tokens respectively but the amount of minted tokens can not pass max supply. It also supports governance by calculating/delegating voting balance of users.

Users can stake/unstake tokens to earn unlocked reward tokens. They can also burn their unlocked tokens by calling burn.

2.1.2. Reward contract

This is the reward contract in the HeroFi Smart Contracts, which extends Ownable abstract contract. With Ownable, by default, Token Owner is contract deployer but he can transfer ownership to another address at any time.

It provides claimReward and claimRewards functions which users can call to claim reward ROFI tokens (divided to locked and unlocked by lockedPercentage) with a valid proof from the backend server. The proof is verified (user address with reward amount) by Merkle Proof so no one can claim reward without permission from owner.

Token Owner (contract deployer) can updateMerkleRoot which will make old _proof for claiming rewards become invalid. He can also change lockedPercentage, which is 80 by default, to change amount of locked/unlocked tokens in reward and update reward token address in case of changing ROFI token contract.

2.2. Findings

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

2.2.1. Reward.sol - Zero locked amount when mintReward with very small _reward LOW

In the mintReward function, the locked amount is calculated by a percentage with _reward.div(100).mul(lockedPercentage) but solidity does not support floating point number so if _reward is less than 100, locked will become zero instead of lockedPercentage.

```
function mintReward(address _to, uint256 _reward) internal {
    ...
    uint256 locked = uint256(_reward.div(100).mul(lockedPercentage));
    ...
}
```

RECOMMENDATION

Always multiple first when calculate percentage amount.

```
function mintReward(address _to, uint256 _reward) internal {
    ...
    uint256 locked = uint256(_reward.mul(lockedPercentage).div(100));
    ...
}
```

UPDATES

- *Dec 09, 2021*: This recommendation has been acknowledged by the HeroFi team.

2.2.2. Reward.sol - Use calldata instead of memory for gas saving INFORMATIVE

In external function with array arguments, using memory will force solidity to copy that array to memory thus wasting more gas than using directly from calldata. Unless you want to write to the variable, always using calldata for external function.

```
function claimReward(uint256 _timestamp, address _user, uint256 _reward, …
  bytes32[] memory _proof) external
function claimRewards(uint256[] memory _timestamps, address _user, uint25…
  6[] memory _rewards, bytes32[][] memory _proofs) public
...
```

RECOMMENDATION

Change memory to calldata for gas saving in all external functions.

```
function claimReward(uint256 _timestamp, address _user, uint256 _reward, …
  bytes32[] calldata _proof) external
function claimRewards(uint256[] memory _timestamps, address _user, uint25…
  6[] calldata _rewards, bytes32[][] memory _proofs) external
...
```

UPDATES

- *Dec 09, 2021*: This recommendation has been acknowledged by the HeroFi team.

2.2.3. Reward.sol - Unused claimLimit storage variable INFORMATIVE

```
uint256 public claimLimit;
```

RECOMMENDATION

Remove unused variable.

UPDATES

- *Dec 09, 2021*: This recommendation has been acknowledged by the HeroFi team.

APPENDIX

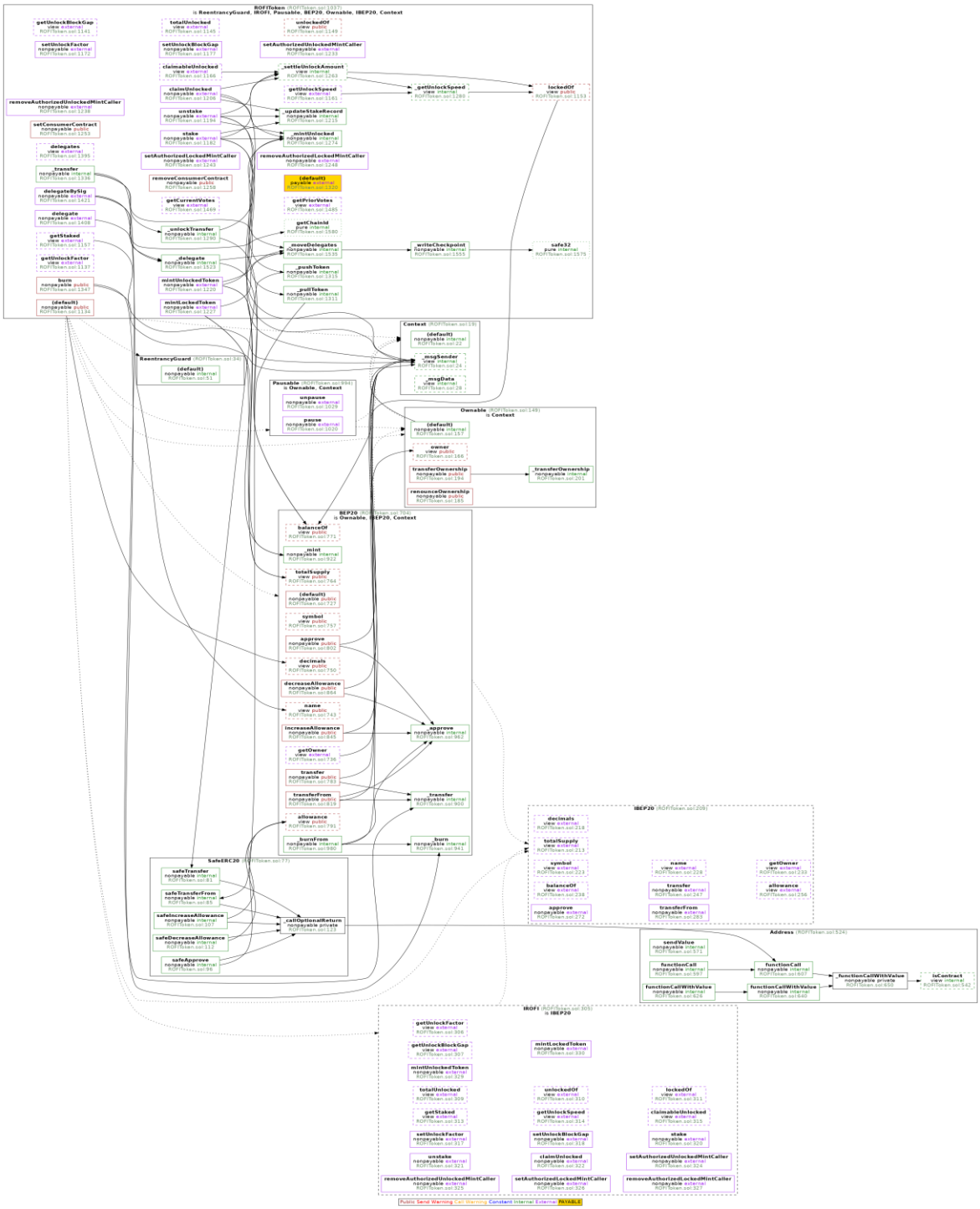

Image 1. ROFI Token call graph

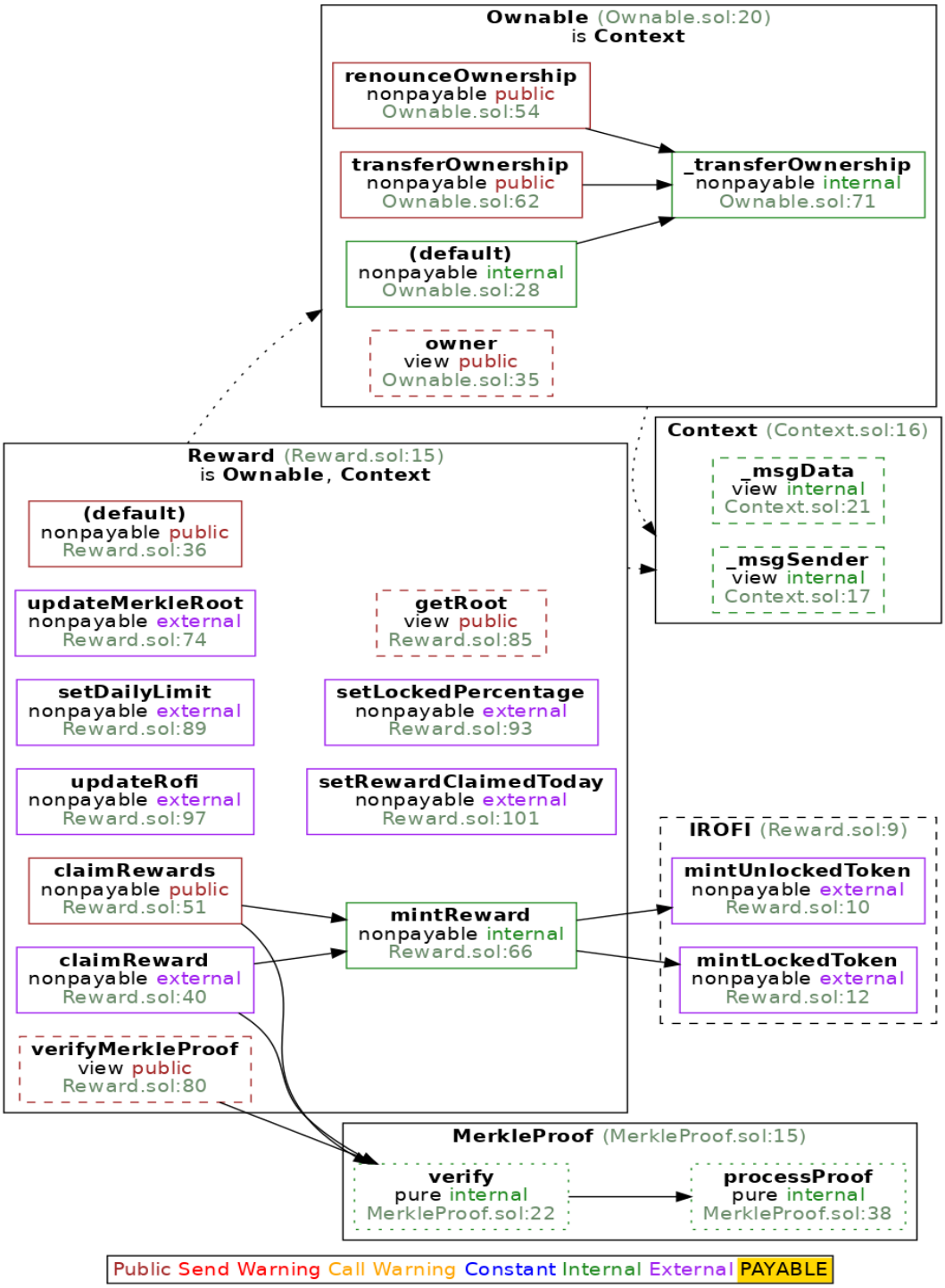

Image 2. Reward call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 09, 2021*	Public Report	Verichains Lab
1.1	*Dec 09, 2021*	Public Report	Verichains Lab

Table 4. Report versions history