verichains

SECURITY AUDIT OF

HEROFIEGG TOKEN

SMART CONTRACT

PUBLIC REPORT

September 16, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ACRONYMS AND 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.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Sep 16, 2021. We would like to thank the HeroFi team 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 HeroFiEgg Contract. 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.

The assessment did not identify any vulnerability issue in HeroFiEgg smart contract code.

Overall, the code reviewed is of good quality, written with the awareness of smart contract development best practices.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About HeroFi and HeroFiEgg

HeroFi is an aRPG game based on blockchain technology. In the game, players control their hero to join missons for reward tokens.

HeroFiEgg (HEROEGG) is the BEP-20 token of HeroFi, with an initial total supply of 520 million.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the HeroFiEgg Contract.

The audited contract is the HeroFiEgg Contract that was deployed on Binance Smart Chain Mainnet at address *0xcfBb1BfA710cb2ebA070CC3beC0C35226FeA4BAF*. The details of the deployed smart contract are listed in Table 1.

FIELD	VALUE
Contract Name	HeroFiEgg
Contract Address	0xcfBb1BfA710cb2ebA070CC3beC0C35226FeA4BAF
Compiler Version	v0.6.12+commit.27d51765
Optimization Enabled	No with 200 runs
Explorer	https://bscscan.com/address/0xcfBb1BfA710cb2ebA070CC3beC0C35226FeA4BAF

Table 1: The deployed smart contract details

1.3. 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 2, 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.
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: Vulnerability severity levels

1.4. 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

The HeroFiEgg Contract is an BEP-20 Token Contract[1], which implements a standard interface for token as defined in IBEP-20[2]. This standard provides basic functionality to transfer tokens, as well as allow tokens to be approved so they can be spent by another on-chain third party.

Table 3 lists some properties of the audited HeroFiEgg Contract (as of the report writing time).

PROPERTY	VALUE
Name	HeroFiEgg
Symbol	HEROEGG
Decimals	18
Owner	0xe0f0C59a6dD34c9EBDDcF9E2Ab7c04b58161dC8E
Total Supply	520,000,000 ($\times 10^{18}$) Note: the number of decimals is 18, so the total representation tokens will be 520,000,000, or 520 million.

Table 3: The HeroFiEgg Token properties

Besides the standard interface of an BEP-20 token, the HeroFiEgg contract also implements some additional functional logics by extending the following contracts:

- Context: provides information about the current execution context, including the sender of the transaction and its data. While these are generally available via msg.sender and msg.data, they should not be accessed in such a direct manner, since when dealing with GSN meta-transactions the account sending and paying for execution may not be the actual sender (as far as an application is concerned).
- Ownable: this contract has a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions. By default, the owner account will be the one that deploys the contract. The current owner can renounce or transfer ownership to a new owner.

[1] https://docs.binance.org/smart-chain/developer/BEP20.html
[2] https://docs.binance.org/smart-chain/developer/IBEP20.sol

- BEP20: this contract implements all function declared in interface IBEP20 which inherited by HeroFiEgg contract.

2.2. Contract codes

The HeroFiEgg Contract was written in Solidity language[3], with the required version to be 0.6.12.

The source codes consist of four contracts, one interface and three libraries. Almost all source codes in the HeroFiEgg Contract are imported from Binance's implementation template of BEP20-related contracts.

2.2.1. Context contract

This contract provides information about the current execution context, including the sender of the transaction and its data. The source code is referenced from OpenZeppelin's implementation.

2.2.2. SafeERC20 library

This library provides wrappers around ERC20 operations. The source code is referenced from OpenZeppelin's implementation.

2.2.3. Ownable contract

This contract makes the HeroFiEgg Contract ownable. The source code is referenced from OpenZeppelin's implementation.

2.2.4. IBEP20 interface

This is the interface of the BEP-20 token standard. The source code is referenced from the official Binance's documentation.

2.2.5. SafeMath library

This library provides wrappers over Solidity's arithmetic operations with added overflow checks. The source code is referenced from OpenZeppelin's implementation.

2.2.6. Address library

This library provides the collection of function related to the address tupe.

[3] https://docs.soliditylang.org/en/latest

2.2.7. BEP20 contract

This is the contract implement almost important functions in the HeroFiEgg contract which extends the *IBEP20*, *Context* and *Ownable* contracts. Below is a summary of some important functions in this contract:

- *constructor()*: constructor set the name, symbol, decimal for the contract.
- *getOwner()*: returns the bep token owner.
- *name()*: returns the token name.
- *decimas()*: returns the decimals.
- *symbol()*: returns the token symbol.
- *totalSupply():* returns the total supply.
- *burnAmount():* returns value of the burn address.
- *balanceOf(address account):* returns the balance of the input account.
- *transfer(address recipient, uint256 amount):* transfers `amount` tokens from the sender to `recipient`.
- *allowance(address owner, address spender):* returns the `amount` which `spender` is still allowed to withdraw from `owner`.
- *approve(address spender, uint256 amount):* allows `spender` to withdraw from the caller's account multiple times, up to the `amount`. If this function is called again it overwrites the current allowance with `amount`.
- *transferFrom(address sender, address recipient, uint256 amount):* transfers `amount` of tokens from address `sender` to address `recipient`.
- *increaseAllowance(address spender, uint256 addedValue):* increases the allowance amount for `spender` by `addedValue`.
- *decreaseAllowance(address spender, uint256 subtractedValue):* decreases the allowance amount for `spender` by `subtractedValue`.

2.2.8. HeroFiEgg contract

This is the main contract, which extends the BEP20. The contract constructor set name, symbol value of the contract.

2.3. Findings

During the audit process, the audit team did not discover any security vulnerability issue in the HeroFiEgg Token Contract.

2.4. Additional notes and recommendations

2.4.1. Missing checking the burn address in _mint functions

Address `0xdEaD` is used as the burn address in the contract, but internal functions `_mint` in contract act if the burn address is `0x0`. It will become an issue in the future when the contract implements a public function which calls `_mint` function.

```
668  function totalSupply() public override view returns (uint256) {
669      return _totalSupply.sub(burntAmount());
670  }
671
672  function burntAmount() public override view returns (uint256) {
673      return this.balanceOf(address(0xdEaD));
674  }
```

```
830  function _mint(address account, uint256 amount) internal {
831      require(account != address(0), 'BEP20: mint to the zero address');
832
833      _totalSupply = _totalSupply.add(amount);
834      _balances[account] = _balances[account].add(amount);
835      emit Transfer(address(0), account, amount);
836  }
```

RECOMMENDATION

Update `_mint` function as in the following code:

```
830  function _mint(address account, uint256 amount) internal {
...      require(account != address(0), 'BEP20: mint to the zero address');

         //check the burn addresses
         require(account != address(0xdEaD), 'BEP20: mint to the burn address');
         _totalSupply = _totalSupply.add(amount);
         _balances[account] = _balances[account].add(amount);
         emit Transfer(address(0), account, amount);
     }
```

2.4.2. Outdated version of Solidity

The required version of Solidity in current HeroFiEgg contract source code is `0.6.12`, which is quite outdated. We recommend updating to `0.8.7` (which is the latest Solidity version at the writing time).

RECOMMENDATION

Update the Solidity version to `0.8.7` using `pragma solidity 0.8.7.`

2.4.3. Unused _burn function

The `_burn` internal function is unused in this contract.

RECOMMENDATION

Remove the *_burn* function.

2.4.4. Unused SafeERC20 library

The library `SafeERC20` is unused in this contract.

RECOMMENDATION

Remove the `SafeERC20` library.

3. VERSION HISTORY

VERSION	DATE	STATUS/CHANGES	CREATED BY
1.0	Sep 16, 2021	Initial private report	Verichains Lab
1.1	Sep 16, 2021	Public report	Verichains Lab

APPENDIX A: FUNCTION CALL GRAPH

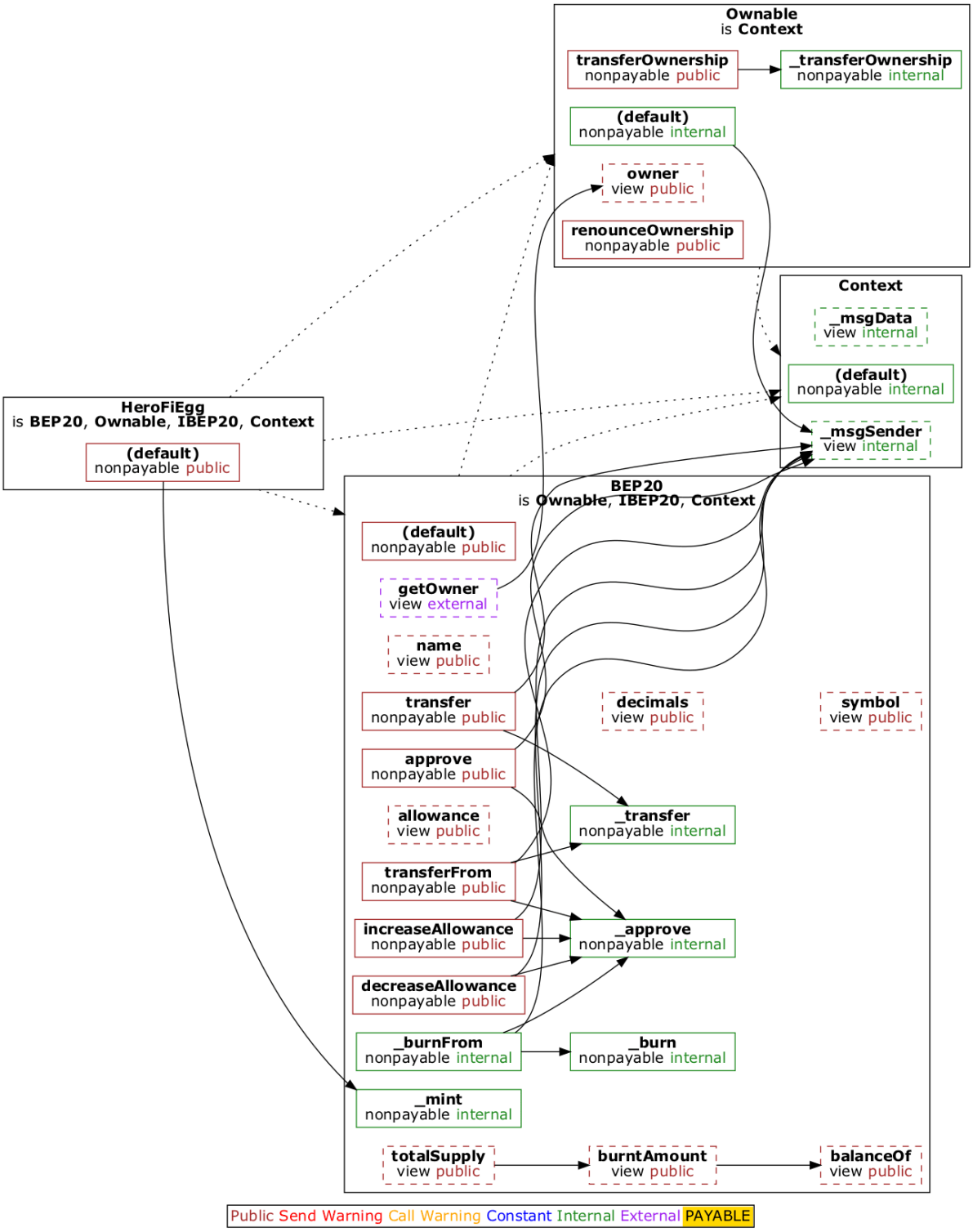

Figure 1: The function call graph of HeroFiEgg smart contract