verichains

SECURITY AUDIT OF

HOWL SMART CONTRACT

Public Report

Oct 16, 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 Oct 16, 2021. We would like to thank the HowlCity 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 HOWL Smart 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.

During the audit process, the audit team had identified no vulnerable issue in the application.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About HOWL Smart Contract

HowlCity is a metaverse blockchain world where everyone competes in racing but collaborates in life.

1.2. Audit scope

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

The audited contract is the HOWL Smart Contract that deployed on Binance Smart Chain Mainnet at address 0x549CC5dF432cdbAEbc8B9158A6bDFe1cbD0Ba16d. The details of the deployed smart contract are listed in Table 1.

FIELD	VALUE
Contract Name	HOWL
Contract Address	0x549CC5dF432cdbAEbc8B9158A6bDFe1cbD0Ba16d
Compiler Version	v0.8.0+commit.c7dfd78e
Optimization Enabled	No with 200 runs
Explorer	https://bscscan.com/address/0x549cc5df432cdbaebc8b9158a6bdfe1cbd0ba16d

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 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.
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

1.4. 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 HOWL Smart Contract is an ERC-20 Token Contract, which implements a standard interface for token as defined in IERC-20. 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 HOWL Smart Contract (as of the report writing time).

PROPERTY	VALUE
Name	HOWL
Symbol	HWL
Decimals	18
Owner	No with 200 runs
Total Supply	540,000,000 ($x10^{18}$) Note: the number of decimals is 18, so the total representation token will be 540,000,000 or 540 million.

Table 3. The HOWL Smart Contract properties

Besides the standard interface of an ERC-20 token, the HOWL Smart Contract also implement some additional functional logic 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.

2.2. Contract codes

The HOWL Smart Contract was written in Solidity language, with the required version to be 0.8.0.

The source codes consist of two contracts, four abstract contracts, two interfaces. Almost all source codes in the HOWL Smart Contract imported OpenZeppelin contracts.

2.2.1. IERC20 interface

This is the interface of the ERC-20 token standard. The source code is referenced from OpenZeppelin's implementation.

2.2.2. IERC20Metadata interface

This is the interface that extends IERC20 interface to add functions for interacting with some metadata. The source code is referenced from OpenZeppelin's implementation.

2.2.3. Context abstract contract

This abstract 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.

2.2.4. Ownable abstract contract

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

2.2.5. ERC20 contract

This is the abstract contract implement ERC20 token. The source code is referenced from OpenZeppelin's implementation.

2.2.6. Pausable abstract contract

Contract module allows children to implement an emergency stop mechanism that can be triggered by an authorized account. The source code is referenced from OpenZeppelin's implementation.

2.2.7. ERC20Burnable abstract contract

This abstract contract extends ERC20 abstract contract that allows token holders to destroy their tokens. The source code is referenced from OpenZeppelin's implementation.

2.2.8. HOWL contract

This is the main contract in the HOWL Smart Contract, which extends ERC20, ERC20Burnable, Pausable, Ownable abstract contract. Almost functions in the HOWL Smart Contract are inherited, some functions are overridden by composing between the old function and the modifiers.

2.3. Findings

During the audit process, the audit team found no vulnerability in the given version of HOWL Smart Contract.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 16, 2021*	Public Report	Verichains Lab

Table 4. Report versions history