SECURITY AUDIT OF

THEMONOPOLIST TOKEN SMART CONTRACT

PUBLIC REPORT

OCT 13, 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
Binance Chain	Binance Chain is a blockchain software system developed by Binance and its community.
Binance Smart Chain (BSC)	Binance Smart Chain (BSC) is a blockchain network built for running smart contract-based applications. BSC runs in parallel with Binance's native Binance Chain (BC), which allows users to get the best of both worlds: the high transaction capacity of BC and the smart contract functionality of BSC.
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 Oct 13, 2021. We would like to thank the TheMonopolist 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 TheMonopolist Token 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 TheMonopolist Token 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 The Monopolist and TheMonopolist Token

In The Monopolist, the players will build their own tactics through rolling dice, investing in buying land, building properties, collecting accommodation fees, and so on, to win the others by various ways.

TheMonopolist Token (MONO) is the ERC-20 token of TheMonopolist, with an initial total supply of 1 billion.

1.2. Audit scope

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

The audited contract is the TheMonopolist Token Contract that was deployed on Binance Smart Chain Mainnet at address `0xD4099A517f2Fbe8a730d2ECaad1D0824B75e084a`. The details of the deployed smart contract are listed in Table 1.

FIELD	VALUE
Contract Name	TheMonopolist
Contract Address	0xD4099A517f2Fbe8a730d2ECaad1D0824B75e084a
Compiler Version	v0.8.0+commit.c7dfd78e
Optimization Enabled	No with 200 runs
Explorer	https://bscscan.com/address/0xD4099A517f2Fbe8a730d2ECaad1D0824B75e084a

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

TheMonopolist 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 TheMonopolist Token Contract (as of the report writing time).

PROPERTY	VALUE
Name	TheMonopolist
Symbol	MONO
Decimals	18
Owner	0x1696904cB82a76D7e6d227bB276db06B1187aa12
Total Supply	1,000,000,000 (× 10^{18}) Note: the number of decimals is 18, so the total representation tokens will be 1,000,000,000, or 1 billion.

Table 3: TheMonopolist Token properties

2.2. Contract codes

The TheMonopolist Token Contract was written in Solidity language[1], with the required version to be `0.8.0`.

The source codes consist of three contracts, two abstract contracts, three interfaces. Almost all source codes in the TheMonopolist Token Contract are imported OpenZeppelin Contracts[2].

2.2.1. IERC20 interface

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

[1] https://docs.soliditylang.org/

[2] https://openzeppelin.com/contracts/

2.2.2. IERC20Metadata interface

This is the interface 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 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 contract has a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions. The source code is referenced from OpenZeppelin’s implementation.

2.2.5. IBP interface

This is the interface which support BotPreventable contract.

2.2.6. BotPreventable contract

This is a contract responsible for preventing bot when the main contract working. It calls external function from another contract through IBP interface.

2.2.7. ERC20 contract

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

2.2.8. TheMonopolist contract

This is the main contract, which extends the ERC20 and BotPreventable contract. The contract override *_transfer* internal function to prevent bot from transferring token.

2.3. Findings

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

2.4. Additional notes and recommedations

2.4.1. The modifier *preventTransfer*

The modifier calls *protect* external function from another contract. Since we do not control the logic of that contract, there is no guarantee that the called contract will not contain any security related issues. With the current context, in case the called contract is compromised, there is not yet a way to exploit the TheMonopolist contract, but we will note that here as a warning for avoiding any related issues in the future.

By the way, if having any issue, the called contract can be easily unused anytime by contract `owner` using the *`setBotPreventEnabled`* function.

3. VERSION HISTORY

Version	Date	Status/Changes	Created by
1.0	Oct 11, 2021	Initial private report	Verichains Lab
1.1	Oct 13, 2021	Public Report	Verichains Lab

APPENDIX A: FUNCTION CALL GRAPH

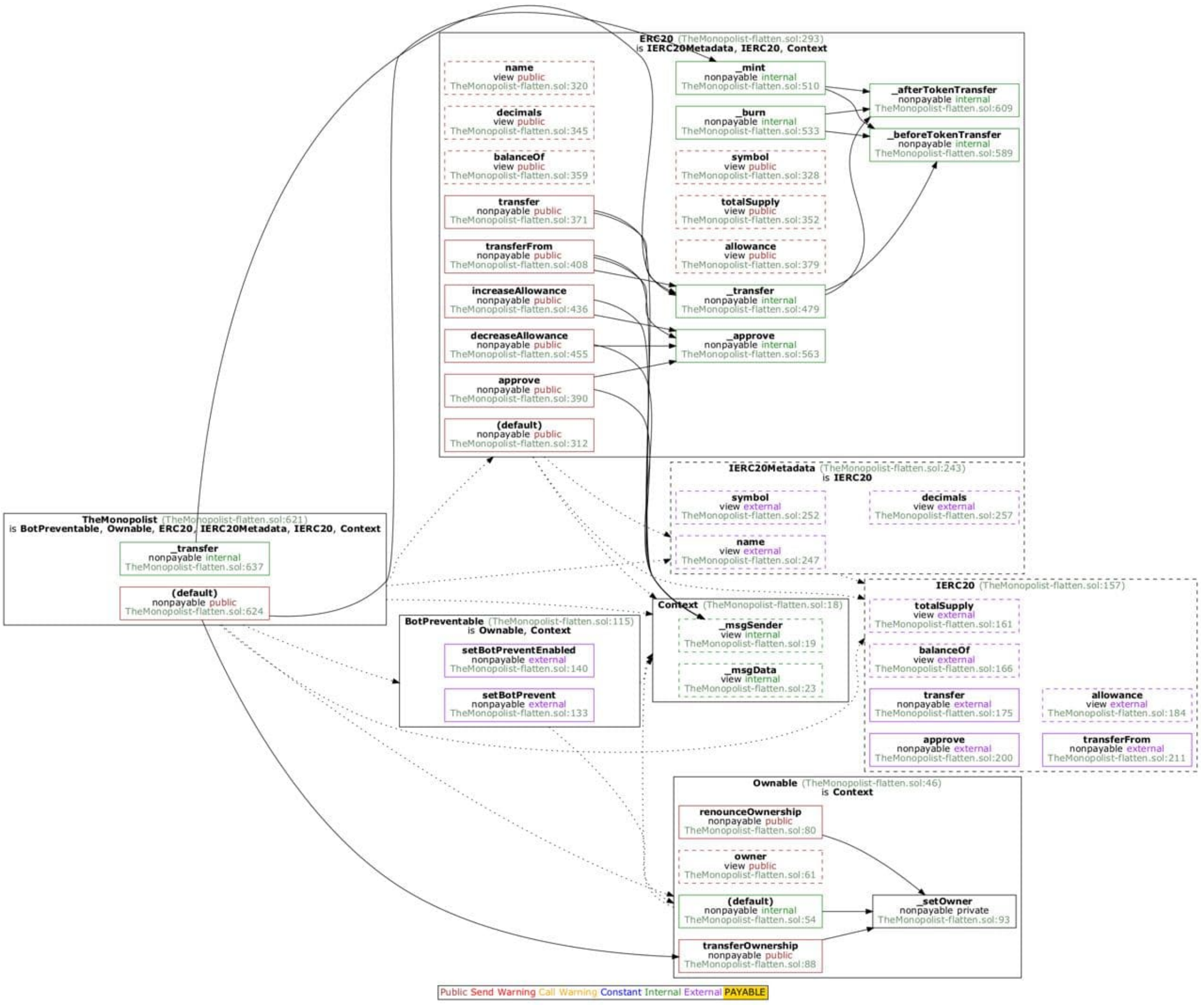
ERC20 (TheMonopolist-flatten.sol:293) is IERC20Metadata, IERC20, Context
name view public TheMonopolist-flatten.sol:320
decimals view public TheMonopolist-flatten.sol:345
balanceOf view public TheMonopolist-flatten.sol:359
transfer nonpayable public TheMonopolist-flatten.sol:371
transferFrom nonpayable public TheMonopolist-flatten.sol:408
increaseAllowance nonpayable public TheMonopolist-flatten.sol:436
decreaseAllowance nonpayable public TheMonopolist-flatten.sol:455
approve nonpayable public TheMonopolist-flatten.sol:390
(default) nonpayable public TheMonopolist-flatten.sol:312
_mint nonpayable internal TheMonopolist-flatten.sol:510
_burn nonpayable internal TheMonopolist-flatten.sol:533
symbol view public TheMonopolist-flatten.sol:328
totalSupply view public TheMonopolist-flatten.sol:352
allowance view public TheMonopolist-flatten.sol:379
_transfer nonpayable internal TheMonopolist-flatten.sol:479
_approve nonpayable internal TheMonopolist-flatten.sol:563
_afterTokenTransfer nonpayable internal TheMonopolist-flatten.sol:609
_beforeTokenTransfer nonpayable internal TheMonopolist-flatten.sol:589
IERC20Metadata (TheMonopolist-flatten.sol:243) is IERC20
symbol view external TheMonopolist-flatten.sol:252
decimals view external TheMonopolist-flatten.sol:257
name view external TheMonopolist-flatten.sol:247
TheMonopolist (TheMonopolist-flatten.sol:621) is BotPreventable, Ownable, ERC20, IERC20Metadata, IERC20, Context
_transfer nonpayable internal TheMonopolist-flatten.sol:637
(default) nonpayable public TheMonopolist-flatten.sol:624
BotPreventable (TheMonopolist-flatten.sol:115) is Ownable, Context
setBotPreventEnabled nonpayable external TheMonopolist-flatten.sol:140
setBotPrevent nonpayable external TheMonopolist-flatten.sol:133
Context (TheMonopolist-flatten.sol:18)
_msgSender view internal TheMonopolist-flatten.sol:19
_msgData view internal TheMonopolist-flatten.sol:23
IERC20 (TheMonopolist-flatten.sol:157)
totalSupply view external TheMonopolist-flatten.sol:161
balanceOf view external TheMonopolist-flatten.sol:166
transfer nonpayable external TheMonopolist-flatten.sol:175
allowance view external TheMonopolist-flatten.sol:184
approve nonpayable external TheMonopolist-flatten.sol:200
transferFrom nonpayable external TheMonopolist-flatten.sol:211
Ownable (TheMonopolist-flatten.sol:46) is Context
renounceOwnership nonpayable public TheMonopolist-flatten.sol:80
owner view public TheMonopolist-flatten.sol:61
(default) nonpayable internal TheMonopolist-flatten.sol:54
transferOwnership nonpayable public TheMonopolist-flatten.sol:88
_setOwner nonpayable private TheMonopolist-flatten.sol:93
Public Send Warning Call Warning Constant Internal External PAYABLE

Figure 1: The function call graph of TheMonopolist Token smart contract