verichains

SECURITY AUDIT OF

FORMACAR ACTION TOKEN SMART CONTRACT

FORMACAR ACTION

Public Report

Dec 29, 2022

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 was prepared by Verichains Lab on Dec 29, 2022. We would like to thank the Formacar Action 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 Formacar Action Token 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 some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY 5

1.1. About Formacar Action Token Smart Contract 5

1.2. Audit scope 5

1.3. Audit methodology 6

1.4. Disclaimer 7

2. AUDIT RESULT 8

2.1. Overview 8

2.1.1. FormarCarGame contract 8

2.2. Findings 10

2.2.1. `AllowLaunchMarket` quantity confirm is not matched document MEDIUM 10

2.2.2. `feeReceiver` can miss the `LowMarketFeeTemps` LOW 11

2.2.3. Gas optimize INFORMATIVE 12

3. VERSION HISTORY 13

1. MANAGEMENT SUMMARY

1.1. About Formacar Action Token Smart Contract

Formacar is a global car-themed IT project with a deeply developed proprietary architecture that incorporates a number of B2B and B2C online services.

The goal of the project is to develop an ecosystem that will bring together all kinds of car-related services and make them available to private users and businesses worldwide. The mission of the project is to unite people around the automotive world, to educate audiences, and to provide affordable yet high-quality technologies on goods, services and entertainment markets.

The key advantage of Formacar is eight years of experience developing proprietary online-based services with the implementation of breakthrough 3D and AR technologies. For Formacar, no geographical barriers exist that would prevent it from promoting its goods and services and concluding deals, because the project has every bit of potential needed to evolve to international status.

1.2. Audit scope

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

The audited contract is the Formacar Action Token Smart Contract that deployed on Binance Smart Chain Mainnet at address `0x3991ffab22c745143aa67d3e88fbcb7b2c539445`. The details of the deployed smart contract are listed in Table 1.

FIELD	VALUE
Contract Name	FormaCarGame
Contract Address	0x3991ffab22c745143aa67d3e88fbcb7b2c539445
Compiler Version	v0.8.9+commit.e5eed63a
Optimization Enabled	Yes with 200 runs
Explorer	*https://bscscan.com/address/0x3991ffab22c745143aa67d3e88fbcb7b2c539445*

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 were 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

2.1.1. FormarCarGame contract

The Formacar Action Token Smart Contract was written in `Solidity` language, with the required version to be `^0.8.9`.

FormaCarGame is an ERC20 token with additional functionality. The contract can be divided into 3 functional modules:

- ERC20 standard token;
- Mechanics of commission withdrawal and control of token trading on DEXs;
- Government-style contract administration and management system.

ERC20 standard token

The contract fully inherits the functionality of the standard. The mechanics of financial module is implemented by rewriting the inherited internal method `_transfer` of this standard.

Table 2 lists some properties of the audited Formacar Action Token Smart Contract (as of the report writing time).

PROPERTY	VALUE
Name	FormaCarGame
Symbol	FCG
Decimals	18
Total Supply	1,000,000,000 (x10^{18}) Note: the number of decimals is 18, so the total representation token will be 1,000,000,000 or 1 billion.

Table 3. The Formacar Action Token Smart Contract properties

Mechanics of commission withdrawal sand control of token trading on DEXs

There are 4 anti mechanics was applied in `_transfer` function:

- Antibot mechanics: Upon the trading start event, the market activates the anti-bot mode, in which an increased commission is imposed on the sale of the token to this market - the standard commission of the market for sale gets multiplied by 20. This mode lasts 1 hour from the start of sales. This also applies to the fee whitelist.

- Antisniper mechanics: For this mechanism, a record is kept of all traders (whom the system identified as such during the transfer). Trader data structures linked to the trader's address are entered into the corresponding mapping. When a trader, not being in monitoring mode, makes a purchase, the monitoring mode is activated, and the next 60 seconds are simply counted all of his purchases. If after these 60 seconds he made 5 or more purchases, then for the next 300 seconds all his trades are rejected. After these 300 seconds, or if there are not 5 purchases during the monitoring period, a new monitoring period starts with the next purchase. Antisnipe works globally, regardless of which market the trader makes the next action on.
- Antiwhale mechanics: Each market maintains its own statistics on the volume of purchases and sales over the past day. When buying or selling on a particular market, a check is made to ensure that the transaction volume does not exceed the bar of 3% (whaleRatePercents parameter) of the daily volume of purchases or sales (respectively) on this market, otherwise the transaction is rejected. These limits are in the form of buyWhaleLimit and sellWhaleLimit for each market, and are assigned automatically when updating statistics periodization every 6 hours.
- Antidump mechanics: The contract maintains global statistics on the volume of purchases and sales for the last hour in total for all markets. If the volume of sales exceeds the volume of purchases by 75% (dumpThresholdPercents parameter), then the anti-dump mode is activated for 1 hour (dumpDurationSeconds parameter), during which, when selling, a value of 60% of the transaction volume is added to the commission (parameter dumpFeePercents) . The antidump is subject to the fee whitelist (isFeeExcluded). If after this hour the situation has not improved, the anti-dump is turned on again.

Government system mechanics

Within the framework of the government decision management system in the $FCG token system, there are two main objects of rights control and application execution. Application - any action to change the dynamic components of the settings in the token management system.

The government consists of two types of participating persons: workers and validators.

- Workers (workers) are persons who create applications for calling certain methods with certain input parameters (arguments). Each worker has an access level - a parameter that must correspond (be no lower) to the level of importance of the method for which the request is created. Initially, 5 workers with levels 3,3,2,1,1 are introduced into the system. There can be an arbitrary number of any workers, the only limitation is that there cannot be less than two higher workers (with level 3 and above).
- Validators are persons involved in the confirmation of applications created by workers. An application is executed when the required number of validators have signed it.

Validators are no different from each other, there can be any number of them, but not less than five. Initially, 5 validators are entered into the system.

2.2. Findings

During the audit process, the audit team found some issues in the given version of Formacar Action Token Smart Contract.

2.2.1. `AllowLaunchMarket` quantity confirm is not matched document MEDIUM

The number of quantity confirm in document is `2`, but in the source code, it is defined by `3`.

12	Add Market (pair , force)	registers an already created AMM in the accounting system	2	2
13	AllowLaunchMarket (pair)	sets the market flag launchAllowed = true. After that, the worker can pour liquidity into this AMM in the required proportions.	2	2
14	RemoveMarket (pair)	method of removing the market from the accounting system	2	3

```
// FormaCarGame.sol
function _getActionApproveCount(uint actionId) internal pure virtual override returns(uint)
{
    if (actionId == 3) return 1; // SetFeeExcluded/Many
    if (actionId == 4) return 1; // SetWhaleExcluded/Many
    if (actionId == 5) return 1; // SetSnipeExcluded/Many
    if (actionId == 6) return 1; // ExcludeFromAll
    if (actionId == 7) return 2; // SetMarketFee
    if (actionId == 8) return 3; // SetDumpControlValues
    if (actionId == 9) return 2; // SetWhaleMinLimit
    if (actionId == 10) return 2; // CreateMarketV2
    if (actionId == 11) return 2; // CreateMarketV3
    if (actionId == 12) return 2; // AddMarket
    if (actionId == 13) return 3; // AllowLaunchMarket <-- Mismatch in here
    if (actionId == 14) return 3; // RemoveMarket
    if (actionId == 15) return 3; // AddDexPeriphery
    if (actionId == 16) return 3; // RemoveDexPeriphery
    if (actionId == 17) return 4; // SetFeeReceiver
    if (actionId == 18) return 4; // LockFeeReceiver
    if (actionId == 19) return 2; // SetLowMarketFeeTemps
    if (actionId == 20) return 3; // SetSnipeControlValues
    if (actionId == 21) return 3; // SetWhaleRatePercents
    return super._getActionApproveCount(actionId);
}
```

UPDATES

- *Dec 29, 2022*: This issue has been acknowledged and fixed by the Formacar Action team. The document was updated.

2.2.2. `feeReceiver` can miss the `LowMarketFeeTemps` LOW

The governance contract uses the `_lowFeePair` state to store the latest pair that wants to update market fees. Then the `feeReceiver` have to call the `acceptLowMarketFee` function to update the pair market fee. But, if the governance triggers to update another pair, the `feeReceiver` can't accept the old pair anymore (the _lowFeePair, _lowFeeBuyMillis, _lowFeeSellMillis were overridden).

```
function _setLowMarketFeeTemps(address pair, uint buyMillis, uint sellMillis) private
{
    require(markets[pair].isMarket, 'FCG: not exist');
    require(sellMillis < 1000 && buyMillis < 1000, 'FCG: limit is 1000 millis');

    _lowFeePair = pair;
    _lowFeeBuyMillis = uint16(buyMillis);
    _lowFeeSellMillis = uint16(sellMillis);
    _lowFeeAt = block.timestamp;
}

function acceptLowMarketFee() external
{
    require(msg.sender == feeReceiver, 'FCG: only fee receiver');
    require(_lowFeeAt + 1 hours > block.timestamp, 'FCG: expired');

    Market storage market = markets[_lowFeePair];
    require(market.isMarket, 'FCG: is not market');

    if (market.buyFeeMillis != _lowFeeBuyMillis) market.buyFeeMillis = _lowFeeBuyMillis;
    if (market.sellFeeMillis != _lowFeeSellMillis) market.sellFeeMillis =
_lowFeeSellMillis;
    _lowFeeAt = 0;

    emit MarketFeeUpdated(_lowFeePair, _lowFeeBuyMillis, _lowFeeSellMillis);
}
```

RECOMMENDATION

The team should update the logic of this flow and allow the `feeReceiver` to pass the pair address in the argument.

UPDATES

- *Dec 29, 2022*: This issue has been acknowledged by the Formacar Action team.

2.2.3. Gas optimize INFORMATIVE

Using `memory` argument will force Solidity to copy it to memory which will cost more gas than using from `calldata` especially when passing large readonly array.

RECOMMENDATION

Consider change `memory` to `calldata` in all contracts for gas saving.

UPDATES

- *Dec 29, 2022*: This issue has been acknowledged by the Formacar Action team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 29, 2022*	Public Report	Verichains Lab

Table 4. Report versions history