Code security assessment

Shiba World Cup

CRYPTO
ALERT
NFT

BLOCKCHAIN SECURITY ANALYSIS GROUP

October 2022

v0.1

Index (click of section title for fast navigation)

Synopsis

On October 24th 2022, the Shiba World Cup's development team formally requested our services to evaluate its smart contract for vulnerabilities or issues, as well as the soundness of its code architecture.

Our analysis unit conducted both automated and manual analysis, including line-by-line examination by our team of coding experts, checking the whole code for potential vulnerabilities, centralization issues, unused/redundant code, optimization oportunities, compiling errors, timestamp/order dependecies, reentrancy errors, known attack vectors and coding best practices.

The contracts were then deployed in the testnet for extensive live testing of all their functionalities, including several different kind of interactions with multiple clients, and a stress test.

After our initial analysis, recommendations were made to the development team.

This document provides a full report of any isues found.

Overview

Project details	
Name	Shiba World Cup
Web	https://shiba-worldcup.io/
Blockchain	Binance Smart Chain
Environment	EVM
Language	Solidity
Compiler version	0.8.17
Contract's Source	Contract deployed at bsc address: 0x27DCC73CbBbe57d006303316dD3e91A0D5d58eeA
Contracts Audited	ShibaWorldCup.sol IRouterV2.sol IFactoryV2.sol
Direct imports (contracts and libraries)	ERC20.sol, Ownable.sol, ERC20Burnable.sol, Context.sol, IERC20.sol, IERC20Metadata.sol
Methodology used	Automated Static analysis Computer assisted code review Manual line-by-line code review Test deployment and stress testing

Variable & Event List

Variables	Type
lpPair	Public
treasury	Public
sellTax	Public
hasLiquidity	Public
onSwap	Private
Mapping	**Type**
_lpPairs	Private
_isExcluded	Private
Events	**Type**
ModifiedExclusion	-
ModifiedPair	-
NewTreasury	-
NewRouter	-

Function list 1/2

ShibaWorldCup.sol		
Type	**Name**	**Results**
Constructor	Constructor	**No issues found**
Internal	_taxSwap	**No issues found**
Internal	_checkLiquidity	**No issues found**
Internal	_transfer	**Minor issue (MI1)**
External payable	receive	**No issues found**
External	isExcluded	**No issues found**
External	isLpPair	**No issues found**
External	setExcluded	**Centralization issue (CE1)**
External	setLpPair	**Centralization issue (CE1)**
External	setTreasury	**Centralization issue (CE1)**
External	setRouter	**Centralization issue (CE1)**
External	burnTax	**Centralization issue (CE2)**
External	swapTax	**Centralization issue (CE1)**

Function list 2/2

IRouterV2.sol		
Type	**Name**	**Results**
External	factory	**No issues found**
External	WETH	**No issues found**
External	addLiquidityETH	**No issues found**
External	swapExactETHforTokens	**No issues found**
External	getAmountsOut	**No issues found**
External	getAmountsIN	**No issues found**
External	swapExactTokensForETHSupportingFeeOnTransferTokens	**No issues found**
External	swapExactETHForTokensSupportingFeeOnTransferTokens	**No issues found**

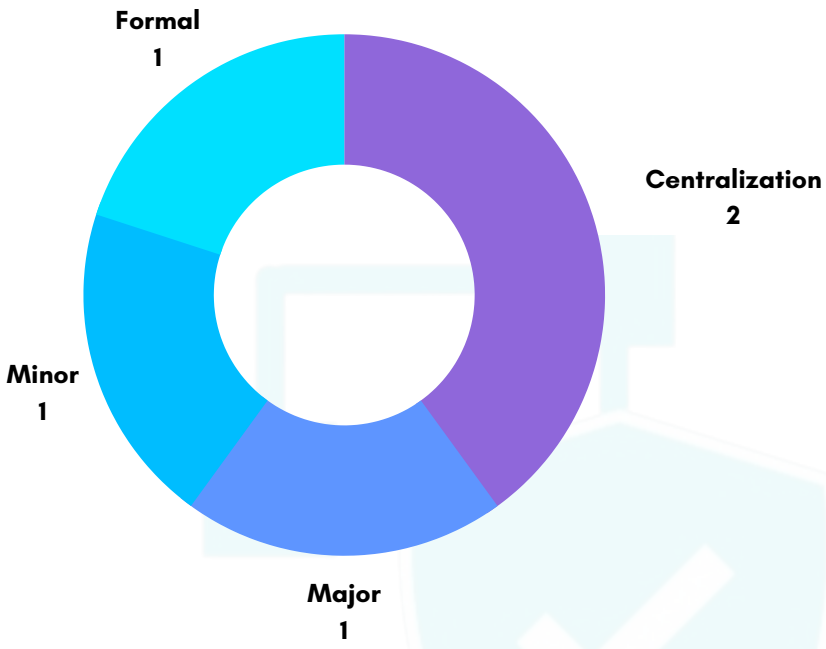
IFactoryV2.sol		
Type	**Name**	**Results**
External	getPair	**No issues found**
External	createPair	**No issues found**

*ERC20.sol**		
Type	**Name**	**Results**
Public	transferFrom	**Major Issue (MA1)**

*ERC20.Burnable.sol**		
Type	**Name**	**Results**
Public	burnFrom	**Major Issue (MA1)**

**Only functions with issues are show for direct import OZ contracts*

Findings summary



Formal: Comments on coding style and best practices issues. Mostly subjective.

Minor: Low-risk issues that cannot harm the contract's execution or expected behaviour.

Major: Medium-risk issues that can harm the contract or it's expected behaviour in a limited way.

Critical: High-risk issues that can seriously harm the contract or compromise the ecosystem's security.

Centralization: Excess of centralized privilege can potentially represent a unfair playground for holders and investors.

Coin2FishToken.sol contract findings:

Severity Level	Found	Objected	Noted	Mitigated	Resolved
Formal	2	0	0	0	0
Minor	1	0	0	0	0
Major	1	0	0	0	0
Centralization	2	0	0	0	0
Critical	0	0	0	0	0

CE1 | Administrative priviledges of owner Role

Current project design grants several administrative priviledge to a single owner role.
Any party who controls the owner wallet will be able to:

- Exclude addresses from taxes (function *setExcluded*)
- Change LP pair (function *setLpPair*)
- Change Treasury address (function *setTreasury*)
- Change Dex Router (function *setRouter*)
- Swap the tokens inside the contract (function *swapTax*)
- Burn the tokens inside the contract (function *burnTax*) *See CE2.

The contract emits an event that broadcast to the Blockchain when changes are made via the *setExcluded*, *setLpPair*, *setTreasury* and *setRouter* functions. This mitigates the issues as any modification is immediately broadcasted to the Blockchain.

However, no events are emited when the *swapTax* and *burnTax* functions are used, so this transactions. We recommend that the development team imform the community whenever this function is called.
We encourage the implementation of adequate safety measures (like multisiing mechanisms) to protect the owner wallet and its private key from any potential malicious third party.

CE2 | Owner can burn the full balance of tokens stored in the contract

The the contract includes non-standard *_tranfer* function. Every time it is called by a non-excluded addres it collects a number of tokens inside the contract as a tax:

```
139
140             // tax transfer sent to this contract
141             super._transfer(sender, address(this), taxAmount);
142             // default transfer sent to recipient
143             super._transfer(sender, recipient, amount - taxAmount);
```

Function *burnTax* allows any party in control of the owner role to burn any amount of tokens present in the contract, up to the full balance of them:

```
236     */
237     function burnTax(uint256 amount) external onlyOwner {
238         uint256 balance = balanceOf(address(this));
239         require(amount > 0 && balance > 0, "Zero amount");
240         uint256 toBurn = amount > balance ? balance : amount;
241         _burn(address(this), toBurn);
242     }
243
```

MA1 | Approved spenders can burn holder's tokens

The *_spendAllowance* mapping is used by the transferFrom function so that an user may approve a third party to transfer their tokens, this type of function is usually used by staking, betting, games or similar dapps to manage user's tokens.

```
158     function transferFrom(
159         address from,
160         address to,
161         uint256 amount
162     ) public virtual override returns (bool) {
163         address spender = _msgSender();
164         _spendAllowance(from, spender, amount);
165         _transfer(from, to, amount);
```

However, in the current contract implementation the function burnFrom also uses the *_spendAllowance* mapping to check how many tokens the user has approve a third party to burn:

```
35      function burnFrom(address account, uint256 amount) public virtual {
36          _spendAllowance(account, _msgSender(), amount);
37          _burn(account, amount);
38      }
39  }
```

Since the same mapping is used by both functions, when an address approves theirs tokens to be transferred by *transferFrom* it is also approving them to be burn by *burnFrom*. Because both functions are declared *public* this can potentially lead to accidental burns or be exploited by a third party to burn user's token .

MI01 | Ineficient code in internal transfer function

The contract uses a non-standar internal *_transfer* function that contains a high number of additional logical checks, including nested conditionals:

```
103     function _transfer(
104         address sender,
105         address recipient,
106         uint256 amount
107     ) internal virtual override {
108         require(
109             sender != address(0x0),
110             "ERC20: transfer from the zero address"
111         );
112         require(
113             recipient != address(0x0),
114             "ERC20: transfer to the zero address"
115         );
116         require(amount > 0, "Transfer amount must be greater than zero");
117 
118         if (!hasLiquidity) {
119             _checkLiquidity();
120         }
121 
122         if (!onSwap) {
123             if (hasLiquidity) {
124                 uint256 balance = balanceOf(address(this));
125                 if (balance > 0) {
126                     _taxSwap(balance);
127                 }
128             }
129         }
130 
131         // check whitelist
132         bool excluded = _isExcluded[sender] || _isExcluded[recipient];
133 
134         if (excluded || !_lpPairs[recipient]) {
135             super._transfer(sender, recipient, amount);
136         } else {
137             // sell tax amount
138             uint256 taxAmount = (amount * sellTax) / 100;
139 
140             // tax transfer sent to this contract
141             super._transfer(sender, address(this), taxAmount);
142             // default transfer sent to recipient
143             super._transfer(sender, recipient, amount - taxAmount);
144         }
145     }
```

This implementation is inneficient, as each logic check has to be made every time the function is called, thus increasing the gas cost of any transactions that use it.

FO1 | Uncommented code

Contract ShibaWorldCup.sol has plenty of clear and well organized comments, but on the other hand, contracts IRouterV2.sol and IFactoryV2.sol have no commentaries at all.
Clearly commented code is one of the most widely recommended best practice in Smart contract development as it improves their readability.

File 3 of 9 : IFactoryV2.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity >=0.6.0 <0.9.0;
3
4  interface IFactoryV2 {
5      event PairCreated(
6          address indexed token0,
7          address indexed token1,
8          address lpPair,
9          uint256
10     );
11
12     function getPair(address tokenA, address tokenB)
13         external
14         view
15         returns (address lpPair);
16
17     function createPair(address tokenA, address tokenB)
18         external
19         returns (address lpPair);
20 }
```

FO2 | Missing emit in _tranfer function

The standard _transfer fuction includes an instruction that emits an event each time the it is called:

```
226     function _transfer(
227         address from,
228         address to,
229         uint256 amount
230     ) internal virtual {
231         require(from != address(0), "ERC20: transfer from the zero address");
232         require(to != address(0), "ERC20: transfer to the zero address");
233 
234         _beforeTokenTransfer(from, to, amount);
235 
236         uint256 fromBalance = _balances[from];
237         require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
238         unchecked {
239             _balances[from] = fromBalance - amount;
240         }
241         _balances[to] += amount;
242 
243         emit Transfer(from, to, amount);
244 
245         _afterTokenTransfer(from, to, amount);
246     }
```

The non-standard version implemented in ShibaWorldCup.sol doesn't include an event to broadcast to the Blockchain when the function is used. Whilethis doesn't affect the correct execution of the contract, is a departure from current recommend best practices.

```
103     function _transfer(
104         address sender,
105         address recipient,
106         uint256 amount
107     ) internal virtual override {
108         require(
109             sender != address(0x0),
110             "ERC20: transfer from the zero address"
111         );
112         require(
113             recipient != address(0x0),
114             "ERC20: transfer to the zero address"
115         );
116         require(amount > 0, "Transfer amount must be greater than zero");
117 
118         if (!hasLiquidity) {
119             _checkLiquidity();
120         }
121 
122         if (!onSwap) {
123             if (hasLiquidity) {
124                 uint256 balance = balanceOf(address(this));
125                 if (balance > 0) {
126                     _taxSwap(balance);
127                 }
128             }
129         }
130 
131         // check whitelist
132         bool excluded = _isExcluded[sender] || _isExcluded[recipient];
133 
134         if (excluded || !_lpPairs[recipient]) {
135             super._transfer(sender, recipient, amount);
136         } else {
137             // sell tax amount
138             uint256 taxAmount = (amount * sellTax) / 100;
139 
140             // tax transfer sent to this contract
141             super._transfer(sender, address(this), taxAmount);
142             // default transfer sent to recipient
143             super._transfer(sender, recipient, amount - taxAmount);
144         }
145     }
```

Final considerations

After our analysis of the reviewed contracts we found no critical issues While the code doesn't seems to have any vulnerabilities that could be exploited by 3rd parties, there are some centralization issues that could lead to loss of funds if the owner wallet's private key gets compromised, or if the backend that controls it gets hacked.

We recommend setting up a decentralised control mechanism to address the Single Point of Failure of the current design. Multiple signature wallets may mitigate the risk, specially if they are managed by different members of the team.

We also recommend the implementation of a notification mechanism that informs users when the swapTax and burnTax fuctions are used as a way to further increase the transparency of the project.

Version history

Version	Changelog
v0.1	• Initial review of contract ShibaWorldCup.sol • Suggestions and findings sent to the development team.

Disclaimer

The information here provided is not, and must not be considered, endorsement, approval or disapproval of the audited project.
This audit report DOES NOT CONSTITUTE INVESTMENT ADVICE. Its scope is limited the technical and centralization aspects of the summited Smart Contracts.
Our team HAS NOT MADE ANY EVALUATION of the project's viability or economic design. Neither do we make any claims about the development team's ability, proficiency or well meaning.
The scope of this review DOES NOT INCLUDE neither the dapp norr the backend of any web3 included in the project design that interacts with the analized smart contract, and in consequence we cannot assure their security.
This audit does not constitute any warranty of the absolute bug-free nature of the code or associated technologies used. Neither can it foresee any unwanted results of its interaction with third party solutions like, exchanges, in-game databases, or others.
Crypto Alert NFT is in NO WAY RESPONSIBLE for any harm, malfunction or asset loss you might incour while interacting with the smart contracts here evaluated, nor is in any way liable for any detrimental results from your interaction with any aspects of the evaluated project.

October 2022

CRYPTO ALERT NFT

BLOCKCHAIN SECURITY ANALYSIS GROUP

https://t.me/cryptoalertnft (community)

https://www.cryptoalertnft.com/ (web)