Code security assessment

Coin2Fish Reborn

BLOCKCHAIN SECURITY ANALYSIS GROUP

August 2022

v1.0

Index (click of section title for fast navigation)

Index....1
Synopsis....2
Overview....3
Variable list....4
Function list....5
Findings' summary....6
Centralization issues overview 7
CE1 | Owner can withdraw the full token balance of the contract....8
CE2 | Owner can arbitrarily set the price of withdraws.... 9
CE3 | Owner can block any address from claiming tokens....10
CE4 | Owner controls LP tokens from liquidity addition mechanism....11
MA1 |Inefficient mechanism for liquidity addition....12
FO1 | Redundant code....14
FO2 | Unused code.... 15
FO3 | Inefficient function type declaration....16
FO4 | Method used to send ether differs from recommended.... 17
FO5 | Opcodes optimization....18
Final considerations....19
Version history.... 20
Disclaimer.... 21

Synopsis

On july 28 2022, the Coin2Fish 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 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. Several changes and improvements were then implemented.

This document provides a full report of any isues found as well as the development team's response to them.

Overview

Project details	
Name	Coin2Fish Reborn
Web	https://www.coin2fish.io/
Blockchain	Binance Smart Chain
Environment	EVM
Language	Solidity
Compiler version	0.8.15
Contract's Source	Original contract deployed at bsc address: 0x4ed661F77274659DaC071Ae881C3b858837B7816 Updated contract deployed at bsc address: 0x965eDD6B429B664082ce56FF31632446FF562d03
Contracts Audited	Coin2FishToken.sol
Direct imports (contracts and libraries)	ERC20.sol, Context.sol, IERC20.sol, IERC20Metadata.sol, SafeMath.sol Ownable.sol, MultiSignature.sol, IUniswapV2Factory.sol, IUniswapV2Router02.sol, IUniswapV2Router01.sol
Methodology used	Automated Static analysis Computer assisted code review Manual line-by-line code review Test deployment and stress testing

Variable List

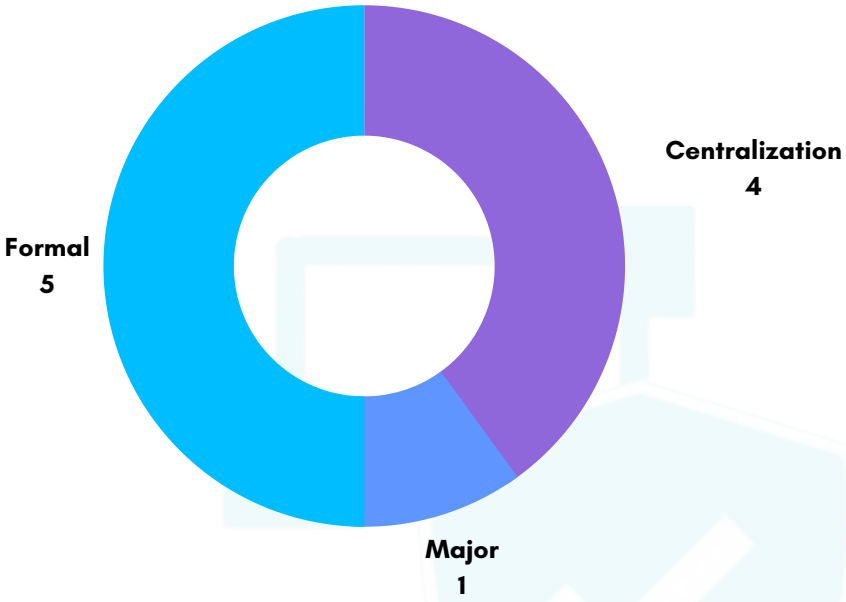
Public variables
uniswapV2Router
eggSalesEnabled
withdrawPrice
_maxWalletAmount
_maxTransactionAmount
addressHeisenDev
addressMarketing
addressTeam
taxFeeHeisenDev
taxFeeMarketing
taxFeeTeam
taxFeeLiquidity
_poolHeisenDev
_poolMarketing
_poolTeam
_poolLiquidity

Private variables
_tokenTotalSupply
firstLiquidityEnabled
_authorizedWithdraw
_accountTransactionLast
_accountTransactionCount
_accountWithdrawalLast
_accountWithdrawalCount
_maxTransactionCount
_maxWithdrawalCount
_maxTransactionWithdrawAmount
_isExcludedFromFees
_isExcludedFromLimits
automatedMarketMakerPairs

Function list

Type	Name	Results
Constructor	Constructor	Formal aspect comment (FO2)
External payable	buyEgg	No issues found
External payable	firstLiquidity	No issues found
External	teamPayment	Centralization issue (CE1)
Internal	_transfer	Formal aspect comment (FO4)
Private	swapAndAddLiquidity	Major issue (MA1)
Private	swapTokensForEth	Major issue (MA1)
Private	updateTaxesFees	Centralization issue (CE1)
Private	updateWithdrawOptions	Centralization issue (CE1, CE2)
Private	updateEggSales	No issues found
Private	addLiquidity	Centralization issue (CE4)
External	withdrawAuthorization	Centralization issue (CE3)
External	withdrawAllowance	No issues found
Internal	isUnderHourlyTransactionLimit	No issues found
Internal	isUnderDailyWithdrawalLimit	No issues found
External	withdraw	Centralization issue (CE2, CE3)
External	submitProposal	No issues found
External	approveProposal	No issues found
Private	_getApprovalCount	No issues found
External	executeProposal	No issues found
External	revokeProposal	No issues found

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	5	0	0	0	5
Minor	0	0	0	0	0
Major	1	0	0	0	1
Centralization	4	0	0	2	2
Critical	0	0	0	0	0

Centralization issues overview

1.1 Current project design grants a great degree of control to a single owner role. We encourage the client to carefully manage the privileged account private key to avoid any potential of being hacked.

As the scope of this audit does not include any review of the web3 / dapp code, we treat this entities as black boxes, assuming they are fully secure. However, in a real world scenario they might be compromised. Iñ

We advice the client to assign the privileged roles to a multiple-signature wallet, a DAO or other descentralized mechanism, as means to avoid the Single Point of Failure design.

Actions taken: A mutisignature mechanism was added. All functions with *onlyOwner* modifier now require 3 signatures.

Status: Mitigated

1.2 From a Software Engineering point of view we also consider it is advisable to consider separating in two the current contract, as to adhere to the Single Responsibility Principle(SRP):

- Token contract.
- Sales and withdraws contract.

Actions taken: None. The team acknowledges the issue but adduces this centralized structure is necessary for their game mechanics design.

Status: Acknowledged

CE1 | Owner can withdraw the full token balance of the contract

Any party who gains control of the owner role can withdraw the full amount of tokens in the contract by consecutively abusing the functions *updateTaxesFees*, *withdrawAuthorization*, *teamPayment*.

Team response:
The following limitations were added:

- Owners cannot make withdrawals.
- Heisen/Team/Marketing cannot make withdrawals.
- Fee cannot exceed 75%.
- If the user has pending withdrawals, new ones cannot be created.
- The withdrawal cannot exceed the maximum withdrawal (100K).

```
274
275     function withdrawAuthorization(address to, uint256 amount, uint256 fee) external onlyBackend {
276         require(!isAnOwner(to), "Owners can't make withdrawals");
277         require(to != backend(), "Backend can't make withdrawals");
278         require(to != addressHeisenDev, "Heisen can't make withdrawals");
279         require(to != addressMarketing, "Skyler can't make withdrawals");
280         require(to != addressTeam, "Team can't make withdrawals");
281         require(fee <= 75, "The fee cannot exceed 75%");
282         require(_authorizedWithdraws[to] == 0, "User has pending Withdrawals");
283         require(amount <= _maxTransactionWithdrawAmount, "Amount can't exceeds the max transaction withdraw
    amount");
284
285         uint256 amountFee = amount.mul(fee).div(100);
286         uint256 totalTaxes = taxFeeHeisenDev + taxFeeMarketing + taxFeeTeam;
287         if (totalTaxes == 0) {
288             _poolHeisenDev = _poolHeisenDev.add(amountFee);
289         }
```

Status: Mitigated

(While functions *teamPayment* and *withdrawAuthorization* still have a significant amount of centralized privileges, the limits added, together with the new multisign requirements, reduce the risk of abuse by third parties or external malicious actors)

CE2 | Owner can arbitrarily set the price of withdraws (honeypot risk)

The function *updateWithdrawOptions* can arbitrarily assign any value to the *withdrawalPrice* variable, thus changing the ether price required for execution of the *withdraw* function:

```
290
291     function updateWithdrawOptions(uint256 _withdrawPrice) external onlyOwner {
292         withdrawPrice = _withdrawPrice;
293         emit UpdateWithdrawOptions(_withdrawPrice);
294     }
```

This could be used to force all users to spend increased amounts of ether to be able to execute the *withdraw* function, or could even effectively block all claims, by setting a non-viable ether price:

```
354
355     function withdraw() public payable {
356         require(msg.value >= (withdrawPrice), "The amount sent is not equal to the amount required for withdraw");
357         uint256 amount = _authorizedWithdraws[msg.sender];
358         super._transfer(address(this), msg.sender, amount);
359         _authorizedWithdraws[msg.sender] = 0;
360         emit Withdraw(amount);
361     }
```

Team response: In the new contract version, the withdraw price is limited to a maximum of 0.005 BNB or 5000000000000000 wei.

```
354     ) external onlyOwner {
355         if (_updateWithdrawOptions) {
356             require(withdrawPrice <= 5000000000000000, "MultiSignatureWallet: Must keep 5000000000000000 Wei or less");
357         }
358         if (_updateTaxesFees) {
359             uint256 sellTotalFees = _heisenDevTaxFee + _marketingTaxFee + _teamTaxFee + _liquidityTaxFee;
360             require(sellTotalFees <= 10, "MultiSignatureWallet: Must keep fees at 10% or less");
361         }
362         if (_transferBackend) {
363             require(_backendAddress != address(0), "MultiSignatureWallet: new owner is the zero address");
364         }
```

Status: Resolved

CE3 | Owner can block any address from claiming tokens

The *withdrawAuthorization* function assigns the value of the *_autorizedWithdraws* variable. An address with owner privileges can assign any arbitrary value to it.

Since the *withdraw* function uses the value of *_autorizedWithdraws* to check how many tokens any address is able to claim, this could be potentially used to arbitrarily limit the withdrawals for any address, or, by setting the value of *_autorizedWithdraws* to 0, effectively use the function as a blacklist.

Team response: The function was modified, so that once the withdraw is set, it cannot be changed.

```
275     function withdrawAuthorization(address to, uint256 amount, uint256 fee) external onlyBackend {
276         require(!isAnOwner(to), "Owners can't make withdrawals");
277         require(to != backend(), "Backend can't make withdrawals");
278         require(to != addressHeisenDev, "Heisen can't make withdrawals");
279         require(to != addressMarketing, "Skyler can't make withdrawals");
280         require(to != addressTeam, "Team can't make withdrawals");
281         require(fee <= 75, "The fee cannot exceed 75%");
282         require(_authorizedWithdraws[to] == 0, "User has pending Withdrawals");
283         require(amount <= _maxTransactionWithdrawAmount, "Amount can't exceeds the max transaction withdraw
amount");
284
285         uint256 amountFee = amount.mul(fee).div(100);
286         uint256 totalTaxes = taxFeeHeisenDev + taxFeeMarketing + taxFeeTeam;
287         if (totalTaxes == 0) {
288             _poolHeisenDev = _poolHeisenDev.add(amountFee);
289         }
290         else {⌨}
306         _authorizedWithdraws[to] = amount;
307     }
```

Status: Mitigated.

(The function could still act as a blacklist under certain circumstances, however, the new multisignature design, together with the *onlyBackend* requirements, significantly reduces the risk of abuse)

CE4 | Autoliquidity mechanism sends LP tokens to owner

Every time the contract added liquidity to the pair on the DEX, the resulting LP tokens were sent to the owner wallet:

```
306     function addLiquidity(uint256 tokens, uint256 bnb) private {
307         _approve(address(this), address(uniswapV2Router), tokens);
308         uniswapV2Router.addLiquidityETH{value: bnb}(
309             address(this),
310             tokens,
311             0, // Take any amount of tokens (ratio varies)
312             0, // Take any amount of BNB (ratio varies)
313             owner(),
314             block.timestamp.add(300)
315         );
316         payable(addr()).transfer(address(this).balance);
317     }
```

If they weren't locked frequently, any party that gained control of the owner wallet could use those LP tokens to drain liquidity from the token pair on the DEX.

Team response: LP tokens are now sent to the contract Instead of the owner, and the contract doesn't have a function to remove liquidity.

Status: Resolved

MA1 | Inefficient mechanism for liquidity addition (1/2)

In original design, function *swapAndAddLiquidity*, swaped half the tokens in *_poolLiquidity* to obtain *ether*, then added liquidity with the other half:

```
250     function swapAndAddLiquidity() public onlyOwner {
251         uint256 half = _poolLiquidity.div(2);
252         uint256 otherHalf = _poolLiquidity.sub(half);
253         uint256 initialBalance = address(this).balance;
254         swapTokensForEth(half);
255         uint256 newBalance = address(this).balance.sub(initialBalance);
256         addLiquidity(otherHalf, newBalance);
257         _poolLiquidity = 0;
258         emit SwapAndAddLiquidity(half, newBalance, otherHalf);
259     }
```

This formula was highly inefficient, as it created a significant differential of *ether*. This surplus was then sent to the owner wallet, causing a net loss of liquidity for the project every time the function was executed:

```
306     function addLiquidity(uint256 tokens, uint256 bnb) private {
307         _approve(address(this), address(uniswapV2Router), tokens);
308         uniswapV2Router.addLiquidityETH{value: bnb}(
309             address(this),
310             tokens,
311             0, // Take any amount of tokens (ratio varies)
312             0, // Take any amount of BNB (ratio varies)
313             owner(),
314             block.timestamp.add(300)
315         );
316         payable(addr()).transfer(address(this).balance);
317     }
```

We recommended implementing a more efficient liquidity addition mechanism. By adjusting the token and BNB ratio, the amount of ether that gets sent to the owner wallet can be reduced to almost zero.

MA1 | Inefficient mechanism for liquidity addition (2/2)

Team response: The function was modified so that it now uses all the BNB available for addition of liquidity.
The function that sent the surplus of BNB to the owner was eliminated:

```
219
220     function swapAndAddLiquidity() private {
221         uint256 contractBalance = address(this).balance;
222         swapTokensForEth(_poolLiquidity);
223         uint256 liquidityTokens = balanceOf(address(this)).mul(10).div(100);
224         addLiquidity(liquidityTokens, contractBalance);
225         _poolLiquidity = 0;
226     }
227
228     function swapTokensForEth(uint256 tokenAmount) private {
229         address[] memory path = new address[](2);
230         path[0] = address(this);
231         path[1] = uniswapV2Router.WETH();
232
233         _approve(address(this), address(uniswapV2Router), tokenAmount);
234
235         uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
236             tokenAmount,
237             0,
238             path,
239             address(this),
240             block.timestamp
241         );
242     }
```

Status: Resolved

FO1 | Redundant code

The following code is redundant and should be removed without risk of affecting the functionality of the contract:

```
225         if (from == owner() && to == owner()) {
226             super._transfer(from, to, amount);
227             return;
228         }
```

Optionally, this section could also be removed, without risking increased gas consumption, as is is rarely used:

```
220         if (amount == 0) {
221             super._transfer(from, to, 0);
222             return;
223         }
```

Team response: Redundant code was removed.

Status: Resolved

FO2 | Unused code

Certain elements are never used by the contract, and in consecuence, should be removed:

- Mapping *_blacklistedAccount*
- Modifier *lockTheSwap*
- Variable swapping
- Events *UpdateUniswapV2Router*, *SetAutomatedMarketMakerPair*

Additionally, class variable *uniswapV2Pair* is only used in the constructor and therefore could be declared locally.

Team response: Unused code was removed.

Status: Resolved

FO3 | Inefficient function type declaration

Functions that are never called by the contract, specifically those functions that require the modifier *onlyOwner*, should be declared *External* Instead of *Public*. *External* functions are more efficient that the *Public* type.

Team response: The type of functions is updated according to their access:
Public - Any address can access.
External - Cannot be accessed internally, only externally.
Internal - Only this contract and contracts deriving from it can access.
Private - Can be accessed only from within this contract.

Status: Resolved

FO4 | Method used to send ether differs from recommend

The method currently used by the contract is significantly different from the recommended way of sending ether, as described in the following example:

```
function sendViaCall(address payable _to) public payable {
    // Call returns a boolean value indicating success or failure.
    // This is the current recommended method to use.
    (bool sent, bytes memory data) = _to.call{value: msg.value}("");
    require(sent, "Failed to send Ether");
}
```

We suggest implementing it to adhere to current Solidity's best practices and recommendations.

Team response: The Ether sending function was updated according to best practices, the [bytes memory data] was eliminated to prevent the warning.

Status: Resolved

FO5 | Opcodes optimization

The following code sections could be reworked to spend less opcodes:

```
bool takeFee = true;

// if any account belongs to _isExcludedFromFee account then remove the fee
if (_isExcludedFromFees[from] || _isExcludedFromFees[to]) {
    takeFee = false;
}
```

Suggested alternative:

```
// if any account belongs to _isExcludedFromFee account then remove the fee
bool takeFee = !(_isExcludedFromFees[from] || _isExcludedFromFees[to]);
```

Team response: Modification made as suggested.

Status: Resolved

Final considerations

In our analysis of the Coin2FishToken.sol contract we found some important areas that can be improved.

While the code doesn't seems to have any vulnerabilities that could be exploited by 3rd parties, there are some important 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.

Furthermore, in our criteria, using a single contract as token, marketplace and bank goes against the Single Responsibility Principle and increases the overall Attack Surface Area. Separating it's functionality in two or more smart contracts would increase the overall project security.

V1.0 update: After our initial review, the development team added a multiple-signature validation process for all centralized functions with onlyOwner require. This significantly reduces the risk of abuse by third parties or external malicious actors. It is still necessary to actively guard the backend from hacking vectors to prevent any potential exploits.

On the other hand, in the development team criteria, the current Single-Contract architecture is necessary for their game mechanics design and in consequence, there's no plans to divide the contract.

Version history

Version	Changelog
v0.1	• Initial review of contract Coin2FishToken.sol • Suggestions and findings sent to the development team.
v1.0	• Responses, code changes, mitigations and resolutions added to the report. • Multisignature implementation reviewed. • First public release version.

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.

August 2022

CRYPTO ALERT NFT

BLOCKCHAIN SECURITY ANALYSIS GROUP

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

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