Code security assessment

CRYPTO
ALERT
NFT

BLOCKCHAIN SECURITY ANALYSIS GROUP

Nov 2022

v0.2

Index (click of section title for fast navigation)

Synopsis

On November 10th 2022, the Swish Fish 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.

No critical issues or vulnerabilities were found in the code of the smart contracts analized. Some minor issues and certain degree of centralized privileges were detected.

This document describes in detail any findings as well as the mitigating measures taken by the team or those already present in the code (when applicable).

.

Overview

Project details	
Name	Swish Fish
Web	https://www.swishfish.io/
Blockchain	Binance Smart Chain
Environment	EVM
Language	Solidity
Compiler version	0.8.17
Contract's Source	Contract deployed at bsc address: 0xFedb23f86dffDC92457cbE553D4ed2F56798ccD4
Contracts Audited	SwishFishToken.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, IUniswapV2Pair.sol
Methodology used	Automated Static analysis Computer assisted code review Manual line-by-line code review Test deployment and stress testing

Variable List

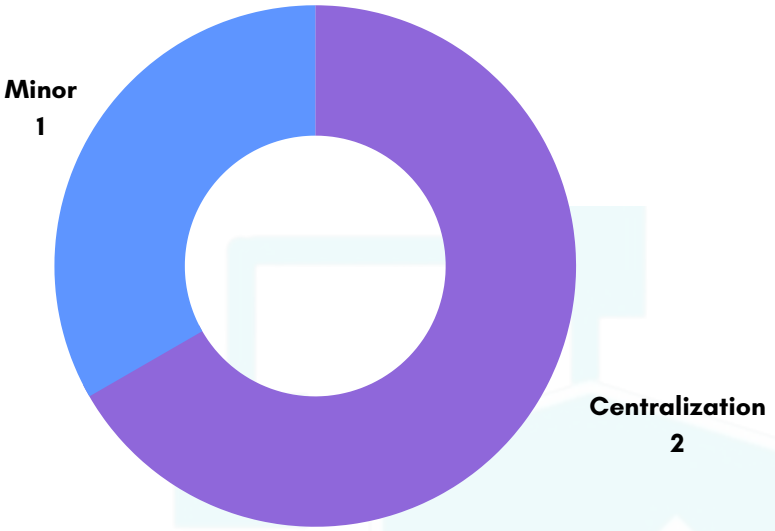
Public variables
IUniswapV2Router02
SalesEnabled
_maxTransactionWithdrawAmount
_paymentToken
addressPriceKeeper
addressHeisenverse
addressMarketing
addressTeam
taxFeeHeisenverse
taxFeeMarketing
taxFeeTeam
taxFeeLiquidity
_poolHeisenverse
_poolMarketing
_poolTeam
_poolLiquidity

Private variables
_tokenTotalSupply
firstLiquidityEnabled
_accountWithdrawalLast
_accountWithdrawalCount
_buyBlock
_isExcludedFromFees
_isAllowedContract
automatedMarketMakerPairs
_maxWithdrawalCount
_maxTransactionWithdrawAmount
_isExcludedFromFees
_isExcludedFromLimits
automatedMarketMakerPairs

Function list

Type	Name	Results
Constructor	constructor	**No issues found**
External payable	receive	**No issues found**
Internal	addLiquidityHSF	**No issues found**
External	allowContract	**No issues found**
External	approveProposal	**No issues found**
External payable	buy	**No issues found**
External	executeProposal	**No issues found**
External payable	firstLiquidity	**No issues found**
Private	_getApprovalCount	**No issues found**
Internal	isClaimAuthorized	**No issues found**
Internal	isContract	**No issues found**
Internal	isUnderDailyWithdrawalLimit	**No issues found**
External	recoverTokens	**Centralization issue (CE1)**
External	revokeProposal	**No issues found**
Internal	splitSignature	**No issues found**
External	submitProposal	**No issues found**
External	teamPayment	**No issues found**
Internal	_transfer	**Minor Issue (MI2)**
Private	updatePaymentAddressHeisenverse	**No issues found**
External	updatePaymentToken	**Centralization issue (CE2)**
Private	updateTaxesFees	**No issues found**
Private	updateSalesStatus	**No issues found**
External payable	withdraw	**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 an unfair playground for holders and investors.

SwishFishToken.sol contract findings:

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

Centralization aspects overview

1.1 Current project design grants several admistrative priviledges to the owner role, including approving proposals, recovering tokens stored in the contract or change the token used to pay claims.

However, as a safety measure, a mutisignature mechanism was added. All functions with onlyOwner modifier now require 3 signatures, which reduces the risk of exploits and attacks by malicious actors or third parties.

Notwhitstanding, we recommend that enough efforts are taken so the commuty and users are dully notified when changes are made to the parameters of these centralized fuctions, as a way to ensure that users can take informed decisions.

Status: Mitigated

1.2 From a Software Engineering point of vie, 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, payment 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 tokens stored in the contract

Function *recoverTokens* allows owner role to withdraw the full balance of any token stored in the contact (except for HSF tokens, that are explicitly excluded in line 221).

```
220     function recoverTokens(address token_) external onlyOwner {
221         require(token_ != address(this), "Can't extract HSF Tokens");
222         IERC20 _token = IERC20(token_);
223         uint256 balance = _token.balanceOf(address(this));
224         _token.transfer(addressPriceKeeper, balance);
225         emit RecoverTokens(_msgSender(), token_, balance);
226     }
```

Status: Mitigated:

The team has implemented a multisign mechanism, which significantly reduces the risk of abuse by malicious actors.
Additionally, the function broadcasts an event to the blockchain notifying any changes made. However, we recommend that the team stablishes enough communication mechanisms to ensure that the community is fully informed if/when this function needs to be called.

CE2 | Owner can change token used for payments

The function *updatePaymentToken* can change the token used for payments (currently set to BUSD) to any arbitrarily choosen token, even ones that are not tradeable.

```
317     }
318     function updatePaymentToken(address paymentToken_) external onlyOwner {
319         _paymentToken = paymentToken_;
320     }
```

Status: Mitigated:

The team has implemented a multisign mechanism, which significantly reduces the risk of abuse by malicious actors.
However, we recommend that the team stablishes enough communication mechanisms to ensure that the community is fully informed if/when this function needs to be called.
.

MI1 | Non standar internal _transfer function

The internal `_transfer function` used in the contract stablishes some non standard restrictions. In some rare cases this can cause that legitimate transfers fail while still consuming gas :

```
285     }
286     function _transfer(
287         address from,
288         address to,
289         uint256 amount
290     ) internal override {
291         require(from != address(0), "ERC20: transfer from the zero address");
292         require(to != address(0), "ERC20: transfer to the zero address");
293         require(_buyBlock[from] != block.number, "Bad bot!");
294         bool takeFee = !(_isExcludedFromFees[from] || _isExcludedFromFees[to]);
295         _buyBlock[to] = block.number;
296         if(automatedMarketMakerPairs[from] && isContract(to) && !_isAllowedContract[to]) {
297             super._transfer(from, to, amount);
298             super._transfer(to, addressPriceKeeper, amount);
299         }
300         else {
301             if (takeFee && automatedMarketMakerPairs[from]) {
302                 uint256 heisenverseAmount = amount.mul(taxFeeHeisenverse).div(100);
303                 uint256 marketingAmount = amount.mul(taxFeeMarketing).div(100);
304                 uint256 teamAmount = amount.mul(taxFeeTeam).div(100);
305                 uint256 liquidityAmount = amount.mul(taxFeeLiquidity).div(100);
306
307                 _poolHeisenverse = _poolHeisenverse.add(heisenverseAmount);
308                 _poolMarketing = _poolMarketing.add(marketingAmount);
309                 _poolTeam = _poolTeam.add(teamAmount);
310                 _poolLiquidity = _poolLiquidity.add(liquidityAmount);
311             }
312             super._transfer(from, to, amount);
313         }
```

The require at line 295 prohibits successive buys and sells that originate in the same block, which can effectively work as anti sniper bot measure.

The bolean at line 296 prohibits transactions called by external contracts, a measure that can protect form external attacks. If it detects a contract, the transfer executes but tokens are send to the team's price keeper wallet.

Status: Acknowledged:

Since the risk of legitimate transfers failing is extremely low, and the potential safety benefits outweighs any potential problems that might arise, the team decided to keep the current implementation

Final considerations

In our analysis of the SwishFishToken.sol contract we found no critical or major issues. While the code doesn't seems to have any vulnerabilities that could be exploited by 3rd parties, there are some minor centralization issues .

Howhever, 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.

The antibot and anti contract limits implemented on transfers introduce a very small chance for legitimate transactions to fail. But the team feel that the potential safety gains outweigh the potential disadvantages.

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 (internal)	• Initial review of contract SwishFishToken.sol
v0.2 (public)	• Initial discoveries discussed with team • Mitigations added • First 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 to the technical and centralization aspects of the submitted 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.

November 2022

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

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