Smart contracts security assessment

Final report

Tariff: Standard

DogMoneySwap

August 2022

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for the DogMoneySwap team. DogMoneySwap project is a fork of SushiSwap. Code was audited after the commit da760ef in @dogmoneyswap/dogmoneyswap GitHub repo. One of the most significant changes in the DogMoneySwap repo: MiniChefV2 contract was removed and the Multicall2 contract was added from @makerdao/multicall GitHub repo after 1e1b443 commit. The changes to the SushiSwap code do not introduce any new medium or high severity issues. SushiSwap contracts were audited before by PeckShield and Quanstamp. Two known issues that are not pointed out in the audits were added to the report.

Name	DogMoneySwap
Audit date	2022-08-18 - 2022-08-24
Language	Solidity
Platform	DogeChain

Contracts checked

Name	Address
Ownable	
SushiToken	
MasterChef	
Multicall2	
SushiBar	
SushiMaker	
Timelock	
UniswapV2Factory	
UniswapV2Router02	
UniswapV2ERC20	
UniswapV2Pair	
MasterChefV2	

SushiMakerKashi

SafeERC20

SafeMath

SignedSafeMath

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyse smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	not passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	not passed
Incorrect Inheritance Order	passed

Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.

Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. pendingOwner is not cancelled (Ownable)

When the function `transferOwnership` (L30) is called with the parameter `direct` equal to `true`, `pendingOwner` is not canceled. It makes it possible to the `pendingOwner` to reclaim ownership back by calling the `claimOwnership()` function. Checking the current owner is possible only by checking `owner` and `pendingOwner` variables.

Recommendation: Set `pendingOwner` to zero address after changing ownership via direct owner change.

Medium severity issues

1. Delegates are not moved (SushiToken)

The governance part of SushiToken doesn't work: the delegates aren't transferred with ordinary ERC20 transfers, i.e. `transfer()` and `transferFrom()`. There's a warning in the forked code in L8, and the governance mechanisms should not be implemented in the DogMoneySwap project.

Low severity issues

1. Redundant code (MasterChef)

The code inside the functions `setMigrator()` and `migrate()` is commented, therefore functions don't affect the state or view some value, but they will be available for external calls which might confuse users. Also, they will use excessive gas during deployment.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

0x Guard