0x Guard

Smart contracts security assessment

Final report

Tariff: Standard

Grindery

November 2022

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for **Grindery**.

The Grindery protocol consists of 2 contracts that interact with ERC20 tokens and several more helper contracts that are inherited by these 2 main contracts.

The first contract's goal is to help users to make multiple transfers of tokens in one transaction.

The second contract's goal is to collect tokens for sending to receivers. These tokens, receivers, and amounts that they will get are defined at the deployment of the contract.

The owner of the protocol has access to all tokens that are on the contracts.

The code is available at the GitHub repository and was audited after the commit e4b0f72591b3a961a9017d2ae1726ed7814cc0e5.

Report update.

The contracts' code was updated according to this report and rechecked after the commit 90522f245ac6fdc8d80dbfa78f462180ebe9f3f8.

Name	Grindery
Audit date	2022-10-26 - 2022-11-02
Language	Solidity
Platform	Harmony

Contracts checked

Name	Address
All contracts	
GrinderyBatchTransfer	

GrinderyDelegatedBatchTransfer

Transferable

Recoverable

Receivable

OwnerRecoverable

BatchTransferable

BatchRecoverable

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 analyze 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	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed

Presence of unused variables	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

No issues were found

Medium severity issues

1. Bad ERC20 transfers (Transferable)

Status: Fixed

There are some ERC20 tokens in the blockchain that have incorrect ERC20 implementations. These tokens don't return bool values in functions `transfer` and `transferFrom`. If the contract interacts with these tokens, functions `_transfer(address,uint256,address)` and `_transfer(address,uint256,address,address)` won't work with them.

Recommendation: It is better to use the SafeERC20 library from OpenZeppelin to protect the contract from losing some functionality or losing funds.

Update: The issue was fixed after the code update.

Low severity issues

1. Floating pragma (All contracts)
Status: Fixed

In all contracts, floating pragma is present. It is a bad practice, fixed pragma should be used instead.

Update: The issue was fixed after the code update.

2. No require messages (All contracts)
Status: Fixed

There are no error messages in any of the `require` statements. When a transaction fails, there is no message that clarifies the reason.

Update: The issue was fixed after the code update.

3. No required functions (GrinderyDelegatedBatchTransfer)
Status: Fixed

There are no functions that return the length of `recipients`, `amounts`, and `tokenAddresses` arrays.

Update: The issue was fixed after the code update.

4. Default variable visibility (GrinderyDelegatedBatchTransfer)
Status: Fixed

For global variables `recipients`, `amounts`, and `tokenAddresses` there are no visibility modifiers.

Update: The issue was fixed after the code update.

5. Gas optimization (GrinderyDelegatedBatchTransfer)
Status: Fixed

In the modifier `isRecoverable()` the global variable `tokenAddresses.length` and global arrays `amounts`, `tokenAddresses` are read multiple times.

In the function `completeTransfer()` global arrays `amounts` and `tokenAddresses` are read

multiple times.

In the function `batchRecover()` the global variable `tokenAddresses.length` is read multiple times.

It is better to use local variables instead of global variables wherever it is possible.

Update: The issue was fixed after the code update.

6. Repeating code (Recoverable)
Status: Fixed

Lines 29 and 30 can be replaced with the modifier `isTransferableRecipient()` which was inherited from the `Transferable` contract.

Update: The issue was fixed after the code update.

7. Indexing event parameters (Recoverable)
Status: Fixed

In the event `Recovered` parameter `token` is not indexed.

Update: The issue was fixed after the code update.

Conclusion

Grindery All contracts, GrinderyBatchTransfer, GrinderyDelegatedBatchTransfer, Transferable, Recoverable, Receivable, OwnerRecoverable, BatchTransferable, BatchRecoverable contracts were audited. 1 medium, 7 low severity issues were found.
1 medium, 7 low severity issues have been fixed in the update.

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.
OxGuard retains exclusive publishing rights for the results of this audit on its website and social networks.

0x Guard