SuperBid

smart contract audit report

Prepared for:
superbid.io

Authors: HashEx audit team

April 2021

Contents

Disclaimer

This is a limited report on our findings, based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn’t say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind, except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report.

The analysis of security is purely based on smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Introduction

HashEx was commissioned by the SuperBid team to perform an audit of SuperBid token smart contracts. The audit was conducted between April 27 and April 29, 2021.

The audited smart contract is deployed to Ethereum mainnet at the address of 0x0563DCe613D559a47877fFD1593549fb9d3510D6.

The audited contract is an ERC20 token with some extensions.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts.
- Ensure that smart contract functions perform as intended.

Information in this report should be used to understand the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

Contracts overview

SimpleERC20

Implementation of ERC20 token standard [1] based on erc20-generator by vittominacori.

Found issues

ID	Title	Severity	Response
01	Usage of the transfer function to send ether	Low	
02	ERC20 transfer return value	Low	
03	Pragma version is not fixed	Low	
04	ServiceReceiver should be an interface	Low	
05	Strict equality to check price	Low	
06	Recommendations	Low	

#01 Usage of the transfer function to send ether Low

`withdraw()` function in the ServiceReceiver (L876) uses transfer() method of sending ether. The recommended method is `call{value}()` with a reentrancy guard as 2300 of gas forwarded by transfer() may not be enough.

It must be noted that although the ServiceReceiver contract code is presented in the verified sources the ServiceReceiver contract was not deployed with the SuperBid token and its code is used as an interface reference to a previously deployed ServiceReceiver contract.

#02 ERC20 transfer return value Low

`recoverERC20()` function in the TokenRecover contract (L839) uses transfer() method of an arbitrary token. We recommend using SafeERC20 library from OpenZeppelin since some popular ERC20 tokens like USDT don't implement returning transfer results.

It must be noted that although the TokenRecover contract code is presented in the verified sources the TokenRecover contract was not deployed with the SuperBid token and its code is used as an interface reference to a previously deployed TokenRecover contract.

#03 Pragma version is not fixed Low

Wide range of allowed Solidity versions has an increased chance of including vulnerable versions.

#04 ServiceReceiver should be an interface Low

ServiceReceiver contract may be declared as an interface with only `pay()` function if there's no intention to use its functionality with SimpleERC20 token.

#05 Strict equality to check price Low

`pay()` function in the ServiceReceiver contract (L861) contains strict equality that may be changed to greater than condition. This would be useful in case of complex prices.

It must be noted that although the ServiceReceiver contract code is presented in verified sources the ServiceReceiver contract was not deployed with the SuperBid token and its code is used as an interface reference to a previously deployed ServiceReceiver contract.

#06 General recommendations Low

For gas saving purpose the following procedures may be performed:

- library `Address` may be removed;

functions `name()`, `symbol()`, `decimals()`, `totalSupply()`, `balanceOf()`, `transfer()`, `transferFrom()`, `allowance()`, `approve()`, `increaseAllowance()`, `decreaseAllowance()`, `renounceOwnership()`, `transferOwnership()`, `pay()`, `getPrice()`, `setPrice()`, `withdraw()`, `generator()` should be declared external.

Conclusion

Reviewed contract is deployed at 0x0563DCe613D559a47877fFD1593549fb9d3510D6 in Ethereum mainnet. The audited contract is an ERC20 token made with standard OpenZeppelin templates with minor modifications.

No critical, high or medium severity issues were found.

Audit includes recommendations on code improvement.

References

1. EIP-20: ERC-20 Token Standard

Appendix. Issues' severity classification

We consider an issue critical if it may cause unlimited losses or breaks the workflow of the contract and could be easily triggered.

High severity issues may lead to limited losses or break interaction with users or other contracts under very specific conditions.

Medium severity issues do not cause full loss of functionality but break the contract logic.

Low severity issues are typically nonoptimal code, unused variables, errors in messages. Usually, these issues do not need immediate reactions.

Appendix B. List of examined issue types

Business logic overview

Functionality checks

Following best practices

Access control and authorization

Reentrancy attacks

Front-run attacks

DoS with (unexpected) revert

DoS with block gas limit

Transaction-ordering dependence

ERC/BEP and other standards violation

Unchecked math

Implicit visibility levels

Excessive gas usage

Timestamp dependence

Forcibly sending ether to a contract

Weak sources of randomness

Shadowing state variables

Usage of deprecated code