CERTIK

Certik Final Report For Sandbox

Contents

Contents 1

Disclaimer 2

About CertiK 2

Executive Summary 3

Testing Summary 4

Review Notes 5

- Introduction 5
- Documentation 6
- Summary 6
- Recommendations 6

Findings 8

Exhibit 1 8

Exhibit 2 9

Exhibit 3 10

Exhibit 4 11

Exhibit 5 12

Exhibit 6 13

Exhibit 7 14

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 Verification Services Agreement between CertiK and Sandbox (the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, 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 CertiK's prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

For more information: https://certik.org.

Executive Summary

This report has been prepared for Sandbox to examine issues and vulnerabilities in the source code of their smart contracts in scope. A comprehensive examination has been performed, utilizing CertiK's Static Analysis, Manual and Dynamic Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Testing Summary

SECURITY LEVEL

Smart Contract Audit

This report has been prepared as a product of the Smart Contract Audit request by The Sandbox. This audit was conducted to discover issues and vulnerabilities in the source code of The Sandbox's Estate and updated LandSale (now called EstateSale) contracts.

TYPE	Smart Contracts & Token
SOURCE CODE	https://github.com/pixowl/sandbox-private-contracts/tree/audit_estate_20200416/src
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	May 12, 2020
DELIVERY DATE	May 18, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by the Sandbox team to audit the design and implementations of the to be released ERC721 based smart contracts. The audited files:

- Estate.sol
- EstateSale/EstateSale.sol
- Estate/EstateBaseToken.sol
- BaseWithStorage/ERC721BaseToken.sol
- contracts_common/src/Interfaces/ERC721Events.sol
- contracts_common/src/BaseWithStorage/SuperOperators.sol
- contracts_common/src/BaseWithStorage/MetaTransactionReceiver.sol
- ReferralValidator/ReferralValidator.sol
- contracts_common/src/BaseWithStorage/Admin.sol

In the repository:

- https://github.com/pixowl/sandbox-private-contracts/tree/audit_estate_20200416/src

The goal of this audit is to review Sandbox implementation for its business model, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

Documentation

Even though the code is clear and very well written the documentation is somewhat lacking and is **something we would advise to be expanded**. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

Summary

The codebase of the project, especially with regards to the ERC721 estate tokens and its sale contract, was aimed to minimize gas usage by compact information encoding. Even the complex geometric nature of the game, the land positions and their mutual adjacency verification were cleverly coded.

While **some of the issues pinpointed were of negligible importance** and referred to coding standards and inefficiencies, **major and critical flaws** were identified that should be remediated as soon as possible to ensure the contracts of the Sandbox team are of the highest standard and quality.

These inefficiencies and flaws can be swiftly dealt by the development team behind the Sandbox project. We will create and maintain a direct communication channel between us and the Sandbox team to aid in amending the issues identified in the report.

Update on 21.05.2020: The Sandbox development team has responded and adequately addressed the issues in this audit report. **No new bugs or security issues occurred from these changes**. The **natspec** has also been updated for better code readability.

Recommendations

With regards to the codebase, the main recommendation we can make is **the expansion of the documentation to address the functionalities of the contracts** from an external perspective rather than an on-code perspective. Additionally, we advise that all our findings are carefully considered and assimilated in the codebase of the project to ensure the highest code standard is achieved.

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, errors and mistakes that appear throughout the code **to achieve a high standard of code quality and security**.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Function's visibility	Security	Critical	ERC721BaseToken.sol Lines 348 - 357

Description:

The function "_burn" 's visibility is public. For a certain owner anyone can call this function by setting the same input for "from" and "owner", hence burn the estate.

Recommendations:

Recommend setting the visibility to private/internal.

Update from Sandbox:

Fixed in commit "dc8d27da1ff020b0bd0d84c33929ab8c93d82806".

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Unchecked variable state	implementation	Major	ERC721BaseToken.sol Lines 60 - 64

[INFORMATIONAL] Description:

The function "_ownerAndOperatorEnabledOf" returns the owner of an estate and whether an operator for the estate has been set. When an estate is burnt by "_burn" the value "_owners[id]" is not set to "0" but still tracks the last owner by setting the 161-th digit to "1". The function "_ownerAndOperatorEnabledOf" does not consider this and would return the last owner even though the estate has already been burnt.

Recommendations:

Recommend using "_ownerOf(id)".

Update from Sandbox:

Not an issue in the current Estate but could be an issue in the future contract that uses the same code. Fixed in commit "dc8d27da1ff020b0bd0d84c33929ab8c93d82806".

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Unclear requirement	Implementation	Discussion	ERC721BaseToken.sol Lines 116-121

[INFORMATIONAL] Description:

The function “approve” cannot be used by “_metaTransactionContracts” even though it is allowed in “approveFor” and we can just set “sender” to be the owner of the estate in “approveFor” to have the same effect for “approve”.

Recommendations:

Recommend allowing “_metaTransactionContracts” in “approve” if intended.

Update from Sandbox:

Our meta transaction system uses the first parameter as identifier, so “approve” would have it as the operator. As such this is not possible.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Unclear requirement	Implementation	Discussion	EstateBaseToken.sol Lines 249-257, 269-277, 259-267.

[INFORMATIONAL] Description:

The implementation of “check_burn_authorized” , “check_add_authorized” and “check_create_authorized” indicates that Meta Transaction Contract, Superoperators, Operators for all, Operators, or even the owners of the estates are deprived of respective burning, adding, creating rights whenever a fixed global minter or breaker are set. Is this behaviour intended?

Update from Sandbox:

Yes, we are planning to potentially gate estate creation and breaking with a potential fee. A new set of contracts would be in charge to check the fees and then do the necessary breaking and creating.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Unclear code	Coding Style	Discussion	EstateBaseToken.sol Line 384

[INFORMATIONAL] Description:

The function `_checkAdjacency` always checks the adjacency of each newly added quad towards the last quad in the estate. Because of this the respective junction is left out which makes the junctions and quad lists unequal in length and usage of “_checkAdjacency” unintuitive.

Recommendations:

Recommend not leaving out these junctions.

Update from Sandbox:

Junctions are only necessary if the selection of quads follows a branching pattern. This means they only need to be put for shapes that require it. Passing a connecting quad for each submission would be possible but unnecessary.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Unchecked scenario	Coding Style	Informational	EstateSale.sol Line 218

[INFORMATIONAL] Description:

In the function “buyLandWithETH” it can happen that “msg.sender” is a contract, hence “msg.sender.transfer(msg.sender - ETHRequired)” would trigger the receive or fallback function and can lead to revert. Nowaday “.transfer” only forwards 2300 gas, hence reentrancy is not a threat here.

Recommendations:

Keep in mind this possibility and have an error message for this case.

Update from Sandbox:

We would leave that way as the “msg.sender” as a contract is expected to deal with ETH reception.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Repeated function calls	Security	Discussion	EstateSale.sol Lines 199-224, 237-257, 167-185.

[INFORMATIONAL] Description:

The functions “buyLandWithSand”, “buyLandWithETH”, “buyLandWithDAI” don’t check whether the input referral has been processed hence we need to be sure that an unauthorized user cannot call these functions more than the intended number of times.

Update from Sandbox:

This is fine. The referral is not per purchase so as long as they buy something the referral is valid.

Exhibit 34

TITLE	TYPE	SEVERITY	LOCATION
Integer overflow	Arithmetics	Major	FixidityLib.sol Lines 36 - 43

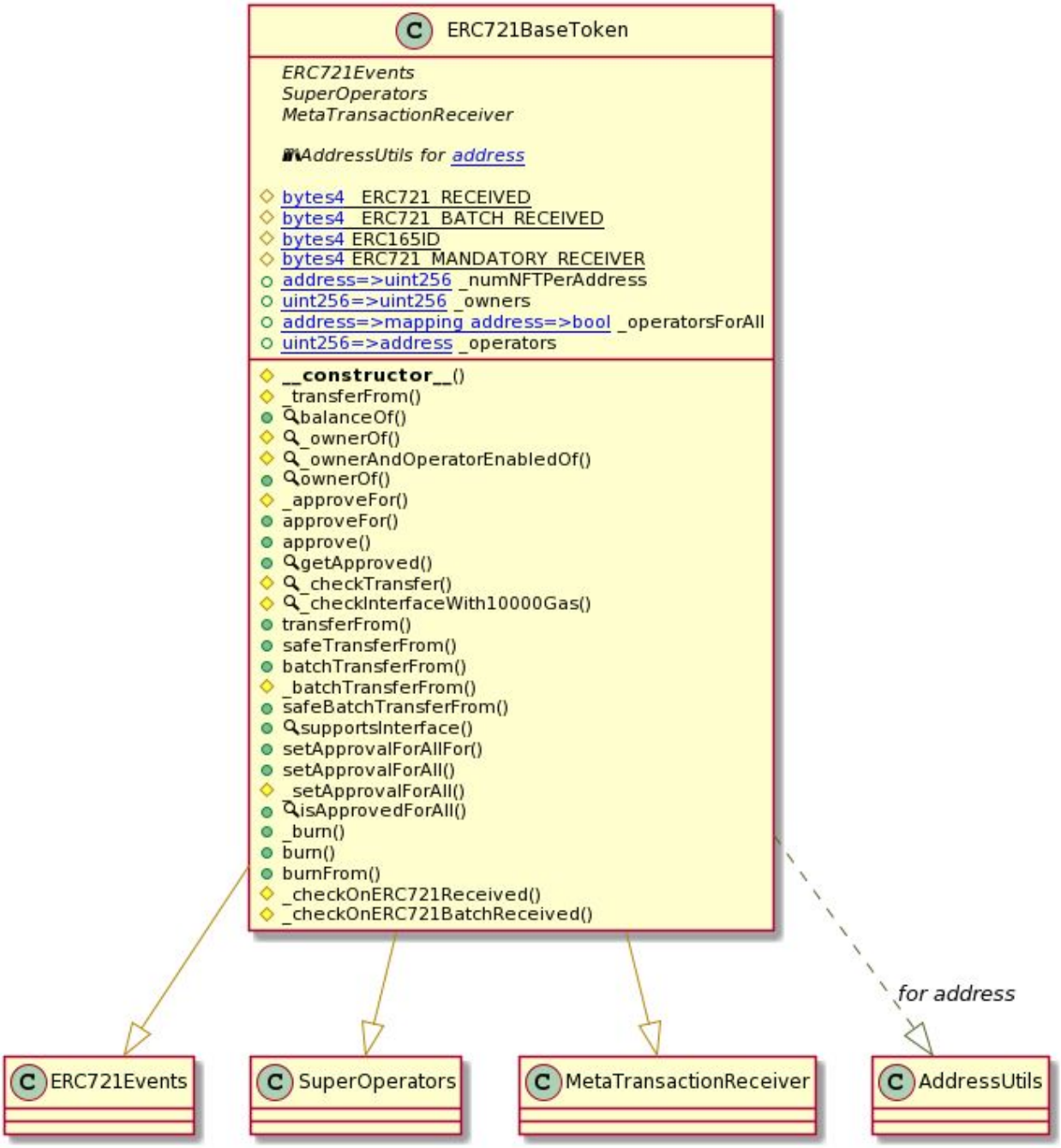
[INFORMATIONAL] Description:

The arithmetic expression on line 42 can overflow, for example if the number of decimal digits is 2 the input of the function “multiply” is 10**50 and 10**50, then the result would be unexpectedly - 422425...

Recommendations:

Use SafeMath to prevent integer overflow.


EstateBaseToken
ERC721BaseToken
uint16 GRID_SIZE
uint256 _nextId
uint256=>null _quadsInEstate
LandToken _land
address _minter
address _breaker
__constructor__()
setMinter()
getMinter()
setBreaker()
getBreaker()
createFromQuad()
addQuad()
createFromMultipleLands()
addSingleLand()
addMultipleLands()
createFromMultipleQuads()
addMultipleQuads()
burn()
burnFrom()
burnAndTransferFrom()
transferAllFromDestroyedEstate()
transferFromDestroyedEstate()
_withdrawalOwnerOf()
_check_owner_authorized()
_check_burn_authorized()
_check_create_authorized()
_check_add_authorized()
_check_withdrawal_authorized()
_encode()
_decode()
_mintEstate()
_addSingleQuad()
_addQuads()
_checkAdjacency()
_adjacent()
_addLands()
_addLand()
_idInPath()
onERC721BatchReceived()
onERC721Received()
supportsInterface()
ERC721BaseToken


ERC721BaseToken
ERC721Events
SuperOperators
MetaTransactionReceiver
AddressUtils for address
bytes4 ERC721_RECEIVED
bytes4 ERC721_BATCH_RECEIVED
bytes4 ERC165ID
bytes4 ERC721_MANDATORY_RECEIVER
address=>uint256 _numNFTPerAddress
uint256=>uint256 _owners
address=>mapping address=>bool _operatorsForAll
uint256=>address _operators
__constructor__()
_transferFrom()
balanceOf()
_ownerOf()
_ownerAndOperatorEnabledOf()
ownerOf()
_approveFor()
approveFor()
approve()
getApproved()
_checkTransfer()
_checkInterfaceWith10000Gas()
transferFrom()
safeTransferFrom()
batchTransferFrom()
_batchTransferFrom()
safeBatchTransferFrom()
supportsInterface()
setApprovalForAllFor()
setApprovalForAll()
_setApprovalForAll()
isApprovedForAll()
_burn()
burn()
burnFrom()
_checkOnERC721Received()
_checkOnERC721BatchReceived()
for address
ERC721Events
SuperOperators
MetaTransactionReceiver
AddressUtils