CERTIK

CertiK Audit Report for The Sandbox

Contents

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.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles

such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable. For more information: https://certik.io.

Executive Summary

This report has been prepared for **Sandbox** to discover issues and vulnerabilities in the source code of their **EstateSale** smart contracts in scope. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual 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 Sandbox.

This audit was conducted to discover issues and vulnerabilities in the source code of Sandbox's EstateSale contracts.

TYPE	Smart Contracts & Tokens
SOURCE CODE	https://github.com/thesandboxgame/sandbox-private-contracts/blob/audit_presale_4_1_20200716/src/EstateSale/EstateSale.sol
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	June 25, 2020
FINAL DELIVERY DATE	August 5, 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 their EstateSale and related smart contracts. The audited files and sha256 checksum are:

- src/EstateSale/EstateSale.sol
  `f053d9489bdd3a38a47c3ddbe068b5a8885aa702a02f35e566452505b1aa26ee`
- src/ReferralValidator/ReferralValidator.sol
  `00e03ce9cfe55f6efb47612d7d9ba6e4dddd3df4b94f63d69d5d9076ec4ab49d`
- src/contracts_common/src/BaseWithStorage/MetaTransactionReceiver.sol
  `9a4ed4de68284c558ab5ab2839d2807f3763a0c41008a25782b2ed415d97f375`
- src/contracts_common/src/Interfaces/ERC1155.sol
  `23e3c40c02dd895f82f522afd75743a7149dfa05966f9a6c3da2086c142ee49c`
- src/contracts_common/src/Libraries/SafeMathWithRequire.sol
  `74002761baaa6a55a3b5186478e9b01bac789507b9c84d345c4eb1a8fdef16cb`
- src/contracts_common/src/Interfaces/Medianizer.sol
  `f5a399e19a6238c05b22ced41d71bb3baf3aa315e3113aa6a072c266123c55a8`
- src/EstateSale/LandToken.sol
  `8ae036053b1745fb4f89359ce966af19673030e708fb585b5bcbf14f7097b387`

Located in

- https://github.com/thesandboxgame/sandbox-private-contracts/blob/audit_presale_4_1_20200716/src/EstateSale/EstateSale.sol

The goal of this audit was to review the Solidity 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

The sources of truth regarding the operation of the contracts in scope are **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 as well as the relevant markdown documentation.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the Sandbox team or reported an issue.

Summary

The codebase of the project, especially with regards to the EstateSale and related contracts, attempts to fulfill a use case that is intricate and ambitious and as such, **inefficiencies in both the design and implementation** of the various contracts were identified and properly documented.

While **most of the issues pinpointed were of negligible importance** and mostly referred to coding standards and inefficiencies, **minor, medium 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 were swiftly dealt with 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.

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, enforce linters and / or coding styles as well as correct any spelling 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
Substitution of “require” Calls w/ Modifier	Coding Style	Informational	EstateSale.sol: L80, L87, L100, L113 ReferralValidator.sol: L37, L63 MetaTransactionReceiver.sol: L18

Description:

Admin.sol contract exposes an `onlyAdmin` modifier that makes sure the `msg.sender` is equal to the `_admin` of the contract, however this check is manually carried out in multiple segments of the codebase.

Recommendations:

Recommend to reuse the `onlyAdmin` modifier.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Emitting events when states changed	Coding Style	Informational	EstateSale.sol: L78~L82, L86~L89, L99~L102, L112~L115 ReferralValidator.sol: L36~L40, L62~L65,

Description:

Throughout the codebase, there are numerous instances where variables are updated/set/changed/etc., without events emitted.

Recommendations:

EstateSale.sol:

- For `setReceivingWallet`, recommend to emit an event after the new wallet address is set.
- For `setDAIEnabled`, `setETHEnabled`, and `setSANDEnabled`, recommend to emit an event after the currency is enabled or disabled.

ReferralValidator.sol:

- For `updateSigningWallet`, recommend to emit an event after the signing wallet is updated.
- For `updateMaxCommissionRate`, recommend to emit an event after the max commission rate is updated.

Exhibit 3

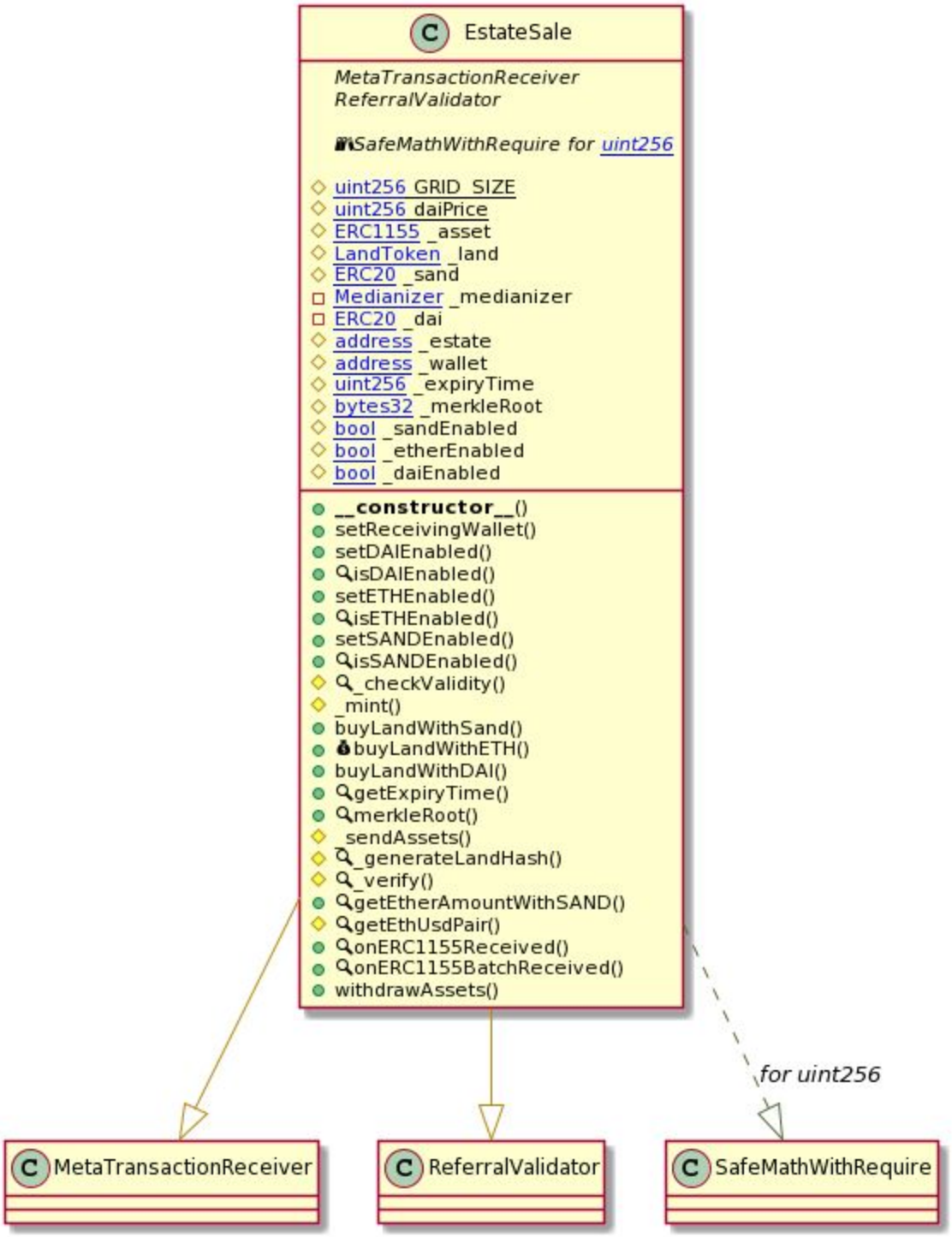
TITLE	TYPE	SEVERITY	LOCATION
Referrer address check	Implementation	Minor	ReferralValidator.sol: L87

Description:

In the function `handleReferralWithETH`, `referrer` is decoded from `referral`. Then no matter if the `isReferralValid` check is passed or not, money with the amount of `commission` would be transferred to the `referrer`.

Recommendations:

Some basic checks for `referrer` could be making sure that `referrer` is not `msg.sender` or `address(0)`.


EstateSale
MetaTransactionReceiver
ReferralValidator
SafeMathWithRequire for uint256
uint256 GRID_SIZE
uint256 daiPrice
ERC1155 _asset
LandToken _land
ERC20 _sand
Medianizer _medianizer
ERC20 _dai
address _estate
address _wallet
uint256 _expiryTime
bytes32 _merkleRoot
bool _sandEnabled
bool _etherEnabled
bool _daiEnabled
__constructor__()
setReceivingWallet()
setDAIEnabled()
isDAIEnabled()
setETHEnabled()
isETHEnabled()
setSANDEnabled()
isSANDEnabled()
_checkValidity()
_mint()
buyLandWithSand()
buyLandWithETH()
buyLandWithDAI()
getExpiryTime()
merkleRoot()
_sendAssets()
_generateLandHash()
_verify()
getEtherAmountWithSAND()
getEthUsdPair()
onERC1155Received()
onERC1155BatchReceived()
withdrawAssets()
for uint256
MetaTransactionReceiver
ReferralValidator
SafeMathWithRequire