invozone

InvoAudit

SMART CONTRACT AUDIT REPORT

for

InvoBlox RealState Escrow Contract

Prepared By: Muhammad Talha

InvoAudit

February 13, 2023

Document Properties

Client	InvoBlox
Title	Smart Contract Audit Report
Target	RealState Escrow Contract
Version	1.0
Author	Muhammad Talha
Auditors	Muhammad Talha
Reviewed by	Auditing Team
Approved by	Auditing Team
Classification	Public

Version Properties

Version	Date	Author(s)	Description
1.0	March 13, 2023	Muhammad Talha	Final Release
1.0-rc	March 18, 2023	Muhammad Talha	Release Candidate

Contact

Name	InvoAudit
Phone	+98798-4567
Email	InvoAudits@invoblox.com

1 | Introduction

Given the opportunity to review the design document and related smart contract source code of Auditchain, we outline in this report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contract can be further improved due to the presence of several issues. This document outlines our audit results.

1.1 About RealState Contract

Blockchain technology can benefit the real estate industry.For example, a smart mortgage contract can remove the need to depend on intermediaries and the constant back and forth between parties involved. Moreover, you wouldn't be waiting months for mortgage approvals.The code for the smart mortgage contract will grant your mortgage request the moment all the documents and requirements defined are fulfilled.In addition, when you rely less on brokers, banks, agents, and lawyers, these processes will significantly reduce the associated costs and processing times.

You can find the realState contract at this address:

- https://goerli.etherscan.io/address/0xfb3830def6cc7bbeedfdcbc787a216b53fb0824b#code

And this is the commit ID after all fixes for the issues found in the audit have been checked in:

-

1.2 About InvoAudit

InvoAudit Inc. is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing).

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;

- Impact measures the technical loss and business damage of a successful attack;

- Severity demonstrates the overall criticality of the risk.

  Likelihood and impact are categorized into three ratings: H, M and L, i.e., high, medium and low respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., Critical, High, Medium,

  To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

- Semantic Consistency Checks: We then manually check the logic of implemented smart con- tracts and compare with the description in the white paper.

- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

- To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699), which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

1.5 Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the RealState Escrow implementation. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logic, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	No of Findings
High	
Medium	
Low	
Informational	
Suggestions	
Total	33

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduc- tion, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities that need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues.

- 1 critical-severity vulnerability
- 2 medium-severity vulnerabilities
- 2 low-severity vulnerabilities.

ID	Severity	Title	Category	Status
001	High	3.1:_ address payable public seller; // Remove this state variable, check every seller against its own NFt.	Error Reporting	
002	High	3.2: Make a struct against that nft id_ and just store all the data in that struct and remove all those mappings.	Patching	
003	High	3.3:_ “approval” Remove this extra mapping.	Patching	
004	High	3.4:_address payable _seller remove this extra line of code from constructor	Patching	
005	Low	3.5:_ seller = _seller; remove this extra line of code from constructor	Patching	
006	Low	3.6:_address _buyer remove this variable from the list method parameters, any buyer could buy this listed nft.	Patching	

007	Low	3.7: remove this onlySeller modifier from the List method, This modifier can only list NFT to one seller.	Patching	
008	Low	3.8: _use IERC721 safe transfer from instead of transferFrom function to transfer the NFT in List Method.	Patching	
009	Medium	3.9: _Transfer the nft at the end of the List method to prevent the Reenterency attack or Use Reentrancy modifier.	Patching	
010	Low	3.10: _store the owner of the nft, who is listing. .	Patching	
011	Low	3.11: _buyer[_nftID] = _buyer; // Remove this extra line of code from here..	Data Validation	
012	High	3.12: onlyBuyer(_nftID) remove this onlyBuyer modifier it will restrict the contract against only one address. .	Error Reporting	
013	Medium	3.13: _Stored the buyer address against that nft id who paid the price in depositEarnest method.	Patching	

		.		
014	Medium	3.14:_there should be a check that NFt must be inspected before buyer deposited in depositEarnest method.	Patching	
015	High	3.15:_there should be a check that if someone paid against that nft_id already then revert the transaction in depositEarnest method. .	Error Reporting	
016	High	3.16:_add a require check that nft must be listed before inspection in updateInspectionStatus method. .	Denial of Service	
017	Medium	3.17:_remove this extra isDeposited[_nftID] check from here its gone be happen after the inspection in updateInspectionStatus method..	Denial of Service	
018	High	3.18:_// Remove this approveSale method,There is no need of this or anyone can call this function and make approval true.	Denial of Service	
019	High	3.19:_adda require check that only buyer can call this finalizeSale function	Denial of Service	

020	High	3.20: adda require check that nft isnt already sold in finalizeSale method then revert the transaction.	Denial of Service	
021	High	3.21: _Remove this extra approval[_nftID][buyer[_nftID]] check as buyer paid the amount there is a definite approval from him in in finalizeSale method. .	Error Reporting	
022	High	3.22: _Remove this extra approval[_nftID][seller] check in finalizeSale method. .	Denial of Service	
023	Medium	3.23: No need to set this variable again here in the finalizeSale method.	Denial of Service	
024	High	3.24: _store that the nft is sold in finalizeSale method.	Denial of Service	
025	High	3.25: _remove this _extra function cancelSale all the functionality must be done in the inspection function.	Denial of Service	
026	High	3.26: adda require check that nft isnt already sold in finalizeSale method then revert the transaction.	Denial of Service	

027	High	3.27: remove this require isDeposited[_nftID] check , inspection must be called before deposit function in cancelSale method.	Denial of Service	
028	High	3.28: _Remove this transfer amount code line because inspection must be called before deposit function so there will be no deposit from buyer in cancelSale method.	Error Reporting	
029	High	3.29: _ use IERC721 safe transfer from instead of transferFrom function to transfer the NFT in List Method in cancelSale method.	Denial of Service	
030	Medium	3.30: _Transfer the nft at the end of the cancelSale method to prevent the Reenterency attack or Use Reentrancy modifier in cancelSale method. .	Denial of Service	
031	High	3.31: _// Remove this onlyBuyer modifier.	Denial of Service	
032	High	3.32: _// Remove this onlySeller modifier.	Denial of Service	
033	High	3.33:Add pausable Functionality in the contract.	Denial of Service	

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.