invozone

InvoAudit

SMART CONTRACT AUDIT REPORT

for

InvoBlox MultiSig Wallet Contract

Prepared By: Muhammad Talha

InvoAudit

February 13, 2023

Document Properties

Client	InvoBlox
Title	Smart Contract Audit Report
Target	Multisig Wallet 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	February 13, 2023	Muhammad Talha	Final Release
1.0-rc	February 5, 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 MultiSig Wallet

Multi-signature wallets or “multisig wallets” for short, are a type of cryptocurrency wallet for which at least two private keys are needed to sign a transaction. Imagine a secure locker with two locks and two keys held by two parties that can only be opened if both provide their keys, thus ensuring that one party is not able to open the box without the other party’s permission.In the Bitcoin Lightning Network, a multi-signature transaction is required for opening a payment channel between two parties, where each partner locks a certain amount of bitcoins into a multisig wallet and then receives one of two required keys.In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://testnet.bscscan.com/address/0xd5c87b3d092ca6418eeca9a7ccb15875bbd58a5b#code

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

- https://goerli.etherscan.io/address/0x84475827c95ff11ad88b93554640a519302d5b73#code

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 Battle Stakes 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	10
Medium	4
Low	6
Informational	0
Suggestions	0
Total	20

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:_ Addition of a require check for total no of approval to add a new owner.	Error Reporting	Resolved
002	High	3.2:_ Addition of a require check for total no of approval for removing an owner.	Error Reporting	Resolved
003	High	3.3:_ Add require check _to address != 0	Error Reporting	Resolved
004	High	3.4:_ Add a require check for that _data != 0	Error Reporting	Resolved
005	Low	3.5:_ No need to define the value of variables with default values.	Patching	Resolved
006	Low	3.6:_ Removal of extra getOwners function.	Patching	Resolved
007	Low	3.7:_ Removal of extra getTransaction function.	Patching	Resolved
008	Low	3.8:_ Remove public bool confirmed state variable.	Patching	

009	Medium	3.9: Define the visibility of the totalTransaction state variable.	Patching	
010	Low	3.10: Removed this extra isExistOwner mapping.	Patching	
011	Low	3.11: Change all these mappings into single mapping against a struct.	Data Validation	
012	High	3.12: Add a require statement that _owner address != 0 in addNewOwner.	Error Reporting	
013	Medium	3.13: Remove this extra !isExistOwner[msg.sender] require statement in addNewOwner(address _owner) method.	Patching	
014	Medium	3.14: Remove this isExistOwner[msg.sender] = true statement.	Patching	
015	High	3.15: “confirmationsForAddOwner" should be updated in addNewOwner method.	Error Reporting	
016	High	3.16: Major new owner addition case handling is missing.	Denial of Service	
017	Medium	3.17: Add new function to set numConfirmationsRequired.	Denial of Service	
018	High	3.18: Logical error in removeOwner(uint256 _ownerIndex) method.	Denial of Service	

019	High	3.19: _ Logical error in removeOwner(uint256 _ownerIndex) method.	Denial of Service	
020	High	3.20: _ Logical error in removeOwner(uint256 _ownerIndex) method.	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.

3 | Detailed Results

First report of findings is:

https://docs.google.com/document/d/1Gf_7cY_4wIxlPi6fqY1bcjjBcd6_rYkU/edit

Contract address after First changing:
https://goerli.etherscan.io/address/0x25b98e9353adb05cd0b570e531c4f71641185832#code

Now the new changing in new contract are:

3.1:_ Addition of a require check for total no of approval to add a new owner.

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

There is no check on how many confirmations are needed or required to add a new owner , otherwise only a single owner can add a new owner.Which does not fulfill the multisig wallet requirement.

```
97      function addNewOwner(
98          address _owner
99          ) public
100           onlyOwner
101           {
102               owners.push(_owner);
```

3.2:_ Addition of a require check for total no of approval for removing an owner.

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

There is no check on how many confirmations are needed or required to remove an old owner , otherwise only a single owner can remove an old owner.Which is not full fill the multisig wallet requirement.

```
106     function removeOwner(
107         uint256 _ownerIndex
108         )  public
109           onlyOwner
110           {
111
112
113               owners[_ownerIndex] = owners[owners.length -1];
114               owners.pop();
```

3.3:_ Add require check _to address != 0

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

In the submitTransaction function add a new require statement that checks that the address _to wouldn't be zero.

```
117
118     function submitTransaction(
119         address _to,
120         uint256 _value,
121         bytes memory _data
122     ) public  {
123
```

3.4:_ Add a require check for that _data != 0

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

In the submitTransaction function add a new require statement that checks that the data wouldn't be zero.

```
117
118     function submitTransaction(
119         address _to,
120         uint256 _value,
121         bytes memory _data
122     ) public  {
123
```

3.5:_ No need to define the value of variables with default values.

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

Both variables in submitTransaction method:

1) transactions[txIndex].executed = false;
2) transactions[txIndex].numConfirmations = 0;

Both values are by default false and zero so no need to define them in the method and save some gas fee in the contract.

```
128
129         transactions[txIndex].to = _to;
130         transactions[txIndex].value = _value;
131         transactions[txIndex].data= _data;
132         transactions[txIndex].executed = false;
133         transactions[txIndex].numConfirmations = 0;
134
135
```

3.6:_ Removal of extra getOwners function.

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

Remove the getOwner() method to get the owners because the owners array is a public variable and solidity makes automatically getter functions for public variables.

```
204     function getOwners() public view returns (address[] memory) {
205         return owners;
206     }
```

3.7:_ Removal of extra getTransaction function.

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

Remove the getTransactionCount() method to get the transactions count and make the total transaction variable public because solidity will automatically generate a getter function for that.

```
function getTransactionCount() public view returns (uint256) {
    return totalTransaction;
}
```

New Findings after completing the remaining Functionalities

3.8:_ Remove public bool confirmed state variable.

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

Remove this variable because it isn't used in the whole contract.

3.9:_ Define the visibility of the totalTransaction state variable.

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

Visibility is necessary to limit the level of access to the variables and functions in your Solidity applications.Using the principle of least privilege helps to limit the potential for unintended consequences and security vulnerabilities in your smart contract. It also makes your code easier to understand and maintain. Therefore, it is highly recommended to use the principle of least privilege when assigning visibility to variables and functions in your Solidity applications.

3.10:_ Change all these mappings into single mapping against a struct.

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

Remove all those mappings “isOwner, isConfirmedForAddOwner, isConfirmedForRemoveOwner” and just use a single struct to store all the data against that owner’s address and use a single mapping for all that.

3.11:_ Add a require statement that _owner address != 0 in addNewOwner.

- ID: PVE-001
- Target: StakingContract
- Severity: Critical
- Category: Business Logic [4]

Description:

Add a require statement to verify that the address of the owner will not be equal to a zero address in “addNewOwner(address _owner)”.