Smart Contract Audit Report for Kommunitas

Audited By

Satyabrata Dash & Aliasgar Kapadia

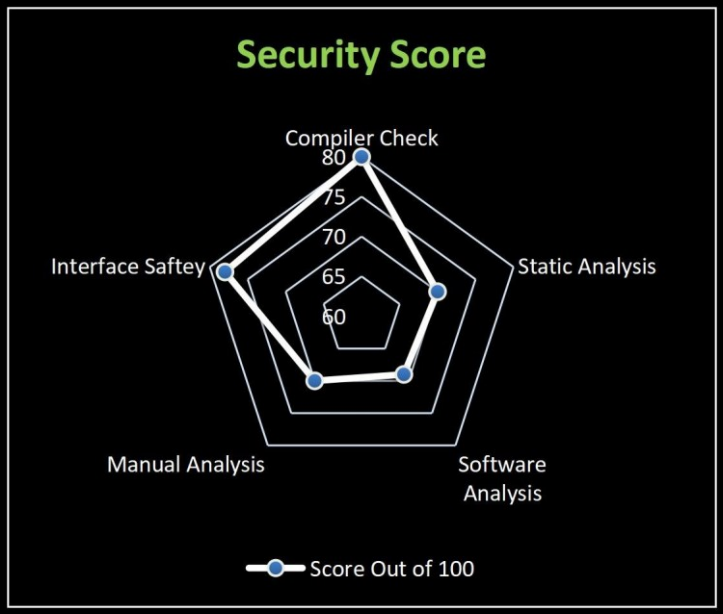
Report Prepared By

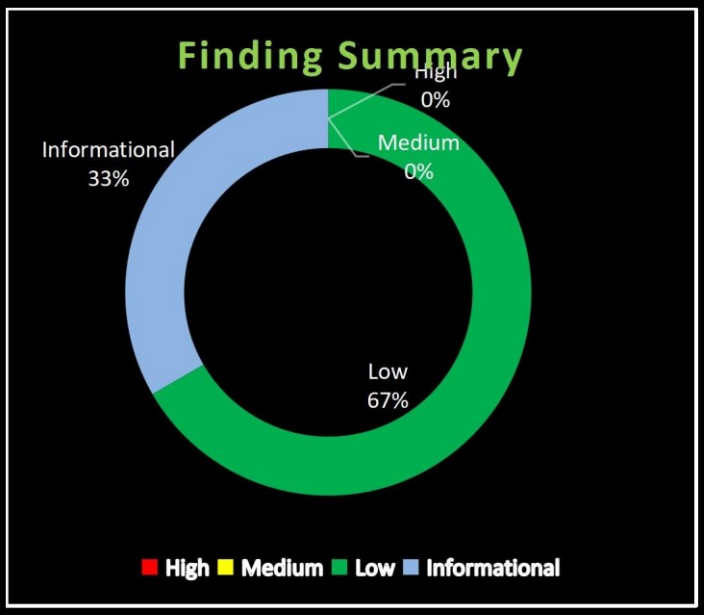
Aliasgar Kapadia

Table of Content

Executive summary

Project Name	Kommunitas
Project Type	Defi
Audit Scope	Check Security and code quality
Audit Method	Static and Manual
Audit Timeline	1th Aug 2022 to 31th March 2011
Source Code	PublicGovSale.sol, PublicGovFactory.sol, IPublicGovFactory.sol, TransferHelper.sol, IKommunitasStakingV2.sol, IKommunitasStaking.sol
Source code Hash	bb1c4787be4705d781db95382a79041e2ee5f8cf





Issue Tracking Table				
	High	Medium	Low	Informational
Open Issues	0	0	8	6
Acknowledged Issues				
Resolved Issues				

Types of Severities

- **High:** The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium:** The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
- **Low:** The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
- **Informational:** The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth.

Types of Issues

- **Open:** Security vulnerabilities identified that must be resolved and are currently unresolved.
- **Acknowledged:** Vulnerabilities which have been acknowledged but are yet to be resolved.
- **Resolved:** These are the issues identified in the initial audit and have been successfully fixed.

Checked Vulnerabilities

- ❖ Re-entrancy
- ❖ Access control
- ❖ Denial of service
- ❖ Timestamp Dependence
- ❖ Integer Overflow/Underflow
- ❖ Transaction Order Dependency
- ❖ Requirement Violation
- ❖ Gas Limit and Loops
- ❖ Incorrect Inheritance Order
- ❖ Centralization
- ❖ Unsafe external calls
- ❖ Business logic contradicting the specification
- ❖ Business logic contradicting the specification

Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the
- whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods and tools were used to review all the smart contracts.

- ### Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

- ### Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

- ### Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

- ### Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

- Tools and Platforms used for Audit

  - Remix IDE
  - Hardhat
  - Truffle Team
  - Mythril
  - Slither
  - Consensys Surya
  - Open Zeppelin Code Analyzer
  - Manticore

Findings

Manual Analysis

Contract Name: PublicGovFactory.Sol

High Severity Issues:

No Issue Found

Medium Severity Issues:

No Issue Found

Low Severity Issues:

Vulnerability: Transfer ownership should be 2 Steps process

Reference: Contract: PublicGovFactory.sol

```
function transferOwnership(address _newOwner) external override onlyOwner{
        require(_newOwner != address(0), "bad");
        owner = _newOwner;
    }
```

Description:

- When the owner mistakenly transfers ownership to an incorrect address, ownership is completely removed from the original owner and cannot be reverted.
- The **transferOwnership ()** function in the **PublicGovSale** contract allows the current owner to transfer his privileges to another address. However, inside **transferOwnership ()**, the **_newOwner** is directly stored in the owner, after validating the **_newOwner** is a non-zero address, which may not be enough.

Remediation:

- It would be much safer if the transition is managed by implementing a two-step approach: _transferOwnership() and _updateOwnership().
- Specifically, the _transferOwnership () function keeps the new address in the storage, _newOwner, instead of modifying the owner() directly. The _updateOwnership() function checks whether _newOwner is msg.sender, which means _newOwner signs the transaction and verifies himself as the new owner. After that, _newOwner could be set into _owner.

Status: Open

Vulnerability: Missing Zero Address Validation in createProject()

Reference: Contract: PublicGovFactory.sol

```
function createProject(
        uint128 _calculation,
        uint128 _start,
        uint128 _duration,
        uint128 _sale,
        uint128 _price,
        uint128[4] calldata _fee_d2,
        address _payment,
        address _gov
    ) external override onlyOwner returns(address project){
        require(_payment != address(0), "bad");
        require(_payment == allPayments[getPaymentIndex[_payment]], "!exist");

        project = address(new PublicGovSale());

        allProjects.push(project);

        PublicGovSale(project).initialize(
            _calculation,
            _start,
            _duration,
            _sale,
            _price,
            _fee_d2,
            _payment,
            _gov
        );

        emit ProjectCreated(project, allProjects.length-1);
    }
```

Description:

- Lack of zero address validation for **_gov** in **createProject()** function may lead to contract functionality might become inaccessible

Remediation:

- Consider adding zero address checks in order to avoid risks.

Status: Open

Vulnerability: Too much centralization

Reference: Contract: PublicGovFactory.sol

Description:

- The role owner has the authority to update critical settings of the contract like **Payment Token, Remove Payment Token, Change Fee Percentage, config** at any time.

Remediation:

- We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:
    - With reasonable latency for community awareness on privileged operations.
    - Multisig with community-voted 3rd-party independent co-signers.
    - DAO or Governance module increasing transparency and community involvement.

Status: Open

Informational Issues:

Vulnerability: Floating Pragma
Reference: Contract: PublicGovFactory.sol `pragma solidity ^0.8.13;`
Description: • Contract uses a floating pragma solidity ^0.8.13.
Remediation: • Contracts should be deployed with the same compiler version and flags that have been used during testing.
Status: Open

Vulnerability: Unclear Error message used

Reference: Contract: PublicGovFactory.sol

```
function setPayment(address _token) external override onlyOwner{
        require (_token != address(0), "bad");
        if(allPayments.length > 0) require(_token != allPayments[getPaymentIndex[_token]],
"existed");

        allPayments.push(_token);
        getPaymentIndex[_token] = allPayments.length-1;
    }
```

Description:

- Contract has so many unclear error messages in require statements, that lead to confusion. Like “bad”

Remediation:

- Try to use some meaningful error message that properly describe the revert condition.

Status: Open

Vulnerability: Missing Events for Some Critical Functions

Reference: Contract: PublicGovFactory.sol

```
function transferOwnership(address _newOwner) external override onlyOwner{
        require(_newOwner != address(0), "bad");
        owner = _newOwner;
    }
```

```
function setPayment(address _token) external override onlyOwner{
        require(_token != address(0), "bad");
        if(allPayments.length > 0) require(_token != allPayments[getPaymentIndex[_token]],
"existed");

        allPayments.push(_token);
        getPaymentIndex[_token] = allPayments.length-1;
    }
```

Description:

- The missing event makes it difficult to track off-chain changes. An event should be emitted for significant transactions calling the following functions: **transferOwnership ()**, **setPayment(), removePayment(), setPercentage(), config().**

Remediation:

- We recommend to emitting an event to log critical updates of mentioned important functions.

Status: Open

Contract Name: PublicGovSale.Sol

High Severity Issues:

No Issue Found

Medium Severity Issues:

No Issue Found

Low Severity Issues:

Vulnerability: Multiple Compiler versions are used (0.8.13 and 0.6.0)

Reference: Contract: PublicGovSale.Sol

```
pragma solidity ^0.8.13;
```

```
pragma solidity >=0.6.0;
```

Description:

- Two different type Solidity compiler versions used. In TransferHelper Contract use >=0.6.0 where other contracts use 0.8.13

Remediation:

- We recommend you to use a single (lock pragma) all over the contracts so that it will not lead to any unwanted bugs in future.

Status: Open

Vulnerability: Transfer ownership should be 2 Steps process

Reference: Contract: PublicGovSale.Sol

```
function transferOwnership(address _newOwner) external onlyOwner {
        require(_newOwner != address(0), "bad");
        owner = _newOwner;
    }
```

Description:

- When the owner mistakenly transfers ownership to an incorrect address, ownership is completely removed from the original owner and cannot be reverted.

- The **transferOwnership()** function in the **PublicGovSale** contract allows the current owner to transfer his privileges to another address. However, inside **transferOwnership()**, the **_newOwner** is directly stored in the owner, after validating the **_newOwner** is a non-zero address, which may not be enough.

Remediation:

- It would be much safer if the transition is managed by implementing a two-step approach: _transferOwnership() and _updateOwnership().

- Specifically, the _transferOwnership () function keeps the new address in the storage, _newOwner, instead of modifying the owner () directly. The _updateOwnership() function checks whether _newOwner is msg.sender, which means _newOwner signs the transaction and verifies himself as the new owner. After that, _newOwner could be set into _owner.

Status: Open

Vulnerability: Missing Zero Address Validation

Reference: Contract: PublicGovSale.Sol

```
    function setGov(address _gov) external onlyOwner {
        require(uint128(block.timestamp) < booster[1].start, "bad");

        gov = _gov;
    }
```

```
    function setPayment(address _payment) external onlyOwner {
        require(uint128(block.timestamp) < booster[1].start, "bad");

        payment = IERC20(_payment);
    }
```

Description:

- Lack of zero address validation for **setGov (), setPayment ()** function may lead to contract functionality might become inaccessible

Remediation:

- Consider adding zero address checks in order to avoid risks.

Status: Open

Vulnerability: Use of a dynamic array

Reference: Contract: PublicGovSale.Sol

Description:

- Functions like **migrateCandidates()**, **setWhitelist_d6()**, **updateWhitelist_d6()** take dynamic arrays as parameter, If the arrays are too long then that could lead to DoS(Denial of service) by exceeding the gas limits of block.

Remediation:

- Make sure to use limited size of array (capped array) or prefer batch transaction.

Status: Open

Vulnerability: Too much centralization

Reference: Contract: PublicGovSale.Sol

Description:

- The role owner has the authority to update critical settings of the contract like **Payment Token, Remove Payment Token, Change Fee Percentage, config** at any time.

Remediation:

- We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:
    - With reasonable latency for community awareness on privileged operations.
    - Multisig with community-voted 3rd-party independent co-signers.
    - DAO or Governance module increasing transparency and community involvement.

Status: Open

Informational Issues:

Vulnerability: Floating Pragma
Reference: Contract: PublicGovSale.Sol `pragma solidity ^0.8.13;`
Description: • Contract uses a floating pragma solidity ^0.8.13.
Remediation: • Contracts should be deployed with the same compiler version and flags that have been used during testing.
Status: Open

Vulnerability: Unclear Error message used

Reference: Contract: PublicGovSale.Sol

```
function setAllocation(address _user, uint32 _running) private returns(bool) {
     require(_setUserTotalStaked(_user), "bad#1");
     require(_setUserAllocation(_user, _running), "bad#2");

     return true;
 }
```

Description:

- Contract has so many unclear error messages in require statements, that lead to confusion. Like [‘bad’]

Remediation:

- Try to use some meaningful error message that properly describe the revert condition.

Status: Open

Vulnerability: Missing Events for Some Critical Functions

Reference: Contract: PublicGovSale.Sol

```
    function refund() external {
        uint128 _refund_d2 = refund_d2;
        uint256 amount = (uint256(getTotalPurchase(msg.sender)) * uint256(price) *
uint256(_refund_d2)) / 1e22; // 1e18 (token decimal) + 1e4 (percent 2 decimal)

        require(isRefund && _refund_d2 > 0 && isBuyer(msg.sender) && !refunded[msg.sender]
&& payment.balanceOf(address(this)) >= amount, "bad");

        refunded[msg.sender] = true;

        TransferHelper.safeTransfer(address(payment), msg.sender, amount);
    }
```

Description:

- The missing event makes it difficult to track off-chain changes. An event should be emitted for significant transactions calling the following functions: **refund (), moveFee (), moveFund (),**

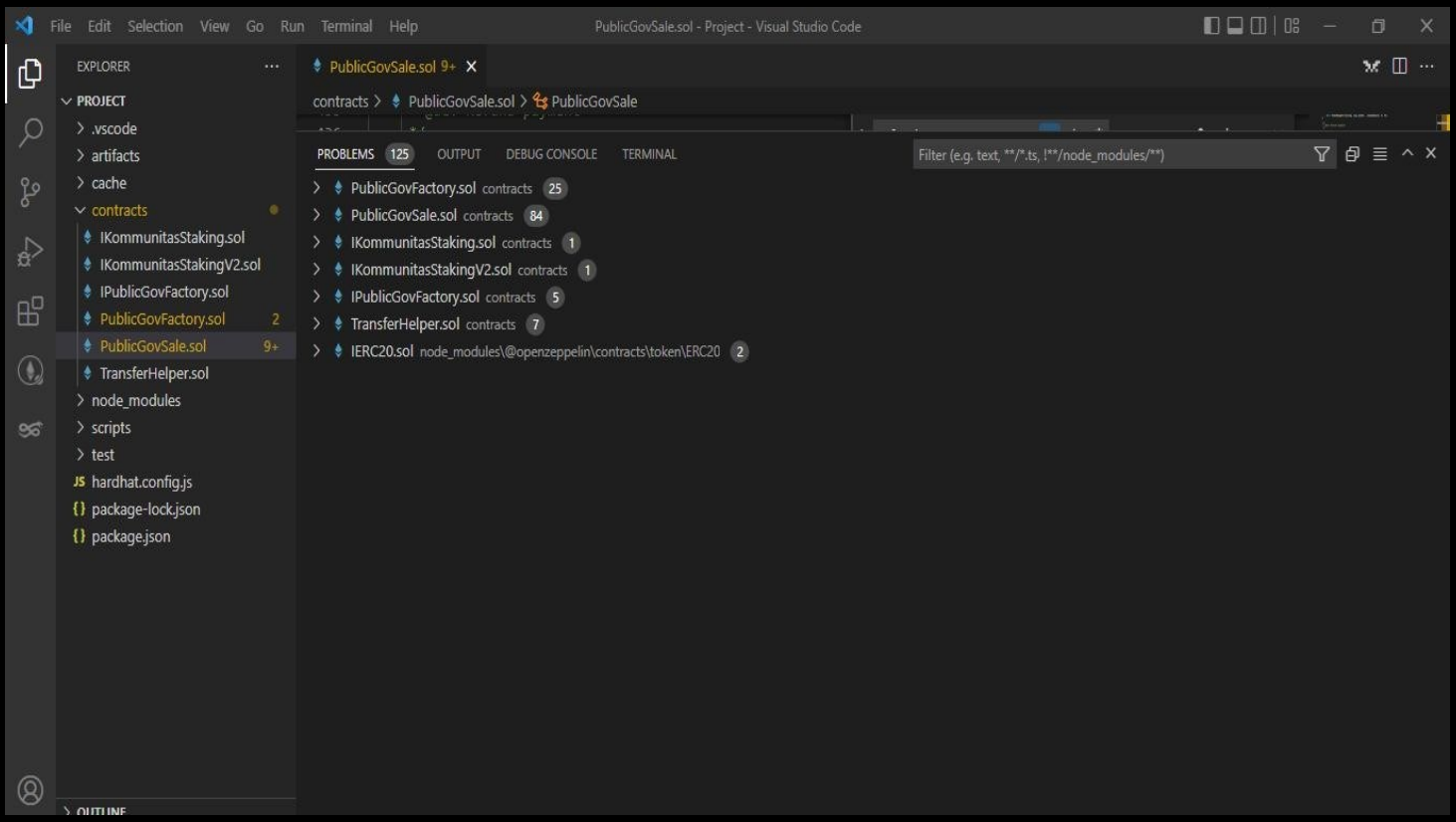
Remediation:

- We recommend to emitting an event to log critical updates of mentioned important functions.

Status: Open

Static Analysis

- **Slither:**
  - **Issue Found:** Total 125 issues were prompted in the for 3 contracts [PublicGovFactory.sol, PublicGovSale.sol, TransferHelper.sol] and most of them was found false positive.
  - The True positive issue is already covered in manual analysis.



Closing Summary

Description about Project overall posture and talk about risk associated with Project.

About Secureverse

Secureverse is the emerging Web3 Security solution provider. We at secureverse provides the Smart Contract audit, Blockchain infrastructure Penetration testing and the Cryptocurrency forensic services with very affordable prices.

To Know More

Twitter: https://twitter.com/secureverse

LinkedIn: https://www.linkedin.com/company/secureverse/

Telegram: https://t.me/secureverse

Email Address: http://secureverse@protonmail.com/