# Full Audit Report

## Jeti Services Security Assessment

Real Cybersecurity
Protecting digital assets

Made in Thailand

SECURI LAB
(THAILAND) contact@securi-lab.com

Audit Report

# Table of Contents 1

FULL AUDIT REPORT

# Report Information

| | |
|---|---|
| About Report | **Jeti Services Security Assessment** |
| Version | **v1.1** |
| Client | **Jeti Service** |
| Language | **Solidity** |
| Confidentiality | **Public** |
| Contract File | **JSRVGovernanceFactory.sol**<br>SHA-1: bbee4fcd8b5ae5689609640c3b64ad3733a5c2f0<br>**JSRVGovernance.sol**<br>SHA-1: 4cbebeb29afeec54528d233b9e6e0509e1474f44<br>**Blacklist.sol**<br>SHA-1: e3022a71f8a6a09a0b79f7f2f1c2358bef d6b36c |
| Audit Method | **Whitebox** |
| Security Assessment Author | **Auditor**<br><br>**Mark K. [Security Researcher \| Redteam]**<br>**Kevin N. [Security Researcher \| Web3 Dev]**<br>**Yusheng T. [Security Researcher \| Incident Response]**<br><br>**Approve Document**<br>**Ronny C. CTO & Head of Security Researcher**<br>**Chinnakit J. CEO & Founder** |

*Audit Method

**Whitebox:** SECURI LAB Team receives all source code from the client to provide the assessment.
**Blackbox:** SECURI LAB Team receives only bytecode from the client to provide the assessment.

**Digital Sign (Only Full Audit Report)**

Digitally signed by Chinnakit J.
DN: cn=Chinnakit J., o=SECURI LAB (Thailand), ou=CEO & Founder, email=contact@securi-lab.com, c=TH
Date: 2023.04.14 19:50:31 +07'00'

# Disclaimer

Regarding this security assessment, there are no guarantees about the security of the program instruction received from the client is hereinafter referred to as "**Source code**".
And **SECURI Lab** hereinafter referred to as "**Service Provider**", the **Service Provider** will not be held liable for any legal liability arising from errors in the security assessment. The responsibility will be the responsibility of the **Client**, hereinafter referred to as "**Service User**" and the **Service User** agrees not to be held liable to the **service provider** in any case. By contract **Service Provider** to conduct security assessments with integrity with professional ethics, and transparency to deliver security assessments to users The **Service Provider** has the right to postpone the delivery of the security assessment. If the security assessment is delayed whether caused by any reason and is not responsible for any delayed security assessments.
If **the service provider** finds a vulnerability The **service provider** will notify the **service user** via the Preliminary Report, which will be kept confidential for security. The **service provider** disclaims responsibility in the event of any attacks occurring whether before conducting a security assessment. Or happened later All responsibility shall be sole with the **service user**.

**Security Assessment Not Financial/Investment Advice Any loss arising from any investment in any project is the responsibility of the investor.**

**SECURI LAB disclaims any liability incurred. Whether it's Rugpull, Abandonment, Soft Rugpull**

The SECURI LAB team has conducted a comprehensive security assessment of the vulnerabilities.
This assessment is tested with an expert assessment. Using the following test requirements

1. Smart Contract Testing with Expert Analysis By testing the most common and uncommon vulnerabilities.
2. Automated program testing It includes a sample vulnerability test and a sample of the potential vulnerabilities being used for the most frequent attacks.
3. Manual Testing with AST/WAS/ASE/SMT and reviewed code line by line
4. Visibility, Mutability, Modifier function testing, such as whether a function can be seen in general, or whether a function can be changed and if so, who can change it.
5. Function association test It will be displayed through the association graph.
6. This safety assessment is cross-checked prior to the delivery of the assessment results.

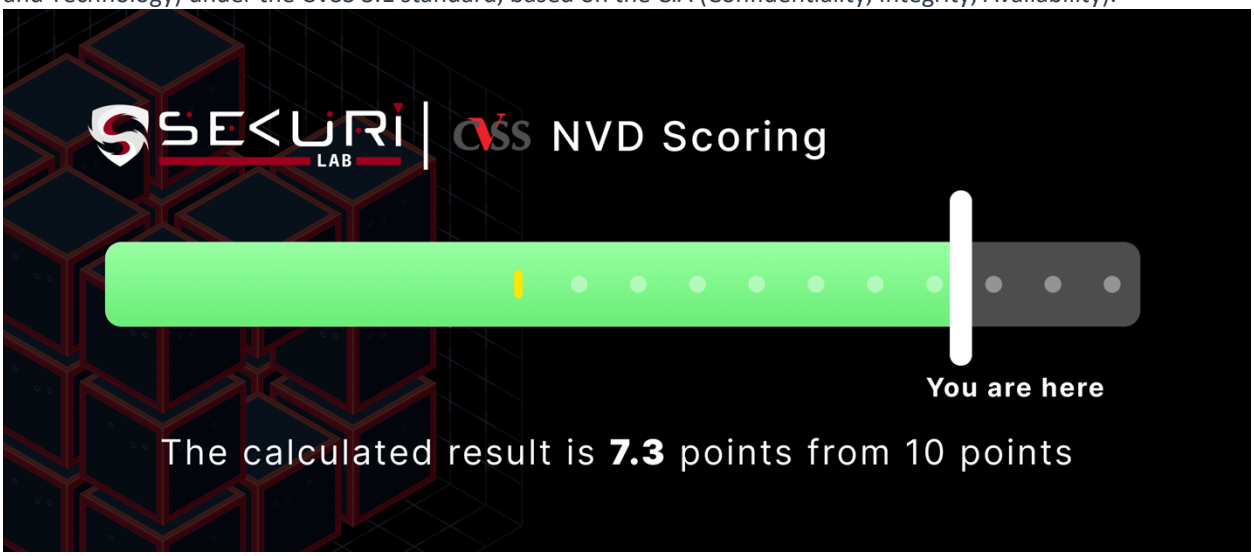
FULL AUDIT REPORT

# Executive Summary

For this security assessment, SECURI LAB received a request from Jeti Services on Tuesday, March 21, 2023.

## NVD CVSS Scoring

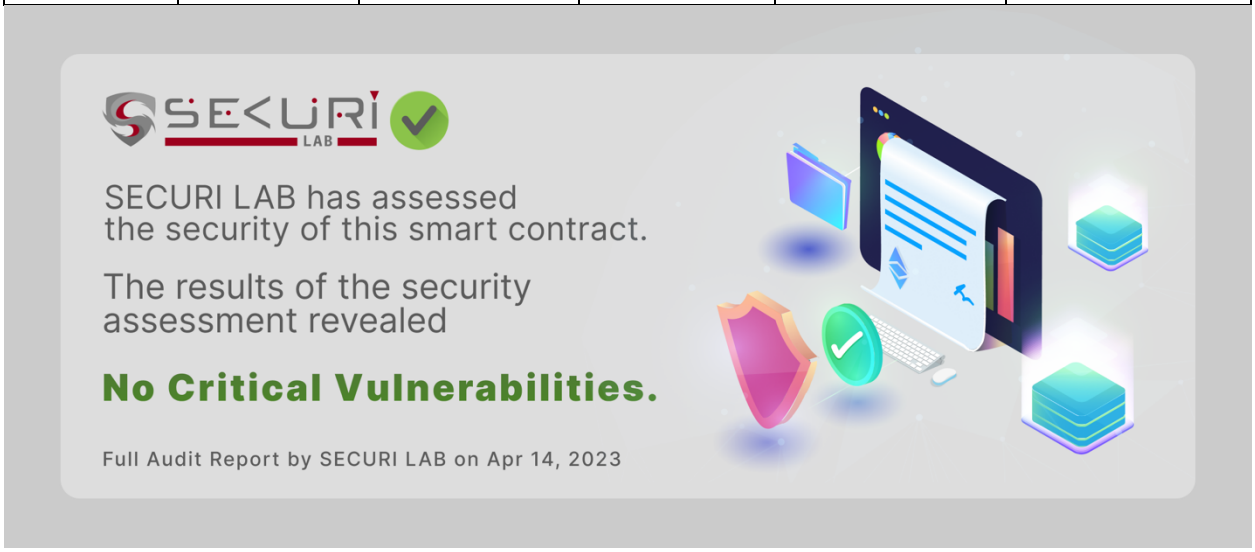
The score was calculated using the NVD (National Vulnerability Database) of NIST (National Institute of Standards and Technology) under the CVSS 3.1 standard, based on the CIA (Confidentiality, Integrity, Availability).




## Audit Result

SECURI LAB evaluated the smart contract security of the project and found: **[Total : 9]**

| Critical | High | Medium | Low | Very Low | Informational |
|---|---|---|---|---|---|
| 0 | 2 | 0 | 2 | 0 | 5 |



# FULL AUDIT REPORT

## Project Introduction

**Scope Information:**

| | |
|---|---|
| Project Name | **Jeti Services** |
| Website | **https://jeti.one/** |
| Chain | **-** |
| Language | **Solidity** |

## Audit Information:

| | |
|---|---|
| Request Date | **Tuesday, March 21, 2023** |
| Audit Date | **Wednesday, March 29, 2023** |
| Re-assessment Date | **Sunday, April 9, 2023** |

## Audit Version History:

| Version | Date | Description |
|---|---|---|
| **1.0** | **Wednesday, March 29, 2023** | **Preliminary Report** |
| **1.1** | **Sunday, April 9, 2023** | **Preliminary Report With Re-assessment** |
| **1.2** | **Friday, April 14, 2023** | **Full audit report** |

## FULL AUDIT REPORT

### Initial Audit Scope:

| | |
|---|---|
| Smart Contract File | **JSRVGovernanceFactory.sol**<br>SHA-1: cd9db78ffdc6b07da3bc5398189eb76dc55ed98f<br>**JSRVGovernance.sol**<br>SHA-1: 33e8e2ce9bfd191cf6f5fb8f0367b494ab086076<br>**Blacklist.sol**<br>SHA-1: e3022a71f8a6a09a0b79f7f2f1c2358befd6b36c |
| Compiler Version | **v0.8.17** |

## Source Units in Scope

Source Units Analyzed: 3
Source Units in Scope: 3 (**100%**)

| Type | File | Logic Contracts | Interfaces | Lines | nLines | nSLOC | Comment Lines | Complex. Score | Capabilities |
|---|---|---|---|---|---|---|---|---|---|
| 📝 🔍 🎨 | contracts/Tokens/JSRVGovernance.sol | 2 | 1 | 618 | 580 | 449 | 2 | 336 | 💰📩 |
| 📝 | contracts/Blacklist.sol | 1 | ———— | 139 | 139 | 96 | 3 | 48 | ———— |
| 📝 🔍 | contracts/JSRVGovernanceFactory.sol | 1 | 1 | 102 | 91 | 74 | 1 | 65 | 💰📩 🌀 |
| 📝 🔍 🎨 | **Totals** | **4** | **2** | **859** | **810** | **619** | **6** | **449** | 💰📩 🌀 |

Legend: [▔]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

## Dependencies / External Imports

| Dependency / Import Path | Count |
|---|---|
| @openzeppelin/contracts/access/Ownable.sol | 3 |
| @openzeppelin/contracts/security/Pausable.sol | 1 |
| @openzeppelin/contracts/security/ReentrancyGuard.sol | 2 |
| @openzeppelin/contracts/token/ERC20/IERC20.sol | 1 |
| @openzeppelin/contracts/utils/Address.sol | 2 |
| @openzeppelin/contracts/utils/Counters.sol | 1 |
| @openzeppelin/contracts/utils/Strings.sol | 1 |
| @openzeppelin/contracts/utils/math/SafeMath.sol | 1 |

Description Report Files Description Table

| File Name | SHA-1 Hash |
|---|---|
| contracts/Tokens/JSRVGovernance.sol | 33e8e2ce9bfd191cf6f5fb8f0367b494ab086076 |
| contracts/Blacklist.sol | e3022a71f8a6a09a0b79f7f2f1c2358befd6b36c |
| contracts/JSRVGovernanceFactory.sol | cd9db78ffdc6b07da3bc5398189eb76dc55ed98f |

FULL AUDIT REPORT

**Initial Re-assessment Audit Scope:**

| | |
|---|---|
| Smart Contract File | **JSRVGovernanceFactory.sol**<br>SHA-1: bbee4fcd8b5ae5689609640c3b64ad3733a5c2f0<br>**JSRVGovernance.sol**<br>SHA-1: 4cbebeb29afeec54528d233b9e6e0509e1474f44<br>**Blacklist.sol**<br>SHA-1: e3022a71f8a6a09a0b79f7f2f1c2358befd6b36c |
| Compiler Version | **v0.8.17** |

## Source Units in Scope

Source Units Analyzed: 3
Source Units in Scope: 3 (**100%**)

| Type | File | Logic Contracts | Interfaces | Lines | nLines | nSLOC | Comment Lines | Complex. Score | Capabilities |
|---|---|---|---|---|---|---|---|---|---|
| 📝🔍🎨 | contracts/Tokens/JSRVGovernance.sol | 2 | 1 | 711 | 672 | 520 | 2 | 373 | 💰📩 |
| 📝 | contracts/Blacklist.sol | 1 | | 139 | 139 | 96 | 3 | 48 | |
| 📝🔍 | contracts/JSRVGovernanceFactory.sol | 1 | 1 | 122 | 111 | 89 | 1 | 76 | 💰📩🌀 |
| 📝🔍🎨 | **Totals** | **4** | **2** | **972** | **922** | **705** | **6** | **497** | 💰📩🌀 |

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

## Dependencies / External Imports

| Dependency / Import Path | Count |
|---|---|
| @openzeppelin/contracts/access/Ownable.sol | 3 |
| @openzeppelin/contracts/security/Pausable.sol | 1 |
| @openzeppelin/contracts/security/ReentrancyGuard.sol | 2 |
| @openzeppelin/contracts/token/ERC20/IERC20.sol | 1 |
| @openzeppelin/contracts/utils/Address.sol | 2 |
| @openzeppelin/contracts/utils/Counters.sol | 1 |
| @openzeppelin/contracts/utils/Strings.sol | 1 |
| @openzeppelin/contracts/utils/math/SafeMath.sol | 1 |

Description Report Files Description Table

| File Name | SHA-1 Hash |
|---|---|
| contracts/JSRVGovernanceFactory.sol | bbee4fcd8b5ae5689609640c3b64ad3733a5c2f0 |
| contracts/Blacklist.sol | e3022a71f8a6a09a0b79f7f2f1c2358befd6b36c |
| contracts/Tokens/JSRVGovernance.sol | 4cbebeb29afeec54528d233b9e6e0509e1474f44 |

## Security Assessment Procedure

Securi has the following procedures and regulations for conducting security assessments:

**1.Request Audit** Client submits a form request through the Securi channel. After receiving the request, Securi will discuss a security assessment. And drafting a contract and agreeing to sign a contract together with the Client

**2.Auditing** Securi performs security assessments of smart contracts obtained through automated analysis and expert manual audits.

**3.Preliminary Report** At this stage, Securi will deliver an initial security assessment. To report on vulnerabilities and errors found under Audit Scope will not publish preliminary reports for safety.

**4.Reassessment** After Securi has delivered the Preliminary Report to the Client, Securi will track the status of the vulnerability or error, which will be published to the Final Report at a later date with the following statuses:

- **a.Acknowledge** The client has been informed about errors or vulnerabilities from the security assessment.
- **b.Resolved** The client has resolved the error or vulnerability. Resolved is probably just a commit, and Securi is unable to verify that the resolved has been implemented or not.
- **c.Decline** Client has rejected the results of the security assessment on the issue.

**5.Final Report** Securi providing full security assessment report and public

## Risk Rating

Risk rating using this commonly defined: $Risk\ rating\ =\ impact\ *\ confidence$

**Impact** The severity and potential impact of an attacker attack

**Confidence** Ensuring that attackers expose and use this vulnerability

Both have a total of 3 levels: **High**, **Medium**, **Low**. By *Informational* will not be classified as a level

| Confidence<br>Impact<br>[Likelihood] | Low | Medium | High |
|---|---|---|---|
| Low | **Very Low** | **Low** | **Medium** |
| Medium | **Low** | **Medium** | **High** |
| High | **Medium** | **High** | **Critical** |

# Vulnerability Severity Summary

**Severity** is a risk assessment It is calculated from the Impact and Confidence values using the following calculation methods,

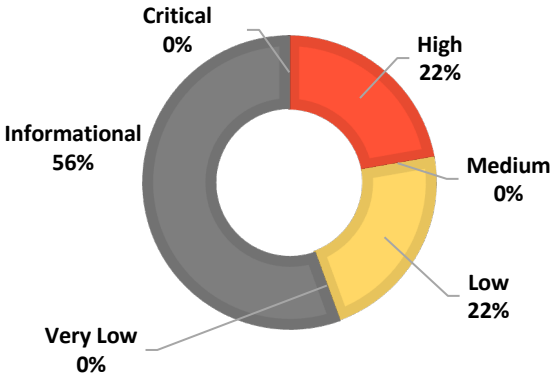
$Risk\ rating\ =\ impact\ *\ confidence$

It is categorized into

**5 categories based** on the **lowest severity**:

Very Low , Low , Medium , **High** , Critical .

For **Informational** & will **Non-class/Optimization/Best-practices will** not be counted as **severity**




| Vulnerability Severity Level | Total |
|---|---|
| Critical | 0 |
| High | 2 |
| Medium | 0 |
| Low | 2 |
| Very Low | 0 |
| Informational | 5 |
| Non-class/Optimization/Best-practices | 5 |

**Category information:**

Centralization

**Centralization Risk** is The risk incurred by a sole proprietor, such as the Owner being able to change something without permission

Economics Risk

**Economics Risk** is Risks that may affect the economic mechanism system, such as the ability to increase Mint token

Logical Issue

**Logical Issue** is that can cause errors to core processing, such as any prior operations that cause background processes to crash.

Authorization

**Authorization** is Possible pitfalls from weak coding allows unrelated people to take any action to modify the values.

Mathematical

**Mathematical** Any erroneous arithmetic operations affect the operation of the system or lead to erroneous values.

Naming Conventions

**Naming Conventions** naming variables that may affect code understanding or naming inconsistencies

Security Risk

**Security Risk** of loss or damage if it's no mitigate

Coding Style

**Coding Style** is Tips coding for efficiency performance

Best Practices

**Best Practices** is suggestions for improvement

Optimization

**Optimization** is performance improvement

Gas Optimization

**Gas Optimization** is increase performance to avoid expensive gas

Dead Code

**Dead Code** having unused code This may result in wasted resources and gas fees.

FULL AUDIT REPORT

## Vulnerability Findings

| ID | Vulnerability Detail | Severity | Category | Status |
|---|---|---|---|---|
| SEC-01 | Centralization Risk | High | Centralization | **Resolved** |
| SEC-02 | Contract's name reused | High | Naming Conventions | **Resolved** |
| SEC-03 | Avoid using block timestamp | Low | Best Practices | **Acknowledge** |
| SEC-04 | Unsafe ERC20 operation(s) | Low | Best Practices | **Acknowledge** |
| SEC-05 | Conformance to numeric notation best practices (too-many-digits) | Informational | Best Practices | **Resolved** |
| SEC-06 | Conformance to numeric notation best practices | Informational | Naming Conventions | **Acknowledge** |
| SEC-07 | Costly operations in a loop (costly-loop) | Informational | Optimization | **Acknowledge** |
| SEC-08 | If different pragma directives are used (pragma) | Informational | Best Practices | **Acknowledge** |
| SEC-09 | Reentrancy vulnerabilities through send and transfer (reentrancy-unlimited-gas) | Informational | Security Risk | **Acknowledge** |
| GAS-01 | Use `selfbalance()` instead of `address(this).balance` | - | Gas Optimization | **Acknowledge** |
| GAS-02 | Use assembly to check for `address(0)` | - | Gas Optimization | **Resolved** |
| GAS-03 | `array[index] += amount` is cheaper than `array[index] = array[index] + amount` (or related variants) | - | Gas Optimization | **Resolved** |
| GAS-04 | Use Custom Errors | - | Gas Optimization | **Resolved** |
| GAS-05 | Use != 0 instead of > 0 for unsigned integer comparison | - | Gas Optimization | **Resolved** |

# SEC-01: Centralization Risk

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Centralization Risk | High | Check on finding | Centralization | **Resolved** |

## Finding:

```
File: Blacklist.sol

8: contract Blacklist is Ownable {

114:     function addBlacklistAdmins(address _blacklistAdminsAddress, address
_allowedContracts) public onlyOwner  {

122:     function removeBlacklistAdmins(address _blacklistAdminsAddress, address
_allowedContracts) public onlyOwner  {

130:     function setAdmin(address _admin) public onlyOwner  {

```
File: JSRVGovernanceFactory.sol

14: contract JSRVGovernanceFactory is ReentrancyGuard, Ownable {

39:   function setFeeTo(address feeReceivingAddress) external onlyOwner {

43:   function setFlatFee(uint256 fee) external onlyOwner {

54:   function newBlacklistContract(address _newBlacklist) public onlyOwner {

58:   function newTokenFeeTo(address _newTokenFeeTo) public onlyOwner {

62:   function newTokenFee(uint256 _newTokenFee) public onlyOwner {

File: Tokens/JSRVGovernance.sol

24: contract JSRVGovernanceToken is IERC20, Ownable, BaseToken, Pausable,
ReentrancyGuard {

180:     function toggleRecycle() public onlyOwner {

184:      function pause() public onlyOwner {
```
```

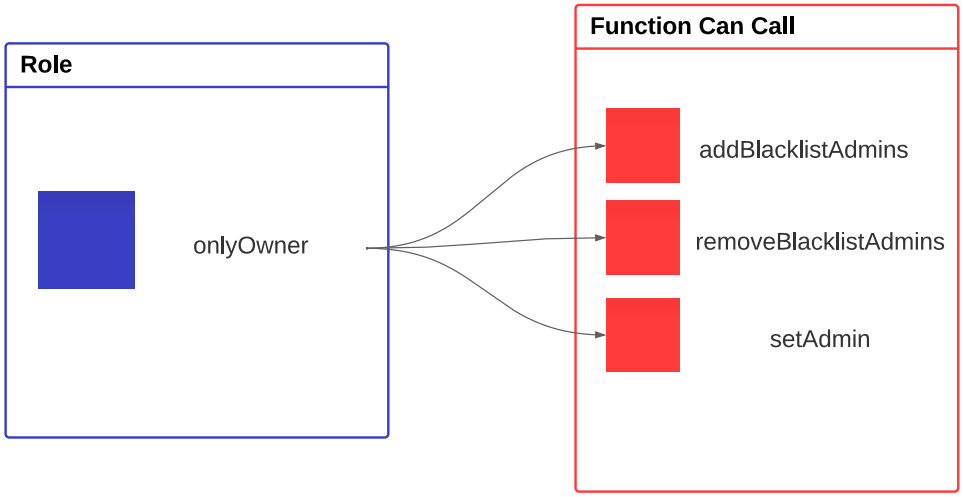
```
188:     function unpause() public onlyOwner {

519:     function depositProfitShare() public payable onlyOwner {

553:     function _setNewRecycleRate(uint256 _newRecycleRate) public onlyOwner {

559:     function _setNewRates(uint256[3] memory _newRate) public onlyOwner {

565:     function _setNewRedeemRate(uint256 _newRedeemRate) public onlyOwner {

571:     function _setNewThreshold(uint256[3] memory _newThreshold) public onlyOwner {

577:     function _setNewMinPurchase(uint256 _newMinPurchase) public onlyOwner {

583:     function _setNewMaxPurchase(uint256 _newMaxPurchase) public onlyOwner {

589:     function _setNewMaxHold(uint256 _newMaxHold) public onlyOwner {

595:     function _setBlacklistAddress(address blacklistAddress_) external onlyOwner {

609:     function withdrawExcess() public payable onlyOwner {
```

**Scenario:**

Centralized risk refers to the potential security risks that arise when a smart contract is controlled by a central entity or a single point of failure. If the contract is controlled by a central authority, then the contract may be vulnerable to attacks that target the centralized entity.

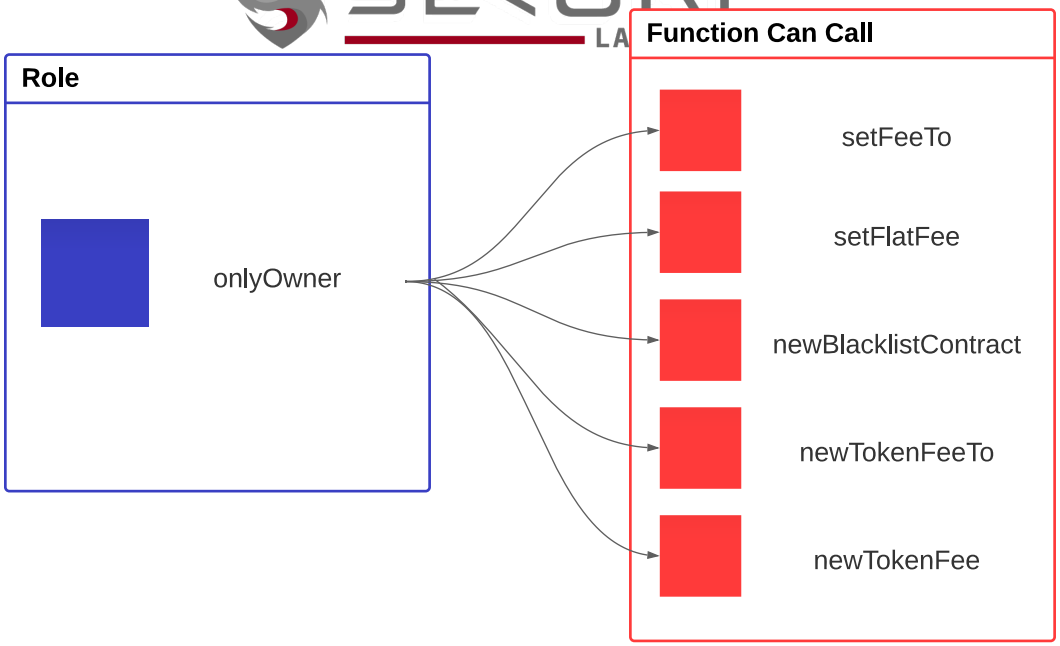
Centralized risk that can lead to rug pulls typically arises from the centralization of control or ownership of a project's assets, particularly in decentralize d finance (DeFi) projects built on blockchain platforms like Ethereum.

Contract Blacklist (File: Blacklist.sol)




The aforementioned function in the Blacklist contract can only be invoked by the onlyOwner. This contract permits calling of addBlacklistAdmins, removeBlacklistAdmins, and setAdmin functions. Additionally, the implementation of a multi-signature feature adds another layer of security to safeguard the owner's account.

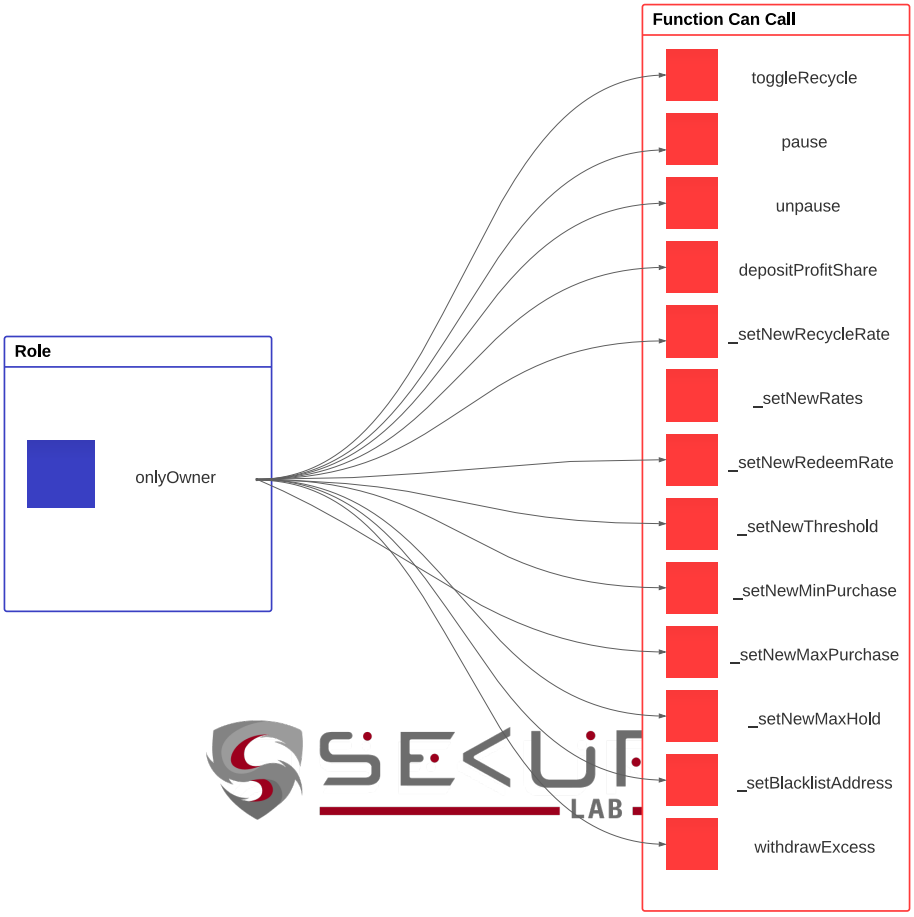
Contract JSRVGovernanceFactory (File: JSRVGovernanceFactory.sol)




In the JSRVGovernanceFactory contract, Owner can call functions setFeeTo, setFlatFee, newBlacklistContract, newTokenFeeto, newTokenFee. Additionally, the implementation of a multi-signature feature adds another layer of security to safeguard the owner's account.

## FULL AUDIT REPORT

Contract JSRVGovernanceToken (File: Tokens/JSRVGovernance.sol)




In the JSRVGovernanceFactory contract, Owner can call functions toggleRecycle, pause, unpause, depositProfitShare, _setNewRecycleRate, _setNewRates, _setNewRedeemRate, _setNewThreshold, _setNewMinPurchase, _setNewMaxPurchase, _setNewMaxHold, _setBlacklistAddress, withdrawExcess. We've found that some functions work in an anti-whale manner and allow the owner to pause trading. Assigning a backlist address and also another function we recommend that for transparency use Timelock to increase the delay for users. Function calls are visible before they are fully executed. Additionally, the implementation of a multi-signature feature adds another layer of security to safeguard the owner's account.

**Recommendation:**

In terms of timeframes, there are three categories: short-term, long-term, and permanent.

For short-term solutions, a combination of timelock and multi-signature (2/3 or 3/5) can be used to mitigate risk by delaying sensitive operations and avoiding a single point of failure in key management. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; assigning privileged roles to multi-signature wallets to prevent private key compromise; and sharing the timelock contract and multi-signer addresses with the public via a medium/blog link.

For long-term solutions, a combination of timelock and DAO can be used to apply decentralization and transparency to the system. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; introducing a DAO/governance/voting module to increase transparency and user involvement; and sharing the timelock contract, multi-signer addresses, and DAO information with the public via a medium/blog link.

Finally, permanent solutions should be implemented to ensure the ongoing security and protection of the system.

**Alleviation:**

Jeti team has already resolved this issue

## SEC-02: Contract's name reused (name-reused)

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Contract's name reused (name-reused) | High | Check on finding | Naming Conventions | **Resolved** |

### Finding:

```
iBlacklist is re-used:
    • iBlacklist (Tokens/JSRVGovernance.sol:14-16)
    • iBlacklist (JSRVGovernanceFactory.sol#10-12)
```

### Recommendation:

Rename the contract.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused

### Exploit Scenario:

If a codebase has two contracts the similar names, the compilation artifacts will not contain one of the contracts with the duplicate name.

Bob's truffle codebase has two contracts named ERC20. When truffle compile runs, only one of the two contracts will generate artifacts in build/contracts. As a result, the second contract cannot be analyzed.

### Alleviation:

Jeti team has already resolved this issue

# SEC-03: Avoid using block timestamp

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Avoid using block timestamp | Low | Check on finding | Best Practices | **Acknowledge** |

## Finding:

```
37:         uint256 blacklistStart = block.timestamp;

68:         uint256 remainTime = blacklist.blacklistEnd > block.timestamp ?
blacklist.blacklistEnd-block.timestamp : 0;

68:         uint256 remainTime = blacklist.blacklistEnd > block.timestamp ?
blacklist.blacklistEnd-block.timestamp : 0;

78:         uint256 remainTime = blacklist.blacklistEnd > block.timestamp ?
blacklist.blacklistEnd-block.timestamp : 0;

78:         uint256 remainTime = blacklist.blacklistEnd > block.timestamp ?
blacklist.blacklistEnd-block.timestamp : 0;

```
```
```

## Recommendation:

Avoid relying on `block.timestamp`.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

## Exploit Scenario:

Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.

"Bob's contract relies on block.timestamp for its randomness. Eve is a miner and manipulates block.timestamp to exploit Bob's contract.

## Alleviation:

Jeti team has already Acknowledge this issue

# SEC-04: Unsafe ERC20 operation(s)

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Unsafe ERC20 operation(s) | Low | Check on finding | Best Practices | **Acknowledge** |

## Finding:

```
File: JSRVGovernanceFactory.sol

96:     payable(feeTo).transfer(flatFee);

```
File: Tokens/JSRVGovernance.sol

512:         payable(msg.sender).transfer(profitShare);

613:         payable(msg.sender).transfer(balance);

```
```

## Recommendation:

Unsafe ERC20 operations can lead to unexpected behavior and potential vulnerabilities in your smart contracts. To mitigate these risks, consider the following recommendations for safer ERC20 operations:

1. Check the return value of ERC20 functions:

Always check the return value of ERC20 functions like **transfer**, **transferFrom**, and **approve**. These functions return a boolean value that indicates whether the operation was successful or not. Make sure to handle potential failures accordingly.

## Exploit Scenario:

-

## Alleviation:

Jeti team has already Acknowledge this issue

# SEC-05: Conformance to numeric notation best practices

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Conformance to numeric notation best practices (too-many-digits) | Informational | Check on finding | Best Practices | **Resolved** |

## Finding:

```
JSRVGovernanceFactory.constructor(address) (JSRVGovernanceFactory.sol:32-37) uses
literals with too many digits:
    • flatFee = 10000000000000000 (JSRVGovernanceFactory.sol#36)
```

## Recommendation:

Description
Literals with many digits are difficult to read and review.
Use:
- [Ether suffix](https://solidity.readthedocs.io/en/latest/units-and-global-variables.html#ether-units),
- [Time suffix](https://solidity.readthedocs.io/en/latest/units-and-global-variables.html#time-units), or
- [The scientific notation](https://solidity.readthedocs.io/en/latest/types.html#rational-and-integer-literals)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

## Exploit Scenario:

```
contract MyContract{
   uint 1_ether = 10000000000000000000;
}
```

While 1_ether looks like 1 ether, it is 10 ether. As a result, it's likely to be used incorrectly.

## Alleviation:

Jeti team has already resolved this issue and changed code to
**flatFee[0] = 0.01 * 10**18 wei;**
**flatFee[1] = 0.01 * 10**18 wei;**

# SEC-06: Conformity to Solidity naming conventions (naming-convention)

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Conformity to Solidity naming conventions (naming-convention) | Informational | Check on finding | Naming Conventions | **Acknowledge** |

## Finding:

```
Function JSRVGovernanceToken._mint(uint256) (Tokens/JSRVGovernance.sol:294-327) is not
in mixedCase
Function JSRVGovernanceToken._recycleMint(uint256) (Tokens/JSRVGovernance.sol:329-351)
is not in mixedCase
Function JSRVGovernanceToken._setBlacklistAddress(address)
(Tokens/JSRVGovernance.sol:595-601) is not in mixedCase
Function JSRVGovernanceToken._setNewMaxHold(uint256) (Tokens/JSRVGovernance.sol:589-
593) is not in mixedCase
Function JSRVGovernanceToken._setNewMaxPurchase(uint256)
(Tokens/JSRVGovernance.sol:583-587) is not in mixedCase
Function JSRVGovernanceToken._setNewMinPurchase(uint256)
(Tokens/JSRVGovernance.sol:577-581) is not in mixedCase
Function JSRVGovernanceToken._setNewRates(uint256[3]) (Tokens/JSRVGovernance.sol:559-
563) is not in mixedCase
Function JSRVGovernanceToken._setNewRecycleRate(uint256)
(Tokens/JSRVGovernance.sol:553-557) is not in mixedCase
Function JSRVGovernanceToken._setNewRedeemRate(uint256)
(Tokens/JSRVGovernance.sol:565-569) is not in mixedCase
Function JSRVGovernanceToken._setNewThreshold(uint256[3])
(Tokens/JSRVGovernance.sol:571-575) is not in mixedCase
```

## Recommendation:

Follow the Solidity [naming convention](https://solidity.readthedocs.io/en/v0.4.25/style-guide.html#naming-conventions).

## Exploit Scenario:

-

## Alleviation:

Jeti team has already Acknowledge this issue

# SEC-07: Costly operations in a loop (costly-loop)

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Costly operations in a loop (costly-loop) | Informational | Check on finding | Optimization | **Acknowledge** |

## Finding:

```
JSRVGovernanceToken.depositProfitShare() (Tokens/JSRVGovernance.sol:519-545) has
costly operations inside a loop:
    • _lastDeposited = i (Tokens/JSRVGovernance.sol#530)
```

## Recommendation:

Use a local variable to hold the loop computation result.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

## Exploit Scenario:

Costly operations inside a loop might waste gas, so optimizations are justified.

```
contract CostlyOperationsInLoop{

  uint loop_count = 100;
  uint state_variable=0;

  function bad() external{
    for (uint i=0; i < loop_count; i++){
      state_variable++;
    }
  }

  function good() external{
   uint local_variable = state_variable;
   for (uint i=0; i < loop_count; i++){
    local_variable++;
   }
   state_variable = local_variable;
  }
}
```

Incrementing state_variable in a loop incurs a lot of gas because of expensive SSTOREs, which might lead to an out-of-gas.

## Alleviation:

Jeti team has already Acknowledge this issue

# SEC-08: If different pragma directives are used (pragma)

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| If different pragma directives are used (pragma) | Informational | Check on finding | Best Practices | **Acknowledge** |

## Finding:

```
Different versions of Solidity are used:
    • Version used: ['0.8.17', '=0.8.17', '^0.8.0', '^0.8.1']
    • 0.8.17 (Tokens/JSRVGovernance.sol:2)
    • 0.8.17 (Blacklist.sol#2)
    • =0.8.17 (JSRVGovernanceFactory.sol#2)
    • ^0.8.0 (@openzeppelin/contracts/utils/math/Math.sol#4)
    • ^0.8.0 (@openzeppelin/contracts/utils/math/SafeMath.sol#4)
    • ^0.8.0 (@openzeppelin/contracts/security/ReentrancyGuard.sol#4)
    • ^0.8.0 (@openzeppelin/contracts/utils/Context.sol#4)
    • ^0.8.0 (@openzeppelin/contracts/utils/Strings.sol#4)
    • ^0.8.0 (@openzeppelin/contracts/security/Pausable.sol#4)
    • ^0.8.0 (@openzeppelin/contracts/utils/Counters.sol#4)
    • ^0.8.0 (@openzeppelin/contracts/access/Ownable.sol#4)
    • ^0.8.0 (@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
    • ^0.8.1 (@openzeppelin/contracts/utils/Address.sol#4)
```

## Recommendation:

Use one Solidity version.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

## Exploit Scenario:

-

## Alleviation:

Jeti team has already Acknowledge this issue

# SEC-09: Reentrancy vulnerabilities through send and transfer (reentrancy-unlimited-gas)

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Reentrancy vulnerabilities through send and transfer (reentrancy-unlimited-gas) | Informational | Check on finding | Security Risk | **Acknowledge** |

## Finding:

```
Reentrancy in
JSRVGovernanceFactory.create(string[2],uint256,uint8,uint8,uint256[6],uint256[3],uint2
56[6]) (JSRVGovernanceFactory.sol:67-101):
    • address(feeTo).transfer(flatFee) (JSRVGovernanceFactory.sol#96)
    • refundExcessiveFee() (JSRVGovernanceFactory.sol#80)
    • (success) = recipient.call{value: amount}()
(@openzeppelin/contracts/utils/Address.sol#63)
    • address(feeTo).transfer(flatFee) (JSRVGovernanceFactory.sol#96)
    • TokenCreated(address(newToken),type_) (JSRVGovernanceFactory.sol#98)
Reentrancy in JSRVGovernanceToken.withdrawExcess() (Tokens/JSRVGovernance.sol:609-
616):
    • address(msg.sender).transfer(balance) (Tokens/JSRVGovernance.sol#613)
    • WithdrawExcessBalance(balance) (Tokens/JSRVGovernance.sol#615)
Reentrancy in JSRVGovernanceToken.redeemProfitShare() (Tokens/JSRVGovernance.sol:460-
517):
    • address(msg.sender).transfer(profitShare) (Tokens/JSRVGovernance.sol#512)
    • transfer(address(this),redeemTokens) (Tokens/JSRVGovernance.sol#514)
    • holder._tokens = holder._tokens + addAmount (Tokens/JSRVGovernance.sol#396)
    • holder._tokens = 0 (Tokens/JSRVGovernance.sol#407)
    • holder._tokens = holder._tokens - remain (Tokens/JSRVGovernance.sol#409)
    • transfer(address(this),redeemTokens) (Tokens/JSRVGovernance.sol#514)
    • _holderLastPurchased[_address] = purchaseRound (Tokens/JSRVGovernance.sol#399)
    • _holderLastRedeemed[msg.sender] = tempLastRedeem (Tokens/JSRVGovernance.sol#516)
    • transfer(address(this),redeemTokens) (Tokens/JSRVGovernance.sol#514)
    • _tOwned[recipient] = _tOwned[recipient] + amount (Tokens/JSRVGovernance.sol#260)
    • _tOwned[msg.sender] = _tOwned[msg.sender] - amount
(Tokens/JSRVGovernance.sol#261)
    • _totalAvailable = _totalAvailable - profitShare (Tokens/JSRVGovernance.sol#513)
    • Transfer(_msgSender(),recipient,amount) (Tokens/JSRVGovernance.sol#259)
    • transfer(address(this),redeemTokens) (Tokens/JSRVGovernance.sol#514)
    • TransferEvent(msg.sender,recipient,amount) (Tokens/JSRVGovernance.sol#265)
    • transfer(address(this),redeemTokens) (Tokens/JSRVGovernance.sol#514)
```

## Recommendation:

Apply the [`check-effects-interactions` pattern](http://solidity.readthedocs.io/en/v0.4.21/security-considerations.html#re-entrancy).

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

## Exploit Scenario:

```
function callme(){
    msg.sender.transfer(balances[msg.sender]):
    balances[msg.sender] = 0;
}
```

send and transfer do not protect from reentrancies in case of gas price changes.

## Alleviation:

Jeti team has already Acknowledge this issue

## GAS-01: Use `selfbalance()` instead of `address(this).balance`

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Use `selfbalance()` instead of `address(this).balance` | - | Check on finding | Gas Optimization | **Acknowledge** |

### Finding:

```
File: Tokens/JSRVGovernance.sol

510:         require(profitShare <= address(this).balance, "Not enought funds to
withdraw");

610:         require(address(this).balance > _totalAvailable, "You have no excesses in
here");

611:         uint256 balance  = address(this).balance - _totalAvailable;
```

### Recommendation:

Use assembly when getting a contract's balance of ETH.

You can use `selfbalance()` instead of `address(this).balance` when getting your contract's balance of ETH to save gas.
Additionally, you can use `balance(address)` instead of `address.balance()` when getting an external contract's balance of ETH.

*Saves 15 gas when checking internal balance, 6 for external*

### Alleviation:

Jeti team has already Acknowledge this issue

## GAS-02: Use assembly to check for `address(0)

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Use assembly to check for `address(0) | - | Check on finding | Gas Optimization | **Resolved** |

### Finding:

```
244:        require(owner != address(0), "ERC20: approve from the zero address");

245:        require(spender != address(0), "ERC20: approve to the zero address");

```
```
```

### Recommendation:

*Saves 6 gas per instance*

*Instances (2)*:

### Alleviation:

Jeti team has already Resolved this issue

## GAS-03: `array[index] += amount` is cheaper than `array[index] = array[index] + amount` (or related variants)

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| `array[index] += amount` is cheaper than `array[index] = array[index] + amount` (or related variants) | - | Check on finding | Gas Optimization | **Resolved** |

### Finding:

```
File: Tokens/JSRVGovernance.sol

142:          _totalSoldTokens[1] = _totalSoldTokens[1] +  _tokenFee;

260:        _tOwned[recipient] = _tOwned[recipient] + amount;

261:        _tOwned[msg.sender] = _tOwned[msg.sender] - amount;

284:        _tOwned[recipient] = _tOwned[recipient] + amount;

285:        _tOwned[sender] = _tOwned[sender] - amount;

316:        _tOwned[msg.sender] = _tOwned[msg.sender] + (roundAmount[0] +
roundAmount[1] + roundAmount[2]);

317:        _tOwned[address(this)] = _tOwned[address(this)] - (roundAmount[0] +
roundAmount[1] + roundAmount[2]);

321:        _totalSoldTokens[purchaseRound] = _totalSoldTokens[purchaseRound] +
(roundAmount[0] + roundAmount[1] + roundAmount[2]);

337:        _tOwned[msg.sender] = _tOwned[msg.sender] + _amount;

338:        _tOwned[address(this)] = _tOwned[address(this)] - _amount;

346:        _totalSoldTokens[purchaseRound] = _totalSoldTokens[purchaseRound] +
_amount;

498:                            _redeemedProfitShare[i] = _redeemedProfitShare[i] +
tempProfitShareValue;
```

```
528:                    _availableProfitShare[i] = _availableProfitShare[i] +
depositedAmount;

```

## Recommendation:

When updating a value in an array with arithmetic, using `array[index] += amount` is cheaper than `array[index] = array[index] + amount`.
This is because you avoid an additonal `mload` when the array is stored in memory, and an `sload` when the array is stored in storage.
This can be applied for any arithmetic operation including `+=`, `-=`,`/=`,`*=`,`^=`,`&=`, `%=`, `<<=`,`>>=`, and `>>>=`.
This optimization can be particularly significant if the pattern occurs during a loop.

*Saves 28 gas for a storage array, 38 for a memory array*

## Alleviation:

Jeti team has already Resolved this issue

## GAS-04: Use Custom Errors

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Use Custom Errors | - | Check on finding | Gas Optimization | **Resolved** |

### Finding:

```
File: Blacklist.sol

34:         require(_blacklistAdminsContracts[msg.sender][_contract] == 1 ||
msg.sender == admin, "You need to be an admin to the contract you are blacklisting
for");

35:         require(_type <= 3, "Invalid blacklist code");

76:         require(blacklist.blacklist_type != BlacklistType.PERMANENT, "You are
permanently banned");

89:         require(_blacklistAdminsContracts[msg.sender][_contract] == 1 ||
msg.sender == admin, "You need to be an admin to the contract you are blacklisting
for");

```
File: JSRVGovernanceFactory.sol

28:     require(msg.value >= flatFee, "Flat fee");

63:     require(_newTokenFee <= 100, "Trying to set the fee to high");

77:     require(limits_[3] > block.timestamp, "must start after current time");

78:     require(limits_[4] > limits_[3], "start must be after cutoff");

79:     require(type_ == 1 || type_ == 0, "You must select they proper type");

```
File: Tokens/JSRVGovernance.sol

106:         require(threshold_[0] + threshold_[1] + threshold_[2] == totalSupply_,
"Invalid threshold amounts");

220:         require(_allowances[_msgSender()][spender]>0, "ERC20: decreased allowance
below zero");
```

```
244:         require(owner != address(0), "ERC20: approve from the zero address");

245:         require(spender != address(0), "ERC20: approve to the zero address");

258:         require((amount + balanceOf(recipient) <= _maxHold && recipient !=
address(this)) || recipient == address(this), "You are trying to hold too many
tokens");

276:         require(_allowances[sender][_msgSender()] > 0, "ERC20: transfer amount
exceeds allowance");

277:         require(amount + balanceOf(recipient) <= _maxHold, "You are trying to
hold too many tokens");

296:         require(_totalSold < _totalSupply, "All sold out");

297:         require((_totalSold + _amount) <= _totalSupply, "Trying to mint too many
tokens");

298:         require(_amount + balanceOf(msg.sender) <= _maxHold, "You are trying to
hold too many tokens");

299:         require(_amount >= _minPurchase && _amount <= _maxPurchase, "You need to
buy the right amounts");

308:         require(msg.value >= (roundFee[0] + roundFee[1] + roundFee[2]), "Not
enough value added");

331:         require(_totalSold >= _totalSupply, "Recycle not yet available");

332:         require(balanceOf(address(this)) > 0, "There are no tokens to recycle");

333:         require(recycleAllowed,"Recycling of tokens not allowed");

334:         require(msg.value >= _amount / _recycleRate, "You need to send enough");

335:         require(_amount + balanceOf(msg.sender) <= _maxHold, "You are trying to
hold too many tokens");

419:         require(balanceOf(msg.sender) > 0, "You have no tokens");

421:         require(currentRound > 0 && _holderLastRedeemed[msg.sender] <=
(currentRound), "Nothing to claim");

461:         require(balanceOf(msg.sender) > 0, "You have no tokens");
```

```
463:        require(_holderLastRedeemed[msg.sender] <= (currentRound), "Nothing to
claim");

509:        require(redeemTokens <= balanceOf(msg.sender), "You don't have enough
tokens");

510:        require(profitShare <= address(this).balance, "Not enought funds to
withdraw");

610:        require(address(this).balance > _totalAvailable, "You have no excesses in
here");
```

## Recommendation:

[Source](https://blog.soliditylang.org/2021/04/21/custom-errors/)
Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

## Alleviation:

Jeti team has already Resolved this issue

## GAS-05: Use != 0 instead of > 0 for unsigned integer comparison

| Vulnerability Detail | Severity | Location | Category | Status |
|---|---|---|---|---|
| Use != 0 instead of > 0 for unsigned integer comparison | - | Check on finding | Gas Optimization | **Resolved** |

### Finding:

```
File: JSRVGovernanceFactory.sol

49:     if (refund > 0) {

```
File: Tokens/JSRVGovernance.sol

220:         require(_allowances[_msgSender()][spender]>0, "ERC20: decreased allowance below zero");

276:         require(_allowances[sender][_msgSender()] > 0, "ERC20: transfer amount exceeds allowance");

332:         require(balanceOf(address(this)) > 0, "There are no tokens to recycle");

395:             if(addAmount > 0){

401:             if(remain > 0){

419:         require(balanceOf(msg.sender) > 0, "You have no tokens");

421:         require(currentRound > 0 && _holderLastRedeemed[msg.sender] <= (currentRound), "Nothing to claim");

432:             if(i>0 && holder._tokens > 0){

432:             if(i>0 && holder._tokens > 0){

433:                 if(_availableProfitShare[i] - _redeemedProfitShare[i] > 0){

434:                     if(_totalSoldTokens[i] > 0 && holder._tokens > 0) {

434:                     if(_totalSoldTokens[i] > 0 && holder._tokens > 0) {

442:                     if(holder._tokens > 0) {
```
```

```
461:         require(balanceOf(msg.sender) > 0, "You have no tokens");

477:             if(i>0 && holder._tokens > 0){

477:             if(i>0 && holder._tokens > 0){

478:                 if(_availableProfitShare[i] - _redeemedProfitShare[i] > 0){

479:                     if(_totalSoldTokens[i] > 0 && holder._tokens > 0) {

479:                     if(_totalSoldTokens[i] > 0 && holder._tokens > 0) {

487:                     if(holder._tokens > 0) {

541:         if(depositedAmount > 0) {

```
```
```

## Recommendation:

-

## Alleviation:

Jeti team has already Resolved this issue

## SWC Findings

| ID | Title | Scanning | Result |
|---|---|---|---|
| SWC-100 | Function Default Visibility | Complete | No risk |
| SWC-101 | Integer Overflow and Underflow | Complete | No risk |
| SWC-102 | Outdated Compiler Version | Complete | No risk |
| SWC-103 | Floating Pragma | Complete | No risk |
| SWC-104 | Unchecked Call Return Value | Complete | No risk |
| SWC-105 | Unprotected Ether Withdrawal | Complete | No risk |
| SWC-106 | Unprotected SELFDESTRUCT Instruction | Complete | No risk |
| SWC-107 | Reentrancy | Complete | No risk |
| SWC-108 | State Variable Default Visibility | Complete | No risk |
| SWC-109 | Uninitialized Storage Pointer | Complete | No risk |
| SWC-110 | Assert Violation | Complete | No risk |
| SWC-111 | Use of Deprecated Solidity Functions | Complete | No risk |
| SWC-112 | Delegatecall to Untrusted Callee | Complete | No risk |
| SWC-113 | DoS with Failed Call | Complete | No risk |
| SWC-114 | Transaction Order Dependence | Complete | No risk |
| SWC-115 | Authorization through tx.origin | Complete | No risk |

| | | | |
|---|---|---|---|
| SWC-116 | Block values as a proxy for time | Complete | Low |
| SWC-117 | Signature Malleability | Complete | No risk |
| SWC-118 | Incorrect Constructor Name | Complete | No risk |
| SWC-119 | Shadowing State Variables | Complete | No risk |
| SWC-120 | Weak Sources of Randomness from Chain Attributes | Complete | No risk |
| SWC-121 | Missing Protection against Signature Replay Attacks | Complete | No risk |
| SWC-122 | Lack of Proper Signature Verification | Complete | No risk |
| SWC-123 | Requirement Violation | Complete | No risk |
| SWC-124 | Write to Arbitrary Storage Location | Complete | No risk |
| SWC-125 | Incorrect Inheritance Order | Complete | No risk |
| SWC-126 | Insufficient Gas Griefing | Complete | No risk |
| SWC-127 | Arbitrary Jump with Function Type Variable | Complete | No risk |
| SWC-128 | DoS With Block Gas Limit | Complete | No risk |
| SWC-129 | Typographical Error | Complete | No risk |
| SWC-130 | Right-To-Left-Override control character (U+202E) | Complete | No risk |
| SWC-131 | Presence of unused variables | Complete | No risk |
| SWC-132 | Unexpected Ether balance | Complete | No risk |

| | | | |
|---|---|---|---|
| SWC-133 | Hash Collisions With Multiple Variable Length Arguments | Complete | No risk |
| SWC-134 | Message call with hardcoded gas amount | Complete | No risk |
| SWC-135 | Code With No Effects | Complete | No risk |
| SWC-136 | Unencrypted Private Data On-Chain | Complete | No risk |

# Visibility, Mutability, Modifier function testing

## Components

| 📝Contracts | 📚Libraries | 🔍Interfaces | 🎨Abstract |
|---|---|---|---|
| 3 | 0 | 2 | 1 |

## Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

| 🌐Public | 💰Payable |
|---|---|
| 54 | 6 |

| External | Internal | Private | Pure | View |
|---|---|---|---|---|
| 10 | 39 | 3 | 0 | 22 |

## StateVariables

| Total | 🌐Public |
|---|---|
| 48 | 34 |

## Capabilities

| Solidity Versions observed | 🧪 Experimental Features | 💰 Can Receive Funds | 🖥️ Uses Assembly | 💣 Has Destroyable Contracts |
|---|---|---|---|---|
| `0.8.17` `=0.8.17` | | `yes` | — | — |

| 📤 Transfers ETH | ⚡ Low-Level Calls | 👥 Delegate Call | 🧮 Uses Hash Functions | 🧹 ECRecover | 🌀 New/Create/Create2 |
|---|---|---|---|---|---|
| `yes` | — | — | — | — | `yes` → `NewContract:JSRVGovernanceToken` |

## FULL AUDIT REPORT

| TryCatch | Σ Unchecked |
| --- | --- |
| | |

Contracts Description Table

| Contract | Type | Bases | | |
|---|---|---|---|---|
| L | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **BaseToken** | Implementation | | | |
| | | | | |
| **iJSRVGovernanceFactory** | Interface | | | |
| L | _getBlacklist | External ! | | NO ! |
| L | _getAdmin | External ! | | NO ! |
| | | | | |
| **JSRVGovernanceToken** | Implementation | IERC20, Ownable, BaseToken, Pausable, ReentrancyGuard | | |
| L | | External ! | 💵 | NO ! |
| L | | Public ! | 💵 | NO ! |
| L | name | Public ! | | NO ! |
| L | symbol | Public ! | | NO ! |
| L | decimals | Public ! | | NO ! |
| L | totalSupply | Public ! | | NO ! |
| L | balanceOf | Public ! | | NO ! |
| L | addOwner | Public ! | 🛑 | onlyOwner |
| L | removeOwner | Public ! | 🛑 | onlyOwner |
| L | changeFeeTo | Public ! | 🛑 | onlyOwner |
| L | toggleRecycle | Public ! | 🛑 | onlyOwner |
| L | toggleAdminSigner | Public ! | 🛑 | isAdmin |

## FULL AUDIT REPORT

| Contract | Type | Bases | | |
|---|---|---|---|---|
| └ | pause | Public ! | | validOwner |
| └ | unpause | Public ! | | validOwner |
| └ | allowance | Public ! | | whenNotPaused |
| └ | increaseAllowance | Public ! | | whenNotPaused |
| └ | decreaseAllowance | Public ! | | whenNotPaused |
| └ | approve | Public ! | | whenNotPaused |
| └ | _approve | Private | | |
| └ | transfer | Public ! | | whenNotPaused |
| └ | transferFrom | Public ! | | whenNotPaused |
| └ | mint | Public ! | | whenNotPaused nonReentrant |
| └ | recycleMint | Public ! | | whenNotPaused nonReentrant |
| └ | getRates | Internal | | |
| └ | getCurrentRound | Public ! | | NO ! |
| └ | getCurrentPurchaseRound | Public ! | | NO ! |
| └ | updateData | Internal | | |
| └ | calcProfitShareClaim | Public ! | | NO ! |
| └ | redeemProfitShare | Public ! | | NO ! |

## FULL AUDIT REPORT

| Contract | Type | Bases | | |
|---|---|---|---|---|
| └ | depositProfitShare | Public ! | | onlyOwner |
| └ | refundExcessiveFee | Internal | | |
| └ | setNewMaxHold | Public ! | | validOwner |
| └ | readHolderData | Public ! | | NO ! |
| └ | availableFunds | Public ! | | validOwner |
| └ | withdrawFunds | Public ! | | onlyOwner |
| └ | createTransaction | Private | | notOpen |
| └ | signTransaction | Public ! | | validOwner<br>txExists<br>notExecuted<br>notConfirmed |
| └ | _withdrawFunds | Private | | |
| | | | | |
| **Blacklist** | Implementation | Ownable | | |
| └ | setBlacklist | Public ! | | NO ! |
| └ | blacklistTimeLeft | Public ! | | NO ! |
| └ | resetBlacklist | Public ! | | NO ! |
| └ | deleteBlacklist | Public ! | | NO ! |
| └ | getBlacklist | External ! | | NO ! |
| └ | getBlacklistType | External ! | | NO ! |
| └ | getBlacklistDetails | External ! | | NO ! |
| └ | addBlacklistAdmins | Public ! | | onlyOwner |
| └ | removeBlacklistAdmins | Public ! | | onlyOwner |
| └ | setAdmin | Public ! | | onlyOwner |

## FULL AUDIT REPORT

| Contract | Type | Bases | | |
|---|---|---|---|---|
| └ | getAdminStatus | Public ❗ | | NO ❗ |
| | | | | |
| **iBlacklist** | Interface | | | |
| └ | getBlacklist | External ❗ | | NO ❗ |
| | | | | |
| **JSRVGovernanceFactory** | Implementation | ReentrancyGuard, Ownable | | |
| └ | | Public ❗ | 🛑 | NO ❗ |
| └ | setFeeTo | External ❗ | 🛑 | onlyOwner |
| └ | setFlatFee | External ❗ | 🛑 | onlyOwner |
| └ | refundExcessiveFee | Internal 🔒 | 🛑 | |
| └ | newBlacklistContract | Public ❗ | 🛑 | onlyOwner |
| └ | newTokenFeeTo | Public ❗ | 🛑 | onlyOwner |
| └ | newAdmin | Public ❗ | 🛑 | onlyOwner |
| └ | newTokenFee | Public ❗ | 🛑 | onlyOwner |
| └ | create | External ❗ | 💵 | enoughFee nonReentrant |
| └ | _getBlacklist | Public ❗ | | NO ❗ |
| └ | _getAdmin | Public ❗ | | NO ❗ |

Legend

| Symbol | Meaning |
|---|---|
| 🛑 | Function can modify state |
| 💵 | Function is payable |

# Inheritate Function Relation Graph

# About SECURI LAB

SECURI LAB is a group of cyber security experts providing cyber security consulting, smart contract security audits, and KYC services.


## Follow Us On:

| | |
|---|---|
| Website | https://securi-lab.com/ |
| Twitter | https://twitter.com/SECURI_LAB |
| Telegram | https://t.me/securi_lab |
| Medium | https://medium.com/@securi |