CYPHER
SHIELD

CypherShield

Audit Report

Prepared for

JILAI

Audit report

Token Smart Contract

Table of contents

1. Scope of the Audit

The scope of this audit was to analyse and document JILAI Token Smart Contract codebase for quality, security, and correctness.

On March 26, 2025 to March 28, 2025, upon the request of JILAI Token Contract, CypherShield Team started a security audit related to JILAI Token Smart Contract.

On April 1, 2025, the said team sent the first report relating to said audit to JILAI Token Smart Contract

JILAI Web3 Development Team fixed the bugs found during the audit process as indicated in the audit report.

Then, from April 8, 2025 to April 10, 2025, the CypherShield Team conducted a security re-audit for JILAI Token Smart Contract.

Smart Contract Code Repository- Audit fixes have been updated in the same repository : https://github.com/Blockchainxtech/Jilai-Contracts/tree/stages/contracts/JilaiToken

Version	Date	Release notes
1.0	28/03/2025	Initial report sent to the client. All findings are in open status.
2.0	08/04/2025	Re-audit was completed for the fixed findings, and the status has been updated.

The scope of this audit was to conduct a comprehensive analysis and documentation of the JILAI Token smart contract codebase with a focus on the following key areas:

1. Code Quality

- Evaluating the overall structure, readability, and maintainability of the code.
- Ensuring adherence to Solidity best practices and industry standards.
- Identifying areas where the code could be optimized for efficiency and clarity.

2. Security

- Identifying potential vulnerabilities, including reentrancy attacks, integer overflows/underflows, access control issues, and unprotected upgradeability.
- Analyzing external calls and dependencies for possible exploits.
- Verifying the proper implementation of access controls, authorization mechanisms, and role management.

3. Correctness and Functionality

- Verifying that the smart contract logic behaves as intended.
- Ensuring that token distribution, vesting schedules, and sale processes are accurately implemented.
- Validating that all mathematical operations and conditions are properly handled to prevent unexpected behavior.

4. Compliance and Consistency

- Ensuring compliance with Solidity naming conventions and code formatting standards.
- Checking for the correct usage of events, modifiers, and error handling to ensure consistency and traceability.

5. Documentation and Transparency

- Providing detailed explanations of the identified issues, their potential impact, and recommended fixes.
- Offering suggestions for code optimization, readability improvements, and gas efficiency enhancements.

2. Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities.Here are some of the commonly known vulnerabilities that we considered:

Here are 25 of the most well-known vulnerabilities to be aware of when auditing Solidity smart contracts:

A. Reentrancy: Vulnerable contracts can be re-entered before the current call completes, allowing attackers to manipulate the state or steal funds.

B. Integer Overflow/Underflow: Arithmetic operations can lead to unexpected results or vulnerabilities if not properly checked for overflow or underflow.

C. Unchecked External Calls: External calls should be carefully validated to prevent unauthorized access or unexpected behavior.

D. Access Control Issues: Lack of proper access controls can allow unauthorized users to manipulate the token distribution sale contract or access sensitive functions.

E. Gas Limit DoS: Inefficient code or loops can consume excessive gas, leading to denial-of-service (DoS) attacks by exhausting gas limits.

F. Unchecked Return Values: Failure to check return values from external calls can result in vulnerabilities such as ignoring errors or unexpected outcomes.

G. Timestamp Dependence: Relying on block timestamps for critical decisions can be manipulated by miners, compromising the contract's integrity.

H. Randomness Vulnerabilities: Insecure random number generation can be exploited to predict outcomes or manipulate the token distribution sale process.

I. Front-Running: Attackers can exploit transaction ordering to gain advantages, such as purchasing tokens at favorable prices before others.

J. Denial-of-Service (DoS): Malicious actors can target the token distribution contract with resource-intensive operations, causing disruption or delays.

K. Uninitialized Storage Pointers: Improper initialization of storage pointers can lead to unexpected data corruption or vulnerabilities.

L. Delegate call/Callcode Vulnerabilities: Misuse of delegatecall or callcode can result in unintended behavior or security vulnerabilities.

M. Insufficient Input Validation: Failing to validate input data can lead to vulnerabilities such as buffer overflows or unexpected contract states.

N. Improper Type Casting: Incorrectly casting data types can lead to vulnerabilities such as data truncation or unexpected behavior.

O. Public State Variables: Exposing sensitive state variables as public can lead to unauthorized access or manipulation.

P. Use of Deprecated Functions/Libraries: Deprecated functions or libraries may have known vulnerabilities that attackers can exploit.

Q. Unprotected Self-Destruct: Lack of protection on self-destruct can lead to funds being irreversibly sent to unintended addresses.

R. Gas Token Vulnerability: Vulnerabilities related to gas tokens can be exploited to manipulate gas costs or abuse gas refunds.

S. Unprotected Ether Withdrawal: Contracts that allow unrestricted withdrawal of Ether can be vulnerable to attacks by unauthorized users.

T. Missing Events for State Changes: Failure to emit events for state changes can make it difficult to track contract behavior and detect anomalies during the token distribution.

U. Forced Ether Transfer: Contracts may be vulnerable if they allow external parties to force Ether transfers without proper validation or authorization.

V. Insecure Dependency Management: Using unverified or untrusted dependencies can introduce vulnerabilities to the token distribution contract.

W. Unprotected Upgradeability: If the token distribution contract is upgradeable, ensure that upgrade mechanisms are secure and not susceptible to unauthorized upgrades or manipulations.

X. Insecure Whitelisting: If the token distribution implements whitelisting for participants, ensure that the whitelisting mechanism is secure and cannot be bypassed or manipulated by attackers.

3. Techniques and Methods:

Throughout the audit of the 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.
- 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.

3.1 Structural Analysis:

In this step, we conducted a structural analysis of the smart contract's design patterns and overall structure. We carefully examined the organization to ensure that the smart contract is structured in a manner that minimizes the likelihood of encountering issues in the future.

3.2 Code Review / Manual Analysis:

We conducted a thorough manual review of the code to uncover new vulnerabilities and confirm any vulnerabilities identified during the static analysis. This involved a detailed examination of the contracts, checking their logic against the descriptions in the whitepaper. Additionally, we manually checked and validated the results from the automated analysis tools.

3.3 Static Analysis:

We performed a static analysis of the smart contracts to pinpoint any potential vulnerabilities. This involved using a set of automated tools specifically designed to test the security aspects of the smart contracts.

3.4 Gas Consumption:

In this phase, we examined how smart contracts perform in real-world usage. We focused on understanding the amount of computational resources (gas) consumed and explored opportunities to optimize the code for lower gas consumption.

3.5 Tools and Platforms Used for Audit:

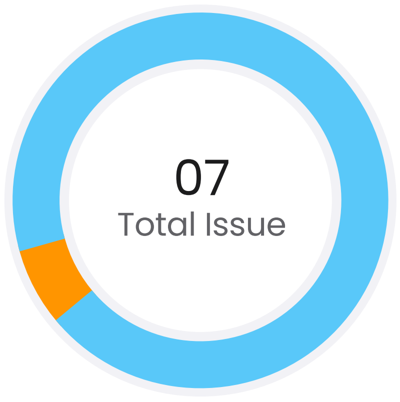
We utilized a range of tools such as Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity Statistic Analysis, Theo, and Visual Studio Code. These tools helped us in various aspects of smart contract development, testing, and analysis, ensuring a comprehensive approach to building secure and efficient contracts.

4. Issue Categories:

In the report, each issue has been categorized based on its severity level. There are four levels of severity, and I've outlined each one below for better understanding.

Risk-level Description	Description
High	A high-severity issue or vulnerability indicates that your smart contract is susceptible to exploitation. These issues are critical as they can significantly impact the performance or functionality of the smart contract. It's highly recommended to address these issues before deploying the contract in a live environment.
Medium	Medium severity issues typically stem from errors and deficiencies within the smart contract code. While not as critical as high-severity issues, these issues can still lead to problems and should be addressed to ensure the contract's robustness and reliability.
Low	Low-level severity issues generally have a minor impact or are considered warnings that may not require immediate fixing. However, it's advisable to address these issues at some point in the future to maintain the overall quality and stability of the smart contract.
Informational	These severity issues are typically related to improvement requests, general questions, cosmetic or documentation errors, or requests for information. They have little to no impact on the smart contract's functionality or performance.

5. Number of issues per severity:



🟥	High	0
🟧	Medium	1
🟦	Low	6
🟩	Informational	0

Title	Category	Severity	Status
Ignored Return Value in _upgradeToAndCall	Code Quality	Medium	This issue is from the OpenZeppelin library, so it does not affect our contract and can be safely ignored.
Inline Assembly in _revert	Code Quality	Low	This issue is from the OpenZeppelin library, so it does not affect our contract and can be safely ignored.
Multiple Solidity Versions Used	Code Quality	Low	Fixed
Dead Code in Context Upgradeable, ERC1967 Upgrade Upgradeable, Initializable, and UUPS Upgradeable	Code Quality	Low	This issue is from the OpenZeppelin library, so it does not affect our contract and can be safely ignored.
Incorrect Versions of Solidity Used	Security	Low	Fixed
Low-Level Calls in Address Upgradeable	Security	Low	This issue is from the OpenZeppelin library, so it does not affect our contract and can be safely ignored.
Non-MixedCase Naming Convention	Code Style	Low	This issue is from the OpenZeppelin library, so it does not affect our contract and can be safely ignored.

6. Issues Found - Code Review / Manual Testing

6.1 Contract - JILAI Token

A. High-severity issues

There is no High severity issues found

B. Medium-severity issues

B.1 Ignored Return Value in _upgradeToAndCall

Description: The return value from AddressUpgradeable.functionDelegateCall(newImplementation, data) is ignored in the _upgradeToAndCall function, which may cause issues in tracking the success or failure of the function call.

Recommendation: To resolve the issue of ignoring the return value in both _upgradeToAndCall and _upgradeBeaconToAndCall, you should handle or log the return value. This can help you ensure that you properly track the success or failure of the delegated function call.

Recommended Code Fix :

```
function _upgradeToAndCall(
   address newImplementation,
   bytes memory data,
   bool forceCall
) internal {
   // Call the function delegate and handle the return value
   (bool success, ) =
AddressUpgradeable.functionDelegateCall(newImplementation, data);
   require(success, "Upgrade failed");
}
```

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

C. Low-severity issues

C.1 Inline Assembly in _revert

Description: The use of inline assembly in AddressUpgradeable._revert(bytes, string) could lead to unexpected behavior, hard-to-debug errors, or issues with future Solidity versions.

Recommendation: Avoid Inline Assembly Usage:
Inline assembly can be useful for gas optimization, but it should be used cautiously. Rewriting functions in pure Solidity, whenever possible, can reduce security risks and improve code maintainability.

For instance, instead of using inline assembly for storage slot reading (which can be done using assembly), you can use Solidity's native function calls, such as storage operations, to access storage slots directly.

Recommended Code Fix :

```
function getAddressSlot(bytes32 slot) internal view returns (address r) {
   // Instead of using assembly, directly access storage
   assembly {
       r := sload(slot)
   }
}
```

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

C.2 Multiple Solidity Versions Used

Description: The project uses multiple Solidity versions, which can cause compatibility issues, unexpected behavior, and difficulties in maintaining the contract.

Recommendation: Unify Solidity Version:
It is recommended to use the same version of Solidity across your project to avoid compatibility issues. The simplest solution is to align the version constraints in all the Solidity files. Choose the latest version that supports the features you need and is compatible with all libraries used.

Adjust Pragma Directives:
Modify the pragma directive in all files to specify a single version or a compatible range.

Recommended Code Fix :

```
// contracts/Jilai.sol
pragma solidity ^0.8.27;
```

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

C.3 Dead Code in Context Upgradeable, ERC1967 Upgrade Upgradeable, Initializable, and UUPS Upgradeable

Description: The project contains multiple functions and methods that are never used. This dead code should be removed to reduce complexity, improve maintainability, and decrease the size of the deployed contract.

Recommendation: Remove Unused Functions and Methods:
The functions and methods identified in the audit report are never used within your contract or its interactions. They should be removed to avoid unnecessary bloat and improve clarity.

Recommended Code Fix :

```
function __Context_init() internal initializer { ... }
function __Context_init_unchained() internal initializer { ... }
function _msgData() internal view returns (bytes calldata) { ... }
```

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

C.4 Low-Level Calls in Address Upgradeable

Description: Low-level calls such as call, static call, and delegate call by pass Solidity's type safety checks and error handling, making them risky. They:
Do not revert on failure unless explicitly handled.
Are prone to reentrancy attacks.
Can introduce unexpected behavior due to gas forwarding issues.

Recommendation: Use OpenZeppelin's Safe Functions:
Replace low-level calls with OpenZeppelin's safe functions, such as Address.functionCall or Address.functionDelegateCall, which handle reverts properly.

Recommended Code Fix :

```
// Instead of this:
(success, returndata) = target.call{value: value}(data);

// Use this:
returndata = Address.functionCallWithValue(target, data, value, "Low-level
call failed");
```

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

C.5 Non-MixedCase Naming Convention

Description: Several functions and variables in the contract do not follow Solidity's mixedCase naming convention, making the codebase inconsistent and potentially less readable. According to Solidity best practices:
Functions and variables should follow mixedCase.
Constants should use UPPER_CASE_WITH_UNDERSCORES.
Events and structs should use CamelCase.

Recommendation: Refactor Function Names to MixedCase
Functions prefixed with __ should be refactored to follow mixedCase naming conventions.

Recommended Code Fix :

```
// Before
function __Ownable_init() public initializer { ... }
function __Ownable_init_unchained() public initializer { ... }

// After (MixedCase)
function initializeOwnable() public initializer { ... }
function initializeOwnableUnchained() public initializer { ... }
```

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

7. Test Cases for Functional Testing

Some of the test cases which were part of the functional testing are listed below:

Description	Status
Ensure that owner can not approve with '0' amount to a spender	Pass
Verify that while trying to approve more than the available balance is failed	Pass
Ensure that approving the tokens for invalid address is not possible	Pass
Ensure that Burning more than balance Should revert (insufficient balance).	Pass
Ensure that Burning from zero address Should revert (invalid sender).	Pass
verify that Reducing the approved spending limit of a spender works correctly by entering the correct amount	Pass
Verify that Decrease below '0' Should revert (allowance cannot be negative).	Pass
Ensure that Increasing a spender's token spending limit works correctly by entering the correct amount	Pass
Ensure that Increase by '0' tokens Should execute but have no effect.	Pass
Verify that Direct token transfer between accounts works correctly	Pass
Ensure that Transfer more than balance Should revert (insufficient funds).	Pass
Verify that Self-transfer Should execute but have no effect.	Pass

8. Automated Tests

8.1 Slither

```
INFO:Detectors:
ERC1967UpgradeUpgradeable._upgradeToAndCall(address,bytes,bool)
(node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#65-70) ignores return value by
AddressUpgradeable.functionDelegateCall(newImplementation,data)
(node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#68)
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool)
(node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#156-162) ignores return value by
AddressUpgradeable.functionDelegateCall(IBeaconUpgradeable(newBeacon).im
plementation(),data) (node_modules/@openzeppelin/contracts-upgradeable/
proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#unused-return
INFO:Detectors:
AddressUpgradeable._revert(bytes,string) (node_modules/@openzeppelin/
contracts-upgradeable/utils/AddressUpgradeable.sol#231-243) uses assembly
   - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/
AddressUpgradeable.sol#236-239)
StorageSlotUpgradeable.getAddressSlot(bytes32) (node_modules/
@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#62-67) uses assembly
   - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#64-66)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (node_modules/
@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#72-77) uses assembly
   - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#74-76)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/
@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#82-87) uses assembly
   - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#84-86)
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/
@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#92-97) uses assembly
   - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#94-96)
StorageSlotUpgradeable.getStringSlot(bytes32) (node_modules/
@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#102-107) uses assembly
   - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#104-106)
StorageSlotUpgradeable.getStringSlot(string) (node_modules/@openzeppelin/
contracts-upgradeable/utils/StorageSlotUpgradeable.sol#112-117) uses
assembly
   - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#114-116)
StorageSlotUpgradeable.getBytesSlot(bytes32) (node_modules/
@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#122-127) uses assembly
   - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#124-126)
StorageSlotUpgradeable.getBytesSlot(bytes) (node_modules/@openzeppelin/
contracts-upgradeable/utils/StorageSlotUpgradeable.sol#132-137) uses
assembly
   - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#134-136)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#assembly-usage
```

```
INFO:Detectors:
4 different versions of Solidity are used:
   - Version constraint ^0.8.27 is used by:
      -^0.8.27 (contracts/Jilai.sol#2)
   - Version constraint ^0.8.0 is used by:
      -^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/
OwnableUpgradeable.sol#4)
      -^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
interfaces/IERC1967Upgradeable.sol#4)
      -^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
interfaces/draft-IERC1822Upgradeable.sol#4)
      -^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
beacon/IBeaconUpgradeable.sol#4)
      -^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
utils/UUPSUpgradeable.sol#4)
      -^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
ERC20/ERC20Upgradeable.sol#4)
-^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/
IERC20Upgradeable.sol#4)
      -^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
ERC20/extensions/IERC20MetadataUpgradeable.sol#4)
      -^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
ContextUpgradeable.sol#4)
      -^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#5)
   - Version constraint ^0.8.2 is used by:
      -^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
ERC1967/ERC1967UpgradeUpgradeable.sol#4)
      -^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
utils/Initializable.sol#4)
   - Version constraint ^0.8.1 is used by:
      -^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/
AddressUpgradeable.sol#4)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#different-pragma-directives-are-used
INFO:Detectors:
ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/
contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is never used
and should be removed
ContextUpgradeable.__Context_init_unchained() (node_modules/
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22)
is never used and should be removed
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-
upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and should
be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (node_modules/
@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#20-21) is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained()
(node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#23-24) is never used and should be removed
ERC1967UpgradeUpgradeable._changeAdmin(address) (node_modules/
@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#120-123) is never used and should be
removed
ERC1967UpgradeUpgradeable._getAdmin() (node_modules/@openzeppelin/
contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#103-105) is never used and should be
removed
ERC1967UpgradeUpgradeable._getBeacon() (node_modules/@openzeppelin/
contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#134-136) is never used and should be
removed
ERC1967UpgradeUpgradeable._setAdmin(address) (node_modules/
@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#110-113) is never used and should be
removed
```

```
ERC1967UpgradeUpgradeable._setBeacon(address) (node_modules/
@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#141-148) is never used and should be
removed
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool)
(node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#156-162) is never used and should be
removed
Initializable._getInitializedVersion() (node_modules/@openzeppelin/
contracts-upgradeable/proxy/utils/Initializable.sol#156-158) is never used
and should be removed
Initializable._isInitializing() (node_modules/@openzeppelin/contracts-
upgradeable/proxy/utils/Initializable.sol#163-165) is never used and should
be removed
UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/
@openzeppelin/contracts-upgradeable/proxy/utils/
UUPSUpgradeable.sol#26-27) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#dead-code
INFO:Detectors:
Version constraint ^0.8.0 contains known severe issues (https://
solidity.readthedocs.io/en/latest/bugs.html)
    - FullInlinerNonExpressionSplitArgumentEvaluationOrder
    - MissingSideEffectsOnSelectorAccess
    - AbiReencodingHeadOverflowWithStaticArrayCleanup
    - DirtyBytesArrayToStorage
    - DataLocationChangeInInternalOverride
    - NestedCalldataArrayAbiReencodingSizeValidation
    - SignedImmutables
    - ABIDecodeTwoDimensionalArrayMemory
    - KeccakCaching.
```

```
It is used by:
   - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/
OwnableUpgradeable.sol#4)
   - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/
IERC1967Upgradeable.sol#4)
   - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/
draft-IERC1822Upgradeable.sol#4)
   - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
beacon/IBeaconUpgradeable.sol#4)
   - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/
UUPSUpgradeable.sol#4)
   - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
ERC20/ERC20Upgradeable.sol#4)
   - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
ERC20/IERC20Upgradeable.sol#4)
   - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
ERC20/extensions/IERC20MetadataUpgradeable.sol#4)
   - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
ContextUpgradeable.sol#4)
   - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
StorageSlotUpgradeable.sol#5)
Version constraint ^0.8.2 contains known severe issues (https://
solidity.readthedocs.io/en/latest/bugs.html)
   - FullInlinerNonExpressionSplitArgumentEvaluationOrder
   - MissingSideEffectsOnSelectorAccess
   - AbiReencodingHeadOverflowWithStaticArrayCleanup
   - DirtyBytesArrayToStorage
   - DataLocationChangeInInternalOverride
   - NestedCalldataArrayAbiReencodingSizeValidation
   - SignedImmutables
   - ABIDecodeTwoDimensionalArrayMemory
   - KeccakCaching.
```

```
It is used by:
   - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
ERC1967/ERC1967UpgradeUpgradeable.sol#4)
   - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/
Initializable.sol#4)
Version constraint ^0.8.1 contains known severe issues (https://
solidity.readthedocs.io/en/latest/bugs.html)
   - FullInlinerNonExpressionSplitArgumentEvaluationOrder
   - MissingSideEffectsOnSelectorAccess
   - AbiReencodingHeadOverflowWithStaticArrayCleanup
   - DirtyBytesArrayToStorage
   - DataLocationChangeInInternalOverride
   - NestedCalldataArrayAbiReencodingSizeValidation
   - SignedImmutables
   - ABIDecodeTwoDimensionalArrayMemory
   - KeccakCaching.
It is used by:
   - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/
AddressUpgradeable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in AddressUpgradeable.sendValue(address,uint256)
(node_modules/@openzeppelin/contracts-upgradeable/utils/
AddressUpgradeable.sol#64-69):
   - (success,None) = recipient.call{value: amount}() (node_modules/
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#67)
Low level call in
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string)
(node_modules/@openzeppelin/contracts-upgradeable/utils/
AddressUpgradeable.sol#128-137):
   - (success,returndata) = target.call{value: value}(data) (node_modules/
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
```

```
Low level call in
AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/
@openzeppelin/contracts-upgradeable/utils/
AddressUpgradeable.sol#155-162):
   - (success,returndata) = target.staticcall(data) (node_modules/
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#160)
Low level call in
AddressUpgradeable.functionDelegateCall(address,bytes,string)
(node_modules/@openzeppelin/contracts-upgradeable/utils/
AddressUpgradeable.sol#180-187):
   - (success,returndata) = target.delegatecall(data) (node_modules/
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#low-level-calls
INFO:Detectors:
Function OwnableUpgradeable.__Ownable_init() (node_modules/
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#29-31)
is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (node_modules/
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#33-35)
is not in mixedCase
Variable OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-
upgradeable/access/OwnableUpgradeable.sol#94) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init()
(node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#20-21) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained()
(node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/
ERC1967UpgradeUpgradeable.sol#23-24) is not in mixedCase
Variable ERC1967UpgradeUpgradeable.__gap (node_modules/@openzeppelin/
contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#169)
is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init() (node_modules/
@openzeppelin/contracts-upgradeable/proxy/utils/
UUPSUpgradeable.sol#23-24) is not in mixedCase
```

```
Function UUPSUpgradeable.__UUPSUpgradeable_init_unchained()
(node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/
UUPSUpgradeable.sol#26-27) is not in mixedCase
Variable UUPSUpgradeable.__self (node_modules/@openzeppelin/contracts-
upgradeable/proxy/utils/UUPSUpgradeable.sol#29) is not in mixedCase
Variable UUPSUpgradeable.__gap (node_modules/@openzeppelin/contracts-
upgradeable/proxy/utils/UUPSUpgradeable.sol#111) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (node_modules/
@openzeppelin/contracts-upgradeable/token/ERC20/
ERC20Upgradeable.sol#55-57) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string)
(node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/
ERC20Upgradeable.sol#59-62) is not in mixedCase
Variable ERC20Upgradeable.__gap (node_modules/@openzeppelin/contracts-
upgradeable/token/ERC20/ERC20Upgradeable.sol#376) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/
contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in
mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22)
is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-
upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:contracts/Jilai.sol analyzed (14 contracts with 93 detectors),
49 result(s) found
```

8.2 Results

The JILAI Token smart contracts demonstrate a solid and secure architectural foundation. All previously identified critical vulnerabilities have been successfully addressed. With the implemented fixes, the contracts are now significantly more secure, gas-efficient, and maintainable.

9. Closing Summary

The smart contracts were well-written and adhered to the guidelines. However, several issues were identified during the audit. The development team resolved all the issues, and the contract re-audit was successfully completed.

CYPHER
SHIELD

As vulnerabilities exist in the web3 spaces, **Cypershield** is one of the kinds of Security and Smart Contract audit company rendering exceptionally professional smart contract auditing services for varied Crypto projects. In the process of rendering your projects, full-on auditing services help you come over your smart contract vulnerabilities and reach a higher scale in the market.

For more information : contact@cyphershield.tech