// Security Assessment

08.14.2024 - 08.23.2024

Shinkai Protocol

Shinkai

HALBORN

Shinkai Protocol - Shinkai

Prepared by: HALBORN

Last Updated 03/19/2025

Date of Engagement: August 14th, 2024 - August 23rd, 2024

Summary

0% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
8	0	0	0	1	7

TABLE OF CONTENTS

1. Introduction

The `Shinkai team` engaged `Halborn` to conduct a security assessment on their smart contracts beginning on 2024-08-14 and ending on 2024-08-23. The security assessment was scoped to the smart contracts provided in the GitHub repositories:

- https://github.com/dcSpark/shinkai-contracts

Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

`Halborn` was provided one week and two days for the engagement and assigned one full-time security engineer to check the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, `Halborn` identified several security concerns that should be addressed by the `Shinkai team.`

3. Test Approach And Methodology

`Halborn` performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could led to arithmetic related vulnerabilities.
- Manual testing by custom scripts.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).
- Static Analysis of security for scoped contract, and imported functions. (`Slither`).
- Local or public testnet deployment (`Foundry` , `Remix IDE`).

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: shinkai-contracts

(b) Assessed Commit ID: 52a83ce

(c) Items in scope:

- src/RegistryControlled.sol
- src/ShinkaiNft.sol
- src/ShinkaiToken.sol
- src/ShinkaiRegistry.sol
- src/ShinkaiNftInterface.sol
- src/ShinkaiRegistryInterface.sol
- src/ShinkaiTokenInterface.sol
- src/StringUtils.sol
- src/UintLib.sol

Out-of-Scope: Third party dependencies and economic attacks.

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	7

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - HARDCODED LIMITER BASED ON CURRENT NUMBER OF BLOCKS PER YEAR	LOW	NOT SOLVED
HAL-02 - MISSING VISIBILITY MODIFIER FOR THE AVAILABLE NAMESPACES	INFORMATIONAL	NOT SOLVED
HAL-03 - LACK OF STORAGE GAP IN UPGRADEABLE CONTRACT	INFORMATIONAL	NOT SOLVED
HAL-04 - USE OF OWNABLEUPGRADEABLE LIBRARY WITH SINGLE-STEP OWNERSHIP TRANSFER	INFORMATIONAL	NOT SOLVED
HAL-05 - CONSIDER USING NAMED MAPPINGS	INFORMATIONAL	NOT SOLVED

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-06 - USE OF UNLICENSED SMART CONTRACTS	INFORMATIONAL	NOT SOLVED
HAL-07 - REDUNDANT REWARD ACCRUAL	INFORMATIONAL	NOT SOLVED
HAL-08 - UNLOCKED PRAGMA COMPILERS	INFORMATIONAL	NOT SOLVED

7. FINDINGS & TECH DETAILS

7.1 (HAL-01) HARDCODED LIMITER BASED ON CURRENT NUMBER OF BLOCKS PER YEAR

// LOW

Description

The `baseRewardsRateMaxMantissa` is set in the ShinkaiRegistry contract as a constant, calculated based on a fixed assumption about the number of blocks per year. Due to its immutability, this value is hardcoded into the contract's bytecode and cannot be altered without implementation upgrade. However, as the block time can change over time due to network adjustments or protocol upgrades, relying on this hardcoded constant can lead to inaccuracies.

```
15 |     uint256 public constant baseRewardsRateMaxMantissa = 190258751902; // 50% inflation yearly (0.5e18 / blocksInYear)
```

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation

Consider implementing a function allowing to dynamically adjust the `baseRewardsRateMaxMantissa` variable.

References

["https://github.com/dcSpark/shinkai-contracts/blob/52a83ce548eac47022b61f7aed23bfaa3e640c7d/src/ShinkaiRegistry.sol#L15"]

7.2 (HAL-02) MISSING VISIBILITY MODIFIER FOR THE AVAILABLE NAMESPACES

// INFORMATIONAL

Description

The `namespace` mapping in the ShinkaiRegistry contract holds a string that is concatenated to the user's identity when they claim a new identity. However, the mapping lacks a visibility modifier, which defaults its visibility to internal. This oversight makes it difficult for users to check the available namespaces before selecting where to concatenate their identity. As a result, users might unintentionally select the wrong namespace, leading to potential identity mismanagement.

```
33 |     mapping(uint256 => string) namespaces;
```

BVSS

AO:A/AC:L/AX:M/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (1.7)

Recommendation

Consider adding a public visibility to the `namespace` mapping.

Remediation Comment

Consider adding a public visibility to the `namespace` mapping.

References

["https://github.com/dcSpark/shinkai-contracts/blob/52a83ce548eac47022b61f7aed23bfaa3e640c7d/src/ShinkaiRegistry.sol#L33"]

7.3 (HAL-03) LACK OF STORAGE GAP IN UPGRADEABLE CONTRACT

// INFORMATIONAL

Description

The `ShinkaiRegistry` contracts is designed to be used with a UUPS proxy pattern. However, it lacks storage gaps. Storage gaps are essential for ensuring that new state variables can be added in future upgrades without affecting the storage layout of inheriting child contracts. Without it, any addition of new state variables in future contract versions can lead to storage collisions.

BVSS

AO:A/AC:L/AX:H/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (1.6)

Recommendation

Consider adding a storage gap as the last storage variable. Place the `uint256[50] private __gap;` variable at the end of storage layout of ShinkaiRegistry contract.

7.4 (HAL-04) USE OF OWNABLEUPGRADEABLE LIBRARY WITH SINGLE-STEP OWNERSHIP TRANSFER

// INFORMATIONAL

Description

The ownership of the contracts can be lost as the ShinkaiRegistry and ShinkaiControlled contracts inherited from the OwnableUpgradeable/Ownable contract and their ownership can be transferred in a single-step process. The address the ownership is changed to should be verified to be active or willing to act as the owner.

BVSS

AO:S/AC:L/AX:L/C:N/I:M/A:L/D:N/Y:N/R:N/S:U (1.1)

Recommendation

It is recommended to implement a two-step process where the owner nominates an account and the nominated account needs to call an `acceptOwnership` function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. This can be achieved by using OpenZeppelin's Ownable2StepUpgradeable contract instead of the OwnableUpgradeable.

Remediation Comment

It is recommended to implement a two-step process where the owner nominates an account and the nominated account needs to call an `acceptOwnership` function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. This can be achieved by using OpenZeppelin's Ownable2StepUpgradeable contract instead of the OwnableUpgradeable.

7.5 (HAL-05) CONSIDER USING NAMED MAPPINGS

// INFORMATIONAL

Description

The project is using Solidity version greater than 0.8.18, which supports named mappings. Using named mappings can improve the readability and maintainability of the code by making the purpose of each mapping clearer. This practice will enhance code readability and make the purpose of each mapping more explicit.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation

Consider refactoring the mappings to use named arguments.
For example, on instead of declaring:

```
mapping(uint256 => string) public tokenIdToIdentity;
```

The mapping could be declared as:

```
mapping(uint256 tokenId => string identity) public tokenIdToIdentity;
```

Remediation Comment

Consider refactoring the mappings to use named arguments.
For example, on instead of declaring:

```
mapping(uint256 => string) public tokenIdToIdentity;
```

The mapping could be declared as:

```
mapping(uint256 tokenId => string identity) public tokenIdToIdentity;
```

7.6 (HAL-06) USE OF UNLICENSED SMART CONTRACTS

// INFORMATIONAL

Description

All the Shinkai smart contracts are marked as unlicensed, as indicated by the SPDX license identifier at the top of the files:

```
// SPDX-License-Identifier: UNLICENSED
```

Using unlicensed contract can lead to legal uncertainties and potential conflicts regarding the usage, modification and distribution rights of the code. This may deter other developers from using or contributing to the project and could potentially lead to legal issues in the future.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation

It is recommended to choose and apply an appropriate open-source license to the smart contracts. Some popular options for blockchain and smart contract projects include:
1. MIT License: A permissive license that allows for reuse with minimal restrictions.
2. GNU General Public License (GPL): A copyleft license that ensures derivative works are also open-source.
3. Apache License 2.0: A permissive license that provides an express grant of patent rights from contributors to users.

Remediation Comment

It is recommended to choose and apply an appropriate open-source license to the smart contracts. Some popular options for blockchain and smart contract projects include:
1. MIT License: A permissive license that allows for reuse with minimal restrictions.
2. GNU General Public License (GPL): A copyleft license that ensures derivative works are also open-source.
3. Apache License 2.0: A permissive license that provides an express grant of patent rights from contributors to users.

7.7 (HAL-07) REDUNDANT REWARD ACCRUAL

// INFORMATIONAL

Description

The `unclaimIdentity` function implemented in ShinkaiRegistry calls claimStakingRewards method before proceeding with the unclaim process to ensure the user's staking rewards are accrued. After accruing, it burn the related NFT.

```
    function unclaimIdentity(string calldata identity) public onlyOwnerIdentity(identity) {
        claimStakingRewards(identity);

        shinkaiNft.burn(identityData[identity].boundNft);
        emit IdentityUnclaim(identity, identityData[identity].boundNft);

        delete tokenIdToIdentity[identityData[identity].boundNft];
        delete identityData[identity].boundNft;
        _resetIdentityData(identity);

        shinToken.transfer(msg.sender, identityData[identity].stakedTokens);
        identityData[identity].stakedTokens = 0;
        emit StakeUpdate(identity, 0);
        identityData[identity].lastUpdated = block.timestamp;
    }
```

The `burn` function in ShinkaiNft triggers the `_beforeTokenTransfer` method, which then calls back to the registry contract, executing the `claimRewards` function.

```
    function _beforeTokenTransfers(address from, address, /*to*/ uint256 startTokenId, uint256 /*quantity*/ )
        internal
        override
    {
        if (from == address(0)) return;
        string memory identity = registry.tokenIdToIdentity(startTokenId);
        registry.claimRewards(identity);
    }
```

The `claimRewards` function accumulates both staking and delegation rewards.

```
    function claimRewards(string memory identity) public returns (uint256 tokensAccrued) {
        tokensAccrued += claimStakingRewards(identity);
        tokensAccrued += claimDelegationRewards(identity);
    }
```

Since the `claimRewards` function already includes a call to claimStakingRewards, the initial call to `claimStakingRewards` in the `unclaimIdentity` function is redundant and can be removed to optimize gas costs.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation

Consider removing the redundant invocation.

Remediation Comment

Consider removing the redundant invocation.

References

["https://github.com/dcSpark/shinkai-contracts/blob/52a83ce548eac47022b61f7aed23bfaa3e640c7d/src/ShinkaiRegistry.sol#L427"]

7.8 (HAL-08) UNLOCKED PRAGMA COMPILERS

// INFORMATIONAL

Description

The files in scope currently use floating pragma version `^0.8.20`, which means that the code can be compiled by any compiler version that is greater than or equal to `0.8.0`, and less than `0.9.0`. It is recommended that contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively.
Additionally, using a newer compiler version that introduces default optimizations, including unchecked overflow for gas efficiency, presents an opportunity for further optimization.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation

Lock the pragma version to the same version used during development and testing.

Remediation Comment

Lock the pragma version to the same version used during development and testing.

8. AUTOMATED TESTING

Static Analysis Report

Description

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results

ShinkaiToken ERC20 analysis

```
# Check ShinkaiToken

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> (uint256) (correct return type)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> (uint256) (correct return type)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> (bool) (correct return type)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> (bool) (correct return type)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> (bool) (correct return type)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> (uint256) (correct return type)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> (string) (correct return type)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> (string) (correct return type)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> (uint8) (correct return type)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


        [ ] ShinkaiToken is not protected for the ERC20 approval race condition
```

ShinkaiNft ERC721 analysis

```
# Check ShinkaiNft

## Check functions
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> (uint256) (correct return type)
        [✓] balanceOf(address) is view
[✓] ownerOf(uint256) is present
        [✓] ownerOf(uint256) -> (address) (correct return type)
        [✓] ownerOf(uint256) is view
[✓] safeTransferFrom(address,address,uint256,bytes) is present
        [✓] safeTransferFrom(address,address,uint256,bytes) -> () (correct return type)
        [✓] Transfer(address,address,uint256) is emitted
[✓] safeTransferFrom(address,address,uint256) is present
        [✓] safeTransferFrom(address,address,uint256) -> () (correct return type)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return type)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return type)
        [✓] Approval(address,address,uint256) is emitted
[✓] setApprovalForAll(address,bool) is present
        [✓] setApprovalForAll(address,bool) -> () (correct return type)
        [✓] ApprovalForAll(address,address,bool) is emitted
[✓] getApproved(uint256) is present
        [✓] getApproved(uint256) -> (address) (correct return type)
        [✓] getApproved(uint256) is view
[✓] isApprovedForAll(address,address) is present
        [✓] isApprovedForAll(address,address) -> (bool) (correct return type)
        [✓] isApprovedForAll(address,address) is view
[✓] supportsInterface(bytes4) is present
        [✓] supportsInterface(bytes4) -> (bool) (correct return type)
        [✓] supportsInterface(bytes4) is view
[✓] name() is present
        [✓] name() -> (string) (correct return type)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> (string) (correct return type)
[✓] tokenURI(uint256) is present
        [✓] tokenURI(uint256) -> (string) (correct return type)

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
        [✓] parameter 2 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
        [✓] parameter 2 is indexed
[✓] ApprovalForAll(address,address,bool) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
```

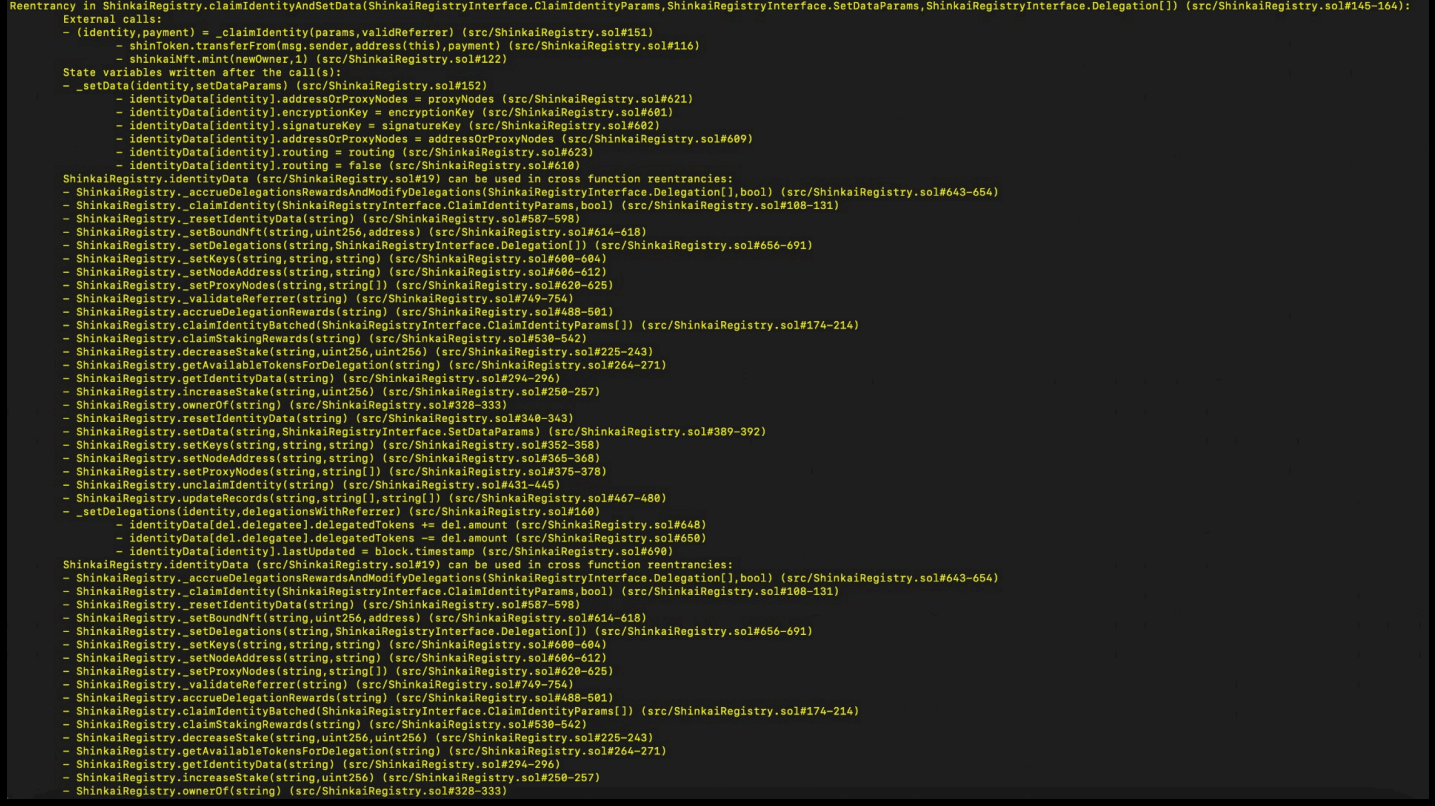
ShinkaiNFT.sol

```
INFO:Detectors:
ShinkaiNft.tokenURI(uint256) (src/ShinkaiNft.sol#59-63) calls abi.encodePacked() with multiple dynamic arguments:
        - string(abi.encodePacked(baseURI,identityOf(tokenId))) (src/ShinkaiNft.sol#62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#abi-encodePacked-collision
INFO:Detectors:
Contract locking ether found:
        Contract ShinkaiNft (src/ShinkaiNft.sol#10-81) has payable functions:
         - ERC721A.approve(address,uint256) (node_modules/erc721a/contracts/ERC721A.sol#425-435)
         - ERC721A.transferFrom(address,address,uint256) (node_modules/erc721a/contracts/ERC721A.sol#540-601)
         - ERC721A.safeTransferFrom(address,address,uint256) (node_modules/erc721a/contracts/ERC721A.sol#606-612)
         - ERC721A.safeTransferFrom(address,address,uint256,bytes) (node_modules/erc721a/contracts/ERC721A.sol#629-640)
         - IERC721A.safeTransferFrom(address,address,uint256,bytes) (node_modules/erc721a/contracts/IERC721A.sol#168-173)
         - IERC721A.safeTransferFrom(address,address,uint256) (node_modules/erc721a/contracts/IERC721A.sol#178-182)
         - IERC721A.transferFrom(address,address,uint256) (node_modules/erc721a/contracts/IERC721A.sol#200-204)
         - IERC721A.approve(address,uint256) (node_modules/erc721a/contracts/IERC721A.sol#220)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
ShinkaiNft._beforeTokenTransfers(address,address,uint256,uint256) (src/ShinkaiNft.sol#69-76) ignores return value by registry.claimRewards(identity) (src/ShinkaiNft.sol#75)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

ShinkaiRegistry.sol

```
INFO:Detectors:
ShinkaiRegistry._claimIdentity(ShinkaiRegistryInterface.ClaimIdentityParams,bool) (src/ShinkaiRegistry.sol#108-131) ignores return value by shinToken.transferFrom(msg.sender,address(this),payment) (src/ShinkaiRegistry.sol#116)
ShinkaiRegistry.claimIdentityBatched(ShinkaiRegistryInterface.ClaimIdentityParams[]) (src/ShinkaiRegistry.sol#174-214) ignores return value by shinToken.transferFrom(msg.sender,address(this),totalPayment) (src/ShinkaiRegistry.sol#213)
ShinkaiRegistry.decreaseStake(string,uint256,uint256) (src/ShinkaiRegistry.sol#225-243) ignores return value by shinToken.transfer(msg.sender,amount) (src/ShinkaiRegistry.sol#239)
ShinkaiRegistry.increaseStake(string,uint256) (src/ShinkaiRegistry.sol#250-257) ignores return value by shinToken.transferFrom(msg.sender,address(this),amount) (src/ShinkaiRegistry.sol#252)
ShinkaiRegistry.unclaimIdentity(string) (src/ShinkaiRegistry.sol#431-445) ignores return value by shinToken.transfer(msg.sender,identityData[identity].stakedTokens) (src/ShinkaiRegistry.sol#441)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
Reentrancy in ShinkaiRegistry._claimIdentity(ShinkaiRegistryInterface.ClaimIdentityParams,bool) (src/ShinkaiRegistry.sol#108-131):
	External calls:
	- shinToken.transferFrom(msg.sender,address(this),payment) (src/ShinkaiRegistry.sol#116)
	State variables written after the call(s):
	- identityData[identity].stakedTokens = payment (src/ShinkaiRegistry.sol#117)
	ShinkaiRegistry.identityData (src/ShinkaiRegistry.sol#19) can be used in cross function reentrancies:
	- ShinkaiRegistry._accrueDelegationsRewardsAndModifyDelegations(ShinkaiRegistryInterface.Delegation[],bool) (src/ShinkaiRegistry.sol#643-654)
	- ShinkaiRegistry._claimIdentity(ShinkaiRegistryInterface.ClaimIdentityParams,bool) (src/ShinkaiRegistry.sol#108-131)
	- ShinkaiRegistry._resetIdentityData(string) (src/ShinkaiRegistry.sol#587-598)
	- ShinkaiRegistry._setBoundNft(string,uint256,address) (src/ShinkaiRegistry.sol#614-618)
	- ShinkaiRegistry._setDelegations(string,ShinkaiRegistryInterface.Delegation[]) (src/ShinkaiRegistry.sol#656-691)
	- ShinkaiRegistry._setKeys(string,string,string) (src/ShinkaiRegistry.sol#600-604)
	- ShinkaiRegistry._setNodeAddress(string,string) (src/ShinkaiRegistry.sol#606-612)
	- ShinkaiRegistry._setProxyNodes(string,string[]) (src/ShinkaiRegistry.sol#620-625)
	- ShinkaiRegistry._validateReferrer(string) (src/ShinkaiRegistry.sol#749-754)
	- ShinkaiRegistry.accrueDelegationRewards(string) (src/ShinkaiRegistry.sol#488-501)
	- ShinkaiRegistry.claimIdentityBatched(ShinkaiRegistryInterface.ClaimIdentityParams[]) (src/ShinkaiRegistry.sol#174-214)
	- ShinkaiRegistry.claimStakingRewards(string) (src/ShinkaiRegistry.sol#530-542)
	- ShinkaiRegistry.decreaseStake(string,uint256,uint256) (src/ShinkaiRegistry.sol#225-243)
	- ShinkaiRegistry.getAvailableTokensForDelegation(string) (src/ShinkaiRegistry.sol#264-271)
	- ShinkaiRegistry.getIdentityData(string) (src/ShinkaiRegistry.sol#294-296)
	- ShinkaiRegistry.increaseStake(string,uint256) (src/ShinkaiRegistry.sol#250-257)
	- ShinkaiRegistry.ownerOf(string) (src/ShinkaiRegistry.sol#328-333)
	- ShinkaiRegistry.resetIdentityData(string) (src/ShinkaiRegistry.sol#340-343)
	- ShinkaiRegistry.setData(string,ShinkaiRegistryInterface.SetDataParams) (src/ShinkaiRegistry.sol#389-392)
	- ShinkaiRegistry.setKeys(string,string,string) (src/ShinkaiRegistry.sol#352-358)
	- ShinkaiRegistry.setNodeAddress(string,string) (src/ShinkaiRegistry.sol#365-368)
	- ShinkaiRegistry.setProxyNodes(string,string[]) (src/ShinkaiRegistry.sol#375-378)
	- ShinkaiRegistry.unclaimIdentity(string) (src/ShinkaiRegistry.sol#431-445)
	- ShinkaiRegistry.updateRecords(string,string[],string[]) (src/ShinkaiRegistry.sol#467-480)
	- identityData[identity].lastUpdated = block.timestamp (src/ShinkaiRegistry.sol#118)
	ShinkaiRegistry.identityData (src/ShinkaiRegistry.sol#19) can be used in cross function reentrancies:
	- ShinkaiRegistry._accrueDelegationsRewardsAndModifyDelegations(ShinkaiRegistryInterface.Delegation[],bool) (src/ShinkaiRegistry.sol#643-654)
	- ShinkaiRegistry._claimIdentity(ShinkaiRegistryInterface.ClaimIdentityParams,bool) (src/ShinkaiRegistry.sol#108-131)
	- ShinkaiRegistry._resetIdentityData(string) (src/ShinkaiRegistry.sol#587-598)
	- ShinkaiRegistry._setBoundNft(string,uint256,address) (src/ShinkaiRegistry.sol#614-618)
	- ShinkaiRegistry._setDelegations(string,ShinkaiRegistryInterface.Delegation[]) (src/ShinkaiRegistry.sol#656-691)
	- ShinkaiRegistry._setKeys(string,string,string) (src/ShinkaiRegistry.sol#600-604)
	- ShinkaiRegistry._setNodeAddress(string,string) (src/ShinkaiRegistry.sol#606-612)
	- ShinkaiRegistry._setProxyNodes(string,string[]) (src/ShinkaiRegistry.sol#620-625)
	- ShinkaiRegistry._validateReferrer(string) (src/ShinkaiRegistry.sol#749-754)
	- ShinkaiRegistry.accrueDelegationRewards(string) (src/ShinkaiRegistry.sol#488-501)
	- ShinkaiRegistry.claimIdentityBatched(ShinkaiRegistryInterface.ClaimIdentityParams[]) (src/ShinkaiRegistry.sol#174-214)
	- ShinkaiRegistry.claimStakingRewards(string) (src/ShinkaiRegistry.sol#530-542)
	- ShinkaiRegistry.decreaseStake(string,uint256,uint256) (src/ShinkaiRegistry.sol#225-243)
```

```
Reentrancy in ShinkaiRegistry._claimIdentity(ShinkaiRegistryInterface.ClaimIdentityParams,bool) (src/ShinkaiRegistry.sol#108-131):
	External calls:
	- shinToken.transferFrom(msg.sender,address(this),payment) (src/ShinkaiRegistry.sol#116)
	- shinkaiNft.mint(newOwner,1) (src/ShinkaiRegistry.sol#122)
	State variables written after the call(s):
	- _setBoundNft(identity,tokenId,newOwner) (src/ShinkaiRegistry.sol#123)
		- identityData[identity].boundNft = tokenId (src/ShinkaiRegistry.sol#615)
	ShinkaiRegistry.identityData (src/ShinkaiRegistry.sol#19) can be used in cross function reentrancies:
	- ShinkaiRegistry._accrueDelegationsRewardsAndModifyDelegations(ShinkaiRegistryInterface.Delegation[],bool) (src/ShinkaiRegistry.sol#643-654)
	- ShinkaiRegistry._claimIdentity(ShinkaiRegistryInterface.ClaimIdentityParams,bool) (src/ShinkaiRegistry.sol#108-131)
	- ShinkaiRegistry._resetIdentityData(string) (src/ShinkaiRegistry.sol#587-598)
	- ShinkaiRegistry._setBoundNft(string,uint256,address) (src/ShinkaiRegistry.sol#614-618)
	- ShinkaiRegistry._setDelegations(string,ShinkaiRegistryInterface.Delegation[]) (src/ShinkaiRegistry.sol#656-691)
	- ShinkaiRegistry._setKeys(string,string,string) (src/ShinkaiRegistry.sol#600-604)
	- ShinkaiRegistry._setNodeAddress(string,string) (src/ShinkaiRegistry.sol#606-612)
	- ShinkaiRegistry._setProxyNodes(string,string[]) (src/ShinkaiRegistry.sol#620-625)
	- ShinkaiRegistry._validateReferrer(string) (src/ShinkaiRegistry.sol#749-754)
	- ShinkaiRegistry.accrueDelegationRewards(string) (src/ShinkaiRegistry.sol#488-501)
	- ShinkaiRegistry.claimIdentityBatched(ShinkaiRegistryInterface.ClaimIdentityParams[]) (src/ShinkaiRegistry.sol#174-214)
	- ShinkaiRegistry.claimStakingRewards(string) (src/ShinkaiRegistry.sol#530-542)
	- ShinkaiRegistry.decreaseStake(string,uint256,uint256) (src/ShinkaiRegistry.sol#225-243)
	- ShinkaiRegistry.getAvailableTokensForDelegation(string) (src/ShinkaiRegistry.sol#264-271)
	- ShinkaiRegistry.getIdentityData(string) (src/ShinkaiRegistry.sol#294-296)
	- ShinkaiRegistry.increaseStake(string,uint256) (src/ShinkaiRegistry.sol#250-257)
	- ShinkaiRegistry.ownerOf(string) (src/ShinkaiRegistry.sol#328-333)
	- ShinkaiRegistry.resetIdentityData(string) (src/ShinkaiRegistry.sol#340-343)
	- ShinkaiRegistry.setData(string,ShinkaiRegistryInterface.SetDataParams) (src/ShinkaiRegistry.sol#389-392)
	- ShinkaiRegistry.setKeys(string,string,string) (src/ShinkaiRegistry.sol#352-358)
	- ShinkaiRegistry.setNodeAddress(string,string) (src/ShinkaiRegistry.sol#365-368)
	- ShinkaiRegistry.setProxyNodes(string,string[]) (src/ShinkaiRegistry.sol#375-378)
	- ShinkaiRegistry.unclaimIdentity(string) (src/ShinkaiRegistry.sol#431-445)
	- ShinkaiRegistry.updateRecords(string,string[],string[]) (src/ShinkaiRegistry.sol#467-480)
Reentrancy in ShinkaiRegistry.claimDelegationRewards(string) (src/ShinkaiRegistry.sol#508-518):
	External calls:
	- shinToken.mint(identityOwner,tokensAccrued) (src/ShinkaiRegistry.sol#514)
	State variables written after the call(s):
	- identityDelegationAccrued[identity] = 0 (src/ShinkaiRegistry.sol#515)
	ShinkaiRegistry.identityDelegationAccrued (src/ShinkaiRegistry.sol#24) can be used in cross function reentrancies:
	- ShinkaiRegistry.accrueDelegationRewards(string) (src/ShinkaiRegistry.sol#488-501)
	- ShinkaiRegistry.claimDelegationRewards(string) (src/ShinkaiRegistry.sol#508-518)
	- ShinkaiRegistry.identityDelegationAccrued (src/ShinkaiRegistry.sol#24)
Reentrancy in ShinkaiRegistry.claimIdentity(ShinkaiRegistryInterface.ClaimIdentityParams) (src/ShinkaiRegistry.sol#94-106):
	External calls:
	- (identity,payment) = _claimIdentity(params,validReferrer) (src/ShinkaiRegistry.sol#97)
		- shinToken.transferFrom(msg.sender,address(this),payment) (src/ShinkaiRegistry.sol#116)
		- shinkaiNft.mint(newOwner,1) (src/ShinkaiRegistry.sol#122)
	State variables written after the call(s):
	- _setDelegations(identity,delegations) (src/ShinkaiRegistry.sol#104)
		- identityData[del.delegatee].delegatedTokens += del.amount (src/ShinkaiRegistry.sol#648)
		- identityData[del.delegatee].delegatedTokens -= del.amount (src/ShinkaiRegistry.sol#650)
		- identityData[identity].lastUpdated = block.timestamp (src/ShinkaiRegistry.sol#690)
	ShinkaiRegistry.identityData (src/ShinkaiRegistry.sol#19) can be used in cross function reentrancies:
	- ShinkaiRegistry._accrueDelegationsRewardsAndModifyDelegations(ShinkaiRegistryInterface.Delegation[],bool) (src/ShinkaiRegistry.sol#643-654)
	- ShinkaiRegistry._claimIdentity(ShinkaiRegistryInterface.ClaimIdentityParams,bool) (src/ShinkaiRegistry.sol#108-131)
	- ShinkaiRegistry._resetIdentityData(string) (src/ShinkaiRegistry.sol#587-598)
	- ShinkaiRegistry._setBoundNft(string,uint256,address) (src/ShinkaiRegistry.sol#614-618)
	- ShinkaiRegistry._setDelegations(string,ShinkaiRegistryInterface.Delegation[]) (src/ShinkaiRegistry.sol#656-691)
	- ShinkaiRegistry._setKeys(string,string,string) (src/ShinkaiRegistry.sol#600-604)
	- ShinkaiRegistry._setNodeAddress(string,string) (src/ShinkaiRegistry.sol#606-612)
	- ShinkaiRegistry._setProxyNodes(string,string[]) (src/ShinkaiRegistry.sol#620-625)
```



Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.