SIGN Staking

SECURITY ASSESSMENT REPORT

June 6, 2025

Prepared for

Contents

1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensure the safety, reliability, and success of blockchain projects. Our services include comprehensive smart contract audits, secure design and architecture consultancy, and smart contract development across leading blockchain platforms such as Ethereum (Solidity), Starknet (Cairo), and Solana (Rust).

At CODESPECT, we are committed to build secure, resilient blockchain infrastructures. We provide strategic guidance and technical expertise, working closely with our partners from concept development through deployment. Our team consists of blockchain security experts and seasoned engineers who apply the latest auditing and security methodologies to help prevent exploits and vulnerabilities in your smart contracts.

Smart Contract Auditing: Security is at the core of everything we do at CODESPECT. Our auditors conduct thorough security assessments of smart contracts written in Solidity, Cairo, and Rust, ensuring that they function as intended without vulnerabilities. We specialize in providing tailored security solutions for projects on EVM-compatible chains and Starknet. Our audit process is highly collaborative, keeping clients involved every step of the way to ensure transparency and security. Our team is also dedicated to cutting-edge research, ensuring that we stay ahead of emerging threats.

Secure Design & Architecture Consultancy: At CODESPECT, we believe that secure development begins at the design phase. Our consultancy services offer deep insights into secure smart contract architecture and blockchain system design, helping you build robust, secure, and scalable decentralized applications. Whether you're working with Ethereum, Starknet, or other blockchain platforms, our team helps you navigate the complexity of blockchain development with confidence.

Tailored Cybersecurity Solutions: CODESPECT offers specialized cybersecurity solutions designed to minimize risks associated with traditional attack vectors, such as phishing, social engineering, and Web2 vulnerabilities. Our solutions are crafted to address the unique security needs of blockchain-based applications, reducing exposure to attacks and ensuring that all aspects of the system are fortified.

With a focus on the intersection of security and innovation, CODESPECT strives to be a trusted partner for blockchain projects at every stage of development and for each aspect of security.

2 Disclaimer

Limitations of this Audit: This report is based solely on the materials and documentation provided to CODESPECT for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented in this report may not be comprehensive and may not identify all possible vulnerabilities. CODESPECT provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, is entirely at your own risk.

Inherent Risks of Blockchain Technology: Blockchain technology is still evolving and is inherently subject to unknown risks and vulnerabilities. This review focuses exclusively on the smart contract code provided and does not cover the compiler layer, underlying programming language elements beyond the reviewed code, or any other potential security risks that may exist outside of the code itself.

Purpose and Reliance of this Report: This report should not be viewed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. Third parties should not rely on this report for any purpose, including making decisions related to investments or purchases.

Liability Disclaimer: To the maximum extent permitted by law, CODESPECT disclaims all liability for the contents of this report and any related services or products that arise from your use of it. This includes but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services: CODESPECT does not warrant, endorse, or assume responsibility for any third-party products or services mentioned in this report, including any open-source or third-party software, code, libraries, materials, or information that may be linked to, referenced by, or accessible through this report. CODESPECT is not responsible for monitoring any transactions between you and third-party providers. We strongly recommend conducting thorough due diligence and exercising caution when engaging with third-party products or services, just as you would for any other product or service transaction.

Further Recommendations: We advise clients to schedule a re-audit after any significant changes to the codebase to ensure ongoing security and reduce the risk of newly introduced vulnerabilities. Additionally, we recommend implementing a bug bounty program to incentivize external developers and security researchers to identify and disclose potential vulnerabilities safely and responsibly.

Disclaimer of Advice: FOR AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY ASSOCIATED SERVICES OR MATERIALS SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Risk Classification

Severity Level	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Table 1: Risk Classification Matrix based on Likelihood and Impact

3.1 Impact

- **High** - Results in a substantial loss of assets (more than 10%) within the protocol or causes significant disruption to the majority of users.
- **Medium** - Losses affect less than 10% globally or impact only a portion of users, but are still considered unacceptable.
- **Low** - Losses may be inconvenient but are manageable, typically involving issues like griefing attacks that can be easily resolved or minor inefficiencies such as gas costs.

3.2 Likelihood

- **High** - Very likely to occur, either easy to exploit or difficult but highly incentivized.
- **Medium** - Likely only under certain conditions or moderately incentivized.
- **Low** - Unlikely unless specific conditions are met, or there is little-to-no incentive for exploitation.

3.3 Action Required for Severity Levels

- **Critical** - Must be addressed immediately if already deployed.
- **High** - Must be resolved before deployment (or urgently if already deployed).
- **Medium** - It is recommended to fix.
- **Low** - Can be fixed if desired but is not crucial.

In addition to High, Medium, and Low severity levels, CODESPECT utilizes two other categories for findings: **Informational** and **Best Practices**.

a) **Informational** findings do not pose a direct security risk but provide useful information the audit team wants to communicate formally.

b) **Best Practices** findings indicate that certain portions of the code deviate from established smart contract development standards.

4 Executive Summary

This document presents the results of a security assessment conducted by CODESPECT for Sign. Sign Protocol is an omni-chain attestation protocol that enables users to freely attest to and verify information on-chain.

The scope of this audit focused on the EVM-based `SIGNStaking` contract. This contract allows users to stake SIGN tokens and earn yield in the form of additional SIGN tokens. The yield can be further boosted by staking a SIGN NFT, which grants enhanced reward rates.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of programs, execution testing.

CODESPECT found six points of attention, one classified as `High`, one classified as `Medium`, one classified as `Low`, one classified as `Informational`, and two classified as `Best Practices`. All of the issues are summarised in Table 2.

Organisation of the document is as follows:

- **Section 5** summarizes the audit.
- **Section 6** describes the system overview.
- **Section 7** presents the issues.
- **Section 8** discusses the documentation provided by the client for this audit.
- **Section 9** presents the compilation and tests.

Issues found:

Severity	Unresolved	Fixed	Acknowledged
High	0	1	0
Medium	0	1	0
Low	0	1	0
Informational	0	0	1
Best Practices	0	1	1
Total	**0**	**4**	**2**

Table 2: Summary of Unresolved, Fixed, and Acknowledged Issues

5 Audit Summary

Audit Type	Security Review
Project Name	SIGN Staking
Type of Project	Staking Contract
Duration of Engagement	5 Days
Duration of Fix Review Phase	1 Day
Draft Report	June 4, 2025
Final Report	June 6, 2025
Repository	sign-token-staking-evm
Commit (Audit V1)	735cc008ea45c4a54e87761217218fb3983e69d5
Commit (Audit V2)	09bd167d69a5c1689e21b6a605715369a5e667ae
Commit (Audit V3)	a8bb7c48d6169c8ba8fde052a56765f2a380f0c1
Commit (Final)	4506cd236a2e777df12c8384218ec4bcd2ee02cf
Documentation Assessment	High
Test Suite Assessment	High
Auditors	talfao, JecikPo

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	LoC
1	SIGNStaking.sol	221
	Total	**221**

5.2 Findings Overview

	Finding	Severity	Update
1	Interest not claimable after unstaking whole stake	High	Fixed
2	Cooldown mechanism is incorrectly implemented	Medium	Fixed
3	APR calculation does not support fractional values	Low	Fixed
4	NFT contract has features which can influence unstaking	Info	Acknowledged
5	Checks effects interactions pattern not followed by several functions	Best Practices	Fixed
6	Lack of guardrails on key protocol parameter values	Best Practices	Acknowledged

6 System Overview

Sign has deployed a staking contract for its newly launched SIGN token. This contract allows any user holding SIGN tokens to stake their assets and earn interest in the form of additional SIGN tokens. The interest rate can be boosted by staking a SIGN NFT.

Note: This system overview follows the V3 commit version of the contracts, which introduced important changes to the unstaking logic.

Users can stake any amount of SIGN tokens, and interest begins accruing immediately. Staking is initiated via the `stake(...)` function, while staking of a SIGN NFT is performed via the `stakeNFT(...)` function:

```
function stake(uint256 amount) external nonReentrant whenConfigured whenNotPaused;

function stakeNFT(uint256 tokenId) external nonReentrant whenConfigured whenNotPaused;
```

Interest accumulation follows a checkpoint-based mechanism. Upon each staking or unstaking action, interest is calculated using the following formula:

$$\text{currentPeriodInterest} = \frac{\text{principalAmount} \times \text{APR} \times \Delta t}{\text{aprPrecision} \times \text{YEAR}}$$

where `principalAmount` is the currently staked amount, `APR` is the annual percentage yield defined by the Sign team (boosted if an NFT is staked), and Δt = currentTimestamp − lastInterestCalculation represents the time elapsed since the last interest update.

Accumulated interest is not compounded. It can be claimed at any time using the `claimInterest(...)` function, which transfers the specified amount of earned SIGN tokens to the user. The SIGN protocol must maintain a sufficient balance of SIGN tokens within the contract to allow successful interest claims.

```
function claimInterest(uint256 amount) external nonReentrant whenNotPaused;
```

The SIGN reserves used for interest payouts are tracked separately, hence interest payout from other staker's principal is avoided.

Unstaking is divided into three functions:

```
function unstake(uint256 amount) external nonReentrant whenNotPaused;

function claimCooldowns(uint256[] calldata indices) external nonReentrant whenNotPaused;

function unstakeNFT() external nonReentrant whenNotPaused;
```

Withdrawal of the stake follows a two-step process. To initiate a withdrawal, the user must first call `unstake(...)` to begin a cooldown period. During this period, the specified amount is excluded from further interest accrual. Once the cooldown expires, the user can complete the withdrawal by calling `claimCooldown(...)`. Multiple cooldowns can run in parallel, if user decides to unstake another amount while a cooldown is already running. Additionally, users can unstake their SIGN NFT at any time using `unstakeNFT(...)`, which removes the APR boost for the duration during which the NFT remains unstaked.

The contract also includes certain centralized control features. Specifically, it implements a pausing mechanism, allowing the contract administrator to disable all user-facing external functions in emergency scenarios. Furthermore, the contract is upgradeable via the UUPS proxy pattern, giving the owner the ability to deploy future logic changes while preserving user state.

7 Issues

7.1 [High] Interest not claimable after unstaking whole stake

File(s): SIGNStaking.sol

Description: The `claimInterest(...)` function allows the staker to claim interest. This function was added to decouple claiming the interest from unstaking.

`claimInterest(...)` when called updates the interest according to the latest timestamp:

```
_updateInterest(msg.sender);
```

The `_updateInterest(...)` uses `calculateInterest(...)` to return the latest accrued interest and assigns the value to `userStake.accumulatedInterest`.

The issue happens when the user first unstakes his or her entire stake, but does not collect interest before. After user empties the stake the `userStake.principalAmount` is set to 0. This causes the following condition to be hit inside the `calculateInterest(...)`:

```
if (userStake.principalAmount == 0) {
    return 0;
}
```

Which means that the user's `userStake.accumulatedInterest` will also be set to zero. This will cause the following line at `claimInterest(...)` to revert:

```
userStake.accumulatedInterest -= amount;
```

The user will be unable to claim interest, the only way to recover from this is to call `stake(...)` again, but that will reset the `userStake.accumulatedInterest` back to 0.

Impact: Loss of accrued interest when claiming after unstaking. The possibility of such a scenario is high, hence the High impact of this issue.

Recommendation(s): Change the `calculateInterest(...)` return value from 0 to `userStake.accumulatedInterest` when `userStake.principalAmount` is 0.

Status: Fixed

Update from TokenTable: f5c92216ffad8b1e875f2e5cde63c7932a399354

7.2 [Medium] Cooldown mechanism is incorrectly implemented

File(s): `SIGNStaking.sol`

Description: The staking contract enforces a cooldown period for withdrawals. To withdraw their stake, a user must first call `unstake(...)`, which initiates a cooldown grace period. Once the period elapses, the user must call `unstake(...)` again to complete the withdrawal of their deposited stake.

However, there are two issues with this mechanism:

1. After initiating the cooldown, the user can still make additional deposits. These new deposits are not subject to a new cooldown period;
2. Stake continues to accrue interest even after the cooldown has been initiated. This creates an exploitable scenario: a user can stake SIGN tokens and immediately call `unstake(...)`—without the intention to fully unstake. The stake will keep earning interest during the cooldown, and after it ends, the user can instantly withdraw, bypassing the intended staking constraints;

Impact: The cooldown mechanism is flawed and can be bypassed, allowing users to game the system and withdraw stake with accumulated interest, without respecting the intended waiting period.

Recommendation(s):

- Prevent users from depositing additional tokens while an unstaking process is active;
- Stop interest accrual immediately upon initiating an unstake;

Status: Fixed

Update from TokenTable: 09bd167d69a5c1689e21b6a605715369a5e667ae

7.3 [Low] APR calculation does not support fractional values

File(s): `SIGNStaking.sol`

Description: A user's stake accrues interest based on the duration it remains in the contract. Interest is calculated during each staking or unstaking action (including NFT stake/unstake events). The interest calculation is based on the time elapsed since the last checkpoint, using the following formula:

```
uint256 currentPeriodInterest = (userStake.amount * currentAPY * timeElapsed) / (100 * YEAR);
```

The divisor `100 * YEAR` implies that the APR is expressed as whole percentages (e.g., 11% but not 11.5%). This restricts the protocol from setting fractional APR values, which are commonly used in DeFi systems.

Following confirmation with the Sign team, it was acknowledged that this limitation stems from the current implementation. The team agreed that the base divisor should be increased to support fractional APRs.

Impact: Limits the protocol's ability to define APRs with decimal precision (e.g., 11.5%), reducing flexibility in interest rate configuration.

Recommendation(s): Increase the base denominator to enable more granular APR values (e.g., use a base of 10,000 instead of 100).

Status: Fixed

Update from TokenTable: f97c626a7f09bf8a54d2dca2c4093a79a3ffa505

7.4 [Info] NFT contract has features which can influence unstaking

File(s): `SIGNStaking.sol`

Description: The SIGN staking contract allows users to stake a SIGN NFT to boost their APR. The implementation of the NFT contract can be found here. This NFT contract includes two functions that may interfere with the unstaking process:

```
function burn(uint256[] calldata ids) external onlyOwner {
    for (uint256 i = 0; i < ids.length; i++) {
        _burn(ids[i]);
    }
}

function freeze(uint256[] calldata ids) external onlyOwner {
    for (uint256 i = 0; i < ids.length; i++) {
        _getSignNFTStorage().frozenIds[ids[i]] = true;
    }
}
```

The `burn(...)` function permanently removes the NFT, while `freeze(...)` prevents it from being transferred. If either of these actions is performed on an NFT currently staked in the `SIGNStaking` contract, the user will be unable to withdraw their stake. This is because the following line in the `unstake(...)` function will revert:

```
if (userStake.nftStakeTime > 0) {
    $.nftContract.safeTransferFrom(address(this), msg.sender, userStake.nftTokenId);
}
```

Additionally, users may still be accumulating boosted interest despite the NFT being frozen or burned, which is likely unintended behaviour.

Impact: A user's stake can become permanently locked if the associated NFT is frozen or burned. While this is primarily a centralisation risk (since Sign controls both contracts), it could still impact user trust and usability.

Recommendation(s):

- Allow users to unstake their SIGN tokens even if the associated NFT is no longer transferable.
- Stop boosted interest accrual for NFTs that are frozen or burned.

Status: Acknowledged

Update from TokenTable: Acknowledged

7.5 [Best Practice] Checks effects interactions pattern not followed by several functions

File(s): SIGNStaking.sol

Description: Several functions in the contract do not follow the Checks-Effects-Interactions (CEI) pattern. In these cases, state variables are updated *after* performing external interactions, which goes against best practices:

```
function unstakeNFT() external nonReentrant {
    // ...
    $.nftContract.safeTransferFrom(address(this), msg.sender, tokenId);
    userStake.nftTokenId = 0;
    userStake.nftStakeTime = 0;
    // ...
}

function stakeNFT(uint256 tokenId) external nonReentrant whenConfigured whenNotPaused {
    // ...
    $.nftContract.transferFrom(msg.sender, address(this), tokenId);
    userStake.nftTokenId = tokenId;
    userStake.nftStakeTime = block.timestamp;
    // ...
}

function stake(uint256 amount) external nonReentrant whenConfigured whenNotPaused {
    // ...
    $.signToken.safeTransferFrom(msg.sender, address(this), amount);
    userStake.principalAmount += amount;
    // ...
}
```

Note: Some of these external interactions do not pose a reentrancy risk, but it is still best practice to follow the CEI pattern.

Impact: There is no danger of reentrancy as functions are protected by the `nonReentrant` modifier, however, it is a best practice to always follow the CEI pattern.

Recommendation(s): Reorder the statements in affected functions so that all state changes occur before any external interaction.

Status: Fixed

Update from TokenTable: e14dcc9b159afb4699e80a3ce820380190fde774

7.6 [Best Practice] Lack of guardrails on key protocol parameter values

File(s): `SIGNStaking.sol`

Description: The protocol allows for setting various parameters defining the staking assets, unstaking rules and interest calculations. The following struct defines the controllable parameters:

```
struct SignStakingStorage {
    IERC20 signToken;
    IERC721 nftContract;
    uint256 standardAPY;
    uint256 boostedAPY;
    uint256 cooldownPeriod;
    /// [...]
}
```

They can be set by a collection of `set` functions guarded by `onlyOwner` modifier. However the protocol lacks safeguards against setting of incorrect values that will damage user experience or endanger the reserve which is used for interest payments. Specifically:

- `setSignToken(...)` and `setNFTContract(...)` could change the token and NFT once users are already staked and hence block unstaking functions;
- `setStandardAPY(...)` and `setBoostedAPY(...)` can be set too high and hence contribute to faster than expected reserve depletion;
- `setCooldownPeriod(...)` could set the cooldown period to unreasonably large value and prevent users from unstaking;

Impact: Hampered user experience or reserve depletion.

Recommendation(s): Set limits on the APR values. Block changing of token and NFT addresses after first user stake. Limit maximum value of the cooldown period.

Status: Acknowledged

Update from TokenTable: Acknowledged

8 Evaluation of Provided Documentation

Sign team provided documentation in two formats:

- **Natspec Comments:** The code includes inline comments using the Natspec format, explaining the purpose of complex functionalities and aiding the understanding of individual functions.
- **Specification Document:** A private specification document was shared with CODESPECT, outlining the intended behavior of the staking contract. This document was used to validate that the implementation aligns with the original design intentions.

The documentation provided by the Sign team was appropriate for the complexity and scale of the protocol. It enabled efficient onboarding and a thorough review of the staking logic.

Additionally, the Sign team remained consistently available and responsive throughout the audit process, promptly addressing all questions and clarifications raised by CODESPECT.

9 Test Suite Evaluation

9.1 Compilation Output

```
> forge build

[] Compiling...
[] Compiling 47 files with Solc 0.8.29
[] Solc 0.8.29 finished in 1.65s
Compiler run successful!
```

9.2 Tests Output

```
> forge test
Ran 37 tests for test/SIGNStaking.t.sol:SIGNStakingTest
[PASS] testFuzz_CalculateInterest_BoostedAPR(uint256,uint256) (runs: 256, : 210075, ~: 210194)
[PASS] testFuzz_CalculateInterest_StandardAPR(uint256,uint256) (runs: 256, : 130860, ~: 131012)
[PASS] testFuzz_ClaimInterest(uint256,uint256) (runs: 256, : 173273, ~: 173328)
[PASS] testFuzz_Stake(uint256) (runs: 256, : 132979, ~: 132730)
[PASS] testFuzz_StakeMultipleTimes(uint256,uint256) (runs: 256, : 163170, ~: 164051)
[PASS] testFuzz_StakeNFT(uint256) (runs: 256, : 170102, ~: 179530)
[PASS] test_ClaimAllCooldownsAtOnce() (gas: 352012)
[PASS] test_ClaimCooldowns_RevertIfCooldownNotReady() (gas: 175483)
[PASS] test_ClaimCooldowns_RevertIfDuplicateIndices() (gas: 297734)
[PASS] test_ClaimCooldowns_RevertIfIndicesNotDescending() (gas: 297733)
[PASS] test_ClaimCooldowns_RevertIfInvalidIndex() (gas: 245391)
[PASS] test_ClaimInterestAfterFullUnstake() (gas: 207171)
[PASS] test_ClaimInterest_RevertIfClaimingMoreThanAvailable() (gas: 149304)
[PASS] test_ClaimInterest_WithNoStake() (gas: 86389)
[PASS] test_ClaimMultipleCooldownsWithDifferentEndTimes() (gas: 282359)
[PASS] test_ClaimPartialInterestTwice() (gas: 185212)
[PASS] test_ClaimSpecificCooldownsByIndices() (gas: 309622)
[PASS] test_ContractNotConfigured() (gas: 78785)
[PASS] test_DepositInterestReserve() (gas: 145078)
[PASS] test_GetStakeInfo() (gas: 126777)
[PASS] test_Initialize() (gas: 52101)
[PASS] test_InterestAccumulationWithAPRChange() (gas: 136917)
[PASS] test_MultipleCooldownsWithDifferentEndTimes() (gas: 335309)
[PASS] test_MultiplePartialUnstakes() (gas: 279612)
[PASS] test_OnlyOwnerFunctions(address) (runs: 256, : 35681, ~: 35681)
[PASS] test_SetBoostedAPR() (gas: 32717)
[PASS] test_SetCooldownPeriod() (gas: 32694)
[PASS] test_SetNFTContract() (gas: 885450)
[PASS] test_SetSignToken() (gas: 453996)
[PASS] test_SetStandardAPR() (gas: 32781)
[PASS] test_StakeNFT_RevertIfAlreadyStaked() (gas: 178459)
[PASS] test_UnstakeNFT() (gas: 144181)
[PASS] test_UnstakeNFT_RevertIfNoNFTStaked() (gas: 37476)
[PASS] test_Unstake_RevertIfInsufficientBalance() (gas: 125420)
[PASS] test_Unstake_StakeMoreDuringCooldown_ClaimOnlyOriginalAmount() (gas: 206612)
[PASS] test_Unstake_WithNFT() (gas: 256675)
[PASS] test_WithdrawInterestReserve() (gas: 59098)
Suite result: ok. 37 passed; 0 failed; 0 skipped; finished in 22.99ms (115.41ms CPU time)

Ran 1 test suite in 138.51ms (22.99ms CPU time): 37 tests passed, 0 failed, 0 skipped (37 total tests)
```

9.3 Notes about Test suite

The Sign team provided a comprehensive and well-structured test suite for the `SIGNStaking` contract. It includes a significant number of fuzz tests that simulate randomized inputs to validate behavior under a wide range of scenarios.

The suite effectively covers core functionalities such as interest calculation (standard and boosted APY), repeated staking, interest claiming, and NFT interactions. It also verifies correct handling of failure conditions, including double NFT staking, premature cooldown claims, and insufficient balances. Access control and configuration functions reserved for the contract owner are likewise thoroughly tested.

CODESPECT considers the test suite to be in line with industry standards and a valuable component of the protocol's overall security posture. However, the Sign team could further improve coverage by adding tests for pausability and upgradability features, which are critical given the centralized control these mechanisms provide.