// Security Assessment

05.08.2026 - 05.12.2026

# Smart Contract Assessment

## *Blue Lighthouse*

HALBORN

# Smart Contract Assessment - Blue Lighthouse

Prepared by: HALBORN

Last Updated 06/25/2026

Date of Engagement: May 8th, 2026 - May 12th, 2026

## Summary

**100%** OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

| ALL FINDINGS | CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|---|
| 13 | 0 | 0 | 0 | 7 | 6 |

## TABLE OF CONTENTS


1. Summary
2. Assessment summary
3. Test approach and methodology
4. Risk methodology
5. Scope
6. Assessment summary & findings overview
7. Findings & Tech Details
    - 7.1 Ownership transfers may become irreversible due to missing two-step validation
    - 7.2 Invalid genesistime configuration may permanently prevent schedule initialization
    - 7.3 Working distribution addresses can be misconfigured as the controller contract itself
    - 7.4 Pool5 activation emits cycle rollover event despite no actual cycle transition
    - 7.5 Duplicated pool execution logic increases contract size and execution overhead
    - 7.6 Duplicated cycle initialization logic increases contract size and code complexity
    - 7.7 Duplicated token distribution logic increases contract size and behavioral inconsistency risk
    - 7.8 Deployment on non-cancun chains may break due to usage of reentrancyguardtransient
    - 7.9 Redundant destination updates may cause unnecessary state changes
    - 7.10 Repeated storage reads in poolsnapshot may introduce unnecessary gas overhead
    - 7.11 Direct erc20 transfers may cause compatibility issues with non-standard tokens

7.12 Redundant struct initialization may introduce unnecessary deployment gas overhead
7.13 Repeated storage reads of genesistime may introduce unnecessary gas overhead

# 1. Summary

`Blue Lighthouse` engaged `Halborn` to conduct a security assessment of the SIMN Token protocol smart contracts beginning on May 8th, 2026 and ending on May 12th, 2026. The security assessment was scoped to the smart contracts provided to the Halborn team. Commit hash and additional scope details can be found in the Scope section of this report.

The reviewed contracts implement a long-term token emission and distribution system for the SilMina (SIMN) ecosystem. The protocol consists of a fixed-supply ERC-20 token contract `SIMN` and a `ScheduleController` contract responsible for managing weekly token emissions across multiple allocation pools over a 99-year protocol schedule. The system supports scheduled burns, liquidity distributions, contributor allocations, and delayed future liquidity emissions using a diminishing quota-based release model.

## 2. ASSESSMENT SUMMARY

`Halborn` was provided with 3 days for this engagement and assigned a full-time security engineer to assess the security of the smart contracts in scope. The assigned engineer possesses deep expertise in blockchain and smart contract security, including hands-on experience auditing tokenomics, emission controllers, and treasury distribution systems.

The objective of this assessment was to:

- Identify potential security issues within the `SIMN Token protocol` smart contracts.
- Verify that the protocol's token distribution, settlement, and emission scheduling logic operates as intended.

In summary, `Halborn` identified several areas for improvement to reduce the likelihood and impact of security risks, which were acknowledged by the `Blue Lighthouse` team. The main recommendations were:

- `Introduce safer ownership transfer patterns using Ownable2Step to reduce operational risk during administrative handovers.`
- `Validate destination addresses during initialization to prevent accidental self-referential token distributions.`
- `Consolidate duplicated execution and settlement logic into reusable internal helper functions to reduce bytecode size and improve behavioral consistency.`
- `Validate deployment assumptions and initialization parameters to reduce the risk of incorrect protocol configuration during future upgrades or redeployments.`

## 3. TEST APPROACH AND METHODOLOGY

Halborn performed a combination of manual review and automated security testing to balance efficiency, accuracy, and coverage throughout the engagement. Manual review was primarily used to analyze protocol logic, token emission scheduling, access control boundaries, accounting behavior, and lifecycle state transitions. Automated tooling was used to supplement coverage and identify potential best-practice deviations.

The following phases and associated tools were used throughout the assessment:

- Research into the architecture and emission design of the `SIMN Token protocol`.
- Manual code review and walkthrough of all in-scope contracts and scheduling logic.
- Manual assessment of privileged operations, upgradeability assumptions, and token accounting behavior.
- Validation of weekly execution flows, cycle rollover behavior, and final settlement mechanics.
- Static analysis of scoped contracts and imported dependencies using Slither.
- Local deployment and execution testing using Foundry and 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)<br>Specific (AO:S) | 1<br>0.2 |

| EXPLOITABILITY METRIC ($M_E$) | METRIC VALUE | NUMERICAL VALUE |
|---|---|---|
| Attack Cost (AC) | Low (AC:L)<br>Medium (AC:M)<br>High (AC:H) | 1<br>0.67<br>0.33 |
| Attack Complexity (AX) | Low (AX:L)<br>Medium (AX:M)<br>High (AX:H) | 1<br>0.67<br>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 (C:N)<br>Low (C:L)<br>Medium (C:M)<br>High (C:H)<br>Critical (C:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Integrity (I) | None (I:N)<br>Low (I:L)<br>Medium (I:M)<br>High (I:H)<br>Critical (I:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Availability (A) | None (A:N)<br>Low (A:L)<br>Medium (A:M)<br>High (A:H)<br>Critical (A:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Deposit (D) | None (D:N)<br>Low (D:L)<br>Medium (D:M)<br>High (D:H)<br>Critical (D:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Yield (Y) | None (Y:N)<br>Low (Y:L)<br>Medium (Y:M)<br>High (Y:H)<br>Critical (Y:C) | 0<br>0.25<br>0.5<br>0.75<br>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)<br>Partial (R:P)<br>Full (R:F) | 1<br>0.5<br>0.25 |
| Scope ($s$) | Changed (S:C)<br>Unchanged (S:U) | 1.25<br>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 |

| SEVERITY | SCORE VALUE RANGE |
| --- | --- |
| Informational | 0 - 1.9 |

## 5. SCOPE

FILE

(a) Submitted File: SilMina-Audit-2026-04-main.zip

(b) Items in scope:

- /SilMina-Audit-2026-04-main/src/ScheduleController.sol
- /SilMina-Audit-2026-04-main/src/SIMN.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 | 7 | 6 |

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| OWNERSHIP TRANSFERS MAY BECOME IRREVERSIBLE DUE TO MISSING TWO-STEP VALIDATION | LOW | RISK ACCEPTED - 06/23/2026 |
| INVALID GENESISTIME CONFIGURATION MAY PERMANENTLY PREVENT SCHEDULE INITIALIZATION | LOW | RISK ACCEPTED - 06/23/2026 |
| WORKING DISTRIBUTION ADDRESSES CAN BE MISCONFIGURED AS THE CONTROLLER CONTRACT ITSELF | LOW | RISK ACCEPTED - 06/23/2026 |

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| POOL5 ACTIVATION EMITS CYCLE ROLLOVER EVENT DESPITE NO ACTUAL CYCLE TRANSITION | LOW | RISK ACCEPTED - 06/23/2026 |
| DUPLICATED POOL EXECUTION LOGIC INCREASES CONTRACT SIZE AND EXECUTION OVERHEAD | LOW | RISK ACCEPTED - 06/23/2026 |
| DUPLICATED CYCLE INITIALIZATION LOGIC INCREASES CONTRACT SIZE AND CODE COMPLEXITY | LOW | RISK ACCEPTED - 06/23/2026 |
| DUPLICATED TOKEN DISTRIBUTION LOGIC INCREASES CONTRACT SIZE AND BEHAVIORAL INCONSISTENCY RISK | LOW | RISK ACCEPTED - 06/23/2026 |
| DEPLOYMENT ON NON-CANCUN CHAINS MAY BREAK DUE TO USAGE OF REENTRANCYGUARDTRANSIENT | INFORMATIONAL | ACKNOWLEDGED - 06/23/2026 |
| REDUNDANT DESTINATION UPDATES MAY CAUSE UNNECESSARY STATE CHANGES | INFORMATIONAL | ACKNOWLEDGED - 06/23/2026 |
| REPEATED STORAGE READS IN POOLSNAPSHOT MAY INTRODUCE UNNECESSARY GAS OVERHEAD | INFORMATIONAL | ACKNOWLEDGED - 06/23/2026 |
| DIRECT ERC20 TRANSFERS MAY CAUSE COMPATIBILITY ISSUES WITH NON-STANDARD TOKENS | INFORMATIONAL | ACKNOWLEDGED - 06/23/2026 |
| REDUNDANT STRUCT INITIALIZATION MAY INTRODUCE UNNECESSARY DEPLOYMENT GAS OVERHEAD | INFORMATIONAL | ACKNOWLEDGED - 06/23/2026 |
| REPEATED STORAGE READS OF GENESISTIME MAY INTRODUCE UNNECESSARY GAS OVERHEAD | INFORMATIONAL | ACKNOWLEDGED - 06/23/2026 |

# 7. FINDINGS & TECH DETAILS

## 7.1 OWNERSHIP TRANSFERS MAY BECOME IRREVERSIBLE DUE TO MISSING TWO-STEP VALIDATION

// LOW

### Description

The `SIMN` contract `constructor` initializes ownership using OpenZeppelin's `Ownable` implementation. The contract therefore relies on a single-step ownership transfer mechanism, where ownership is immediately transferred to the new address without requiring confirmation from the recipient.

If ownership is transferred to an incorrect, non-functional, or inaccessible address, administrative privileges may become permanently unrecoverable. This introduces operational risk, particularly because the owner is responsible for privileged actions such as configuring the `scheduleController` and executing the initial `genesisMint`.

Using `Ownable2Step` would provide a safer ownership transfer flow by requiring the pending owner to explicitly accept ownership before the transfer is finalized.

### BVSS

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

### Recommendation

Consider using OpenZeppelin's `Ownable2Step` implementation to reduce the risk of accidental or irreversible ownership transfers.

### Remediation Comment

**RISK ACCEPTED:** The `Blue Lighthouse` team made a business decision to accept the risk of this finding and not alter the contracts, stating that:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.2 INVALID GENESISTIME CONFIGURATION MAY PERMANENTLY PREVENT SCHEDULE INITIALIZATION

// LOW

### Description

In the `ScheduleController` contract the `initializeController` function does not validate that the provided `_genesisTime` remains within the allowed initialization window relative to the current timestamp.

As a result, if `_genesisTime` is configured too far in the past such that: `block.timestamp > genesisTime + INITIALIZATION_WINDOW` then `startSchedule()` will always revert with `InitializationWindowExpired`, permanently preventing the schedule from being started.

While the current deployment configuration appears to initialize `genesisTime` correctly, this validation becomes important for future upgrades, redeployments, migrations, or modified initialization flows where an incorrect `genesisTime` may accidentally be supplied.

### BVSS

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

### Recommendation

Consider validating during initialization that the provided `_genesisTime` does not already exceed the allowed initialization window relative to the current block timestamp.

### Remediation Comment

**RISK ACCEPTED:** The `Blue Lighthouse` team made a business decision to accept the risk of this finding and not alter the contracts, stating that:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.3 WORKING DISTRIBUTION ADDRESSES CAN BE MISCONFIGURED AS THE CONTROLLER CONTRACT ITSELF

// LOW

### Description

The `ScheduleController.initializeController` function allows `_liqDestination` and `_contributorDest` to be configured as the `ScheduleController` contract itself. Since both `_dispatch()` and `_finalSettle()` distribute tokens using `token.transfer(destination, amount)`, assigning the controller as the destination results in self-transfers where tokens never actually leave the treasury contract.

Although accounting variables such as `totalReleasedToLiquidity` and `totalTransferredToContributor` are updated as if distributions occurred, the underlying token balances remain within the controller contract, potentially breaking emission accounting assumptions and preventing intended distributions.

While the current deployment configuration appears to use valid external destination addresses, adding explicit validation would help prevent future misconfiguration risks during upgrades or redeployments.

### BVSS

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

### Recommendation

Consider validating liquidity and contributor destination addresses during initialization and destination updates to ensure they are not set to `address(this)`.

### Remediation Comment

**RISK ACCEPTED:** The `Blue Lighthouse` team made a business decision to accept the risk of this finding and not alter the contracts, stating that:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.4 POOL5 ACTIVATION EMITS CYCLE ROLLOVER EVENT DESPITE NO ACTUAL CYCLE TRANSITION

// LOW

### Description

The `ScheduleController._activatePool5` function emits the `CycleRolledOver` event during the initial activation of the future liquidity pool: emit `cycleRolledOver(POOL_FUTURE_LIQ, pool.cycleIndex, bal, quota, weekly);`.

However, `_activatePool5` does not actually perform a cycle rollover. Instead, it initializes Pool 5 for the first time after the activation lock period and configures its initial quota and weekly distribution values.

Using the same `CycleRolledOver` event for both initial pool activation and actual yearly cycle transitions may create ambiguity for off-chain indexers, monitoring systems, and analytics integrations that rely on event semantics to distinguish lifecycle actions.

### BVSS

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

### Recommendation

Consider introducing a initial activation events from regular cycle rollover events to improve event clarity and off-chain observability.

### Remediation Comment

**RISK ACCEPTED:** The `Blue Lighthouse` team made a business decision to accept the risk of this finding and not alter the contracts, stating that:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.5 DUPLICATED POOL EXECUTION LOGIC INCREASES CONTRACT SIZE AND EXECUTION OVERHEAD

// LOW

### Description

The `ScheduleController` contract implements separate execution flows in `executeWeek` and `_tryExecutePool`, both containing largely duplicated pool execution logic.

Maintaining duplicated execution paths increases overall contract size, deployment bytecode, and execution overhead while also introducing unnecessary code complexity. Since both functions perform nearly identical operations with only minor behavioral differences, the current implementation results in redundant logic being stored and executed multiple times.

Refactoring the shared logic into a unified internal execution handler would reduce code duplication and improve overall contract efficiency.

### BVSS

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

### Recommendation

Consider consolidating the shared pool execution logic into a single internal function that supports both reverting and non-reverting execution modes through configurable behavior flags.

Copy Code

```
function executeWeek(uint8 poolId) external nonReentrant {
        if (!scheduleStarted) revert ScheduleNotStarted();
        _validatePoolId(poolId);
        _executePoolCore(poolId, true);
    }

/// @notice Attempts weekly execution for every pool; skips pools that are not executable.
function executeAvailable() external nonReentrant {
    if (!scheduleStarted) revert ScheduleNotStarted();
    _executePoolCore(POOL_BURN, false);
    _executePoolCore(POOL_LIQUIDITY, false);
    _executePoolCore(POOL_CONTRIBUTOR, false);
    _executePoolCore(POOL_FUTURE_LIQ, false);
}
function _executePoolCore(uint8 poolId, bool revertOnSkip) internal {
    Pool storage pool = pools[poolId];

    // ── Gate: settled ──
    if (pool.settled) {
        if (revertOnSkip) revert PoolAlreadySettled(poolId);
        return;
    }
    // ── Gate: Pool 5 lock ──
    if (pool.poolType == PoolType.FUTURE_LIQUIDITY) {
        if (block.timestamp < pool.activationTime) {
            if (revertOnSkip) revert Pool5Locked(pool.activationTime, uint64(block.timestamp));
            return;
        }
```

```
        // First-time activation
        if (pool.cycleIndex == 0 && pool.weeksExecutedInCycle == 0 && pool.annualQuota == 0) {
            _activatePool5();
        }
    }
```

### Remediation Comment

**RISK ACCEPTED:** The `Blue Lighthouse` team made a business decision to accept the risk of this finding and not alter the contracts, stating that:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.6 DUPLICATED CYCLE INITIALIZATION LOGIC INCREASES CONTRACT SIZE AND CODE COMPLEXITY

// LOW

### Description

The `ScheduleController` contract maintains separate implementations for `_startNewCycle` and `_activatePool5`, despite both functions performing nearly identical cycle initialization logic.

Maintaining duplicated logic increases overall contract bytecode size, introduces unnecessary code complexity, and makes future auditing or upgrades more difficult. Any future modification to the cycle initialization flow may also require updates across multiple execution paths, increasing the likelihood of inconsistent behavior.

Refactoring the shared logic into a unified internal cycle initialization function would reduce duplicate code and improve overall contract structure.

### BVSS

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

### Recommendation

Consider consolidating the shared cycle initialization logic into a single reusable internal helper function to reduce bytecode size, simplify execution flow, and improve maintainability.

Copy Code

```
function _initCycle(uint8 poolId, uint64 startTime) internal {
    Pool storage pool = pools[poolId];

    uint256 bal    = pool.balance;
    uint256 quota  = bal / QUOTA_DIVISOR;
    uint256 weekly = quota / WEEKS_PER_YEAR;

    pool.annualQuota         = uint128(quota);
    pool.weeklyAmount        = uint128(weekly);
    pool.weeksExecutedInCycle = 0;
    pool.nextExecutionTime   = startTime + uint64(WEEK_DURATION);

    emit CycleRolledOver(poolId, pool.cycleIndex, bal, quota, weekly);
}
// future pool, replacement of the _activatePool5()
if (pool.cycleIndex == 0 && pool.weeksExecutedInCycle == 0 && pool.annualQuota == 0) {
  _initCycle(POOL_FUTURE_LIQ, pool.activationTime);
}

//  Cycle rollover, replacement of the  _startNewCycle()
if (pool.weeksExecutedInCycle >= WEEKS_PER_YEAR) {
    pool.cycleIndex += 1;
    _initCycle(poolId, pool.nextExecutionTime - uint64(WEEK_DURATION));
}
```

### Remediation Comment

**RISK ACCEPTED:** The `Blue Lighthouse` team made a business decision to accept the risk of this finding and not alter the contracts, stating that:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.7 DUPLICATED TOKEN DISTRIBUTION LOGIC INCREASES CONTRACT SIZE AND BEHAVIORAL INCONSISTENCY RISK

// LOW

### Description

The `ScheduleController` contract maintains separate token distribution implementations across `_dispatch` and `_finalSettle`, despite both functions performing nearly identical transfer and accounting operations.

Maintaining duplicated distribution logic increases overall contract bytecode size, introduces unnecessary code duplication, and increases the likelihood of inconsistent behavior during future upgrades or modifications. Any future update to transfer handling, accounting logic, or token interaction patterns would need to be applied across multiple execution paths, increasing maintenance and auditing complexity.

Refactoring the shared transfer behavior into a unified internal distribution helper would improve consistency and simplify the overall execution flow.

### BVSS

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

### Recommendation

Consider consolidating the shared distribution logic into a single reusable internal helper function that handles all transfer and accounting behavior for both weekly execution and final settlement flows.

Copy Code

```
function _finalSettle(uint8 poolId) internal {
    Pool storage pool = pools[poolId];
    if (pool.settled) return;

    uint256 dust = pool.balance;
    pool.balance = 0;
    pool.settled = true;

    ActionType dustAction = ActionType.NONE;
    if (dust > 0) {
        (dustAction, ) = _send(poolId, pool.poolType, dust);
    }

    emit FinalSettlement(poolId, dust, dustAction);
}

function _send(uint8 poolId, PoolType ptype, uint256 amount) internal returns (ActionType action, address dest) {
    if (ptype == PoolType.BURN) {
        totalBurned += amount;
        token.controllerBurn(amount);
        return (ActionType.BURN, address(0));
    }

    dest = pools[poolId].destination;

    if (ptype == PoolType.LIQUIDITY || ptype == PoolType.FUTURE_LIQUIDITY) {
```

```
        totalReleasedToLiquidity += amount;
    } else {
        // PoolType.CONTRIBUTOR
        totalTransferredToContributor += amount;
    }

    bool ok = token.transfer(dest, amount);
    if (!ok) revert TransferFailed();

    action = (ptype == PoolType.CONTRIBUTOR) ? ActionType.CONTRIBUTOR : ActionType.LIQUIDITY;
}

// Call dispatch(_send()) directly this way instead of creating an internal function in executeWeek and _finalSett
(ActionType action, address dest) = _send(poolId, pool.poolType, amount);
```

### Remediation Comment

**RISK ACCEPTED:** The `Blue Lighthouse` team made a business decision to accept the risk of this finding and not alter the contracts, stating that:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.8 DEPLOYMENT ON NON-CANCUN CHAINS MAY BREAK DUE TO USAGE OF REENTRANCYGUARDTRANSIENT

// INFORMATIONAL

### Description

The `ScheduleController` uses OpenZeppelin's `ReentrancyGuardTransient`, which depends on transient storage introduced in EIP-1153. Since transient storage is only supported on Cancun-compatible EVM chains, deploying the contract on chains that have not implemented the Cancun upgrade may cause transactions to revert or the contract to become non-functional due to unsupported opcodes.

This creates a deployment and operational risk for environments where EIP-1153 support is unavailable.

### BVSS

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

### Recommendation

Consider validating Cancun / EIP-1153 support on target deployment chains before deployment, or use the standard `ReentrancyGuard` implementation to maintain compatibility across non-Cancun EVM chains.

### Remediation Comment

**ACKNOWLEDGED:** The `Blue Lighthouse` team made a business decision to acknowledge this finding and not alter the contracts, stating:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.9 REDUNDANT DESTINATION UPDATES MAY CAUSE UNNECESSARY STATE CHANGES

// INFORMATIONAL

### Description

The `ScheduleController.setDestination` function does not validate whether the provided `newDest` is different from the currently configured destination address.

As a result, administrators can repeatedly set the same destination address, causing unnecessary state writes and redundant `DestinationUpdated` event emissions without introducing any actual configuration change.

### BVSS

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

### Recommendation

Consider validating that the new destination address differs from the currently configured destination before updating storage and emitting events.

### Remediation Comment

**ACKNOWLEDGED:** The `Blue Lighthouse` team made a business decision to acknowledge this finding and not alter the contracts, stating:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.10 REPEATED STORAGE READS IN POOLSNAPSHOT MAY INTRODUCE UNNECESSARY GAS OVERHEAD

// INFORMATIONAL

### Description

The `ScheduleController.poolSnapshot` function reads multiple fields from the `Pool` struct directly through a storage reference. Since the function returns nearly the entire struct, repeatedly accessing storage fields may introduce unnecessary storage reads during execution.

Using a memory copy of the struct could improve readability and potentially reduce repeated storage access overhead when returning full snapshot data.

### BVSS

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

### Recommendation

Consider loading the `Pool` struct into memory before returning its fields when accessing most.

### Remediation Comment

**ACKNOWLEDGED:** The `Blue Lighthouse` team made a business decision to acknowledge this finding and not alter the contracts, stating:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.11 DIRECT ERC20 TRANSFERS MAY CAUSE COMPATIBILITY ISSUES WITH NON-STANDARD TOKENS

// INFORMATIONAL

### Description

The `ScheduleController` contract performs token transfers using direct ERC20 `transfer` calls instead of OpenZeppelin's `SafeERC20.safeTransfer`.

Some non-standard ERC20 implementations do not correctly return boolean values or may exhibit inconsistent transfer behavior, which can lead to unexpected transaction failures or compatibility issues when interacting with such tokens.

Although the current deployment appears to use the `SIMN` token implementation, adopting `SafeERC20.safeTransfer` would provide safer and more defensive token interaction patterns, particularly for future upgrades or token integrations.

### BVSS

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

### Recommendation

Consider using OpenZeppelin's `SafeERC20` library and replacing direct `transfer` calls with `safeTransfer` to improve compatibility with non-standard ERC20 implementations.

### Remediation Comment

**ACKNOWLEDGED:** The `Blue Lighthouse` team made a business decision to acknowledge this finding and not alter the contracts, stating:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.12 REDUNDANT STRUCT INITIALIZATION MAY INTRODUCE UNNECESSARY DEPLOYMENT GAS OVERHEAD

// INFORMATIONAL

### Description

The `ScheduleController` contract initializes pool configurations using full struct assignments, explicitly setting multiple fields to their default zero values during initialization.

Since Solidity storage variables are already zero-initialized by default, explicitly assigning zero values to all struct fields may introduce unnecessary storage operations and slightly increase deployment gas costs.

Using selective field assignments for only non-default values would provide a more gas-efficient initialization pattern while maintaining the same behavior.

### BVSS

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

### Recommendation

Consider initializing only the required non-default struct fields instead of performing full struct assignments with redundant zero-value initialization.

Copy Code

```
// Pool 2 (BURN) — destination stays address(0), default struct values are fine
pools[POOL_BURN].poolType = PoolType.BURN;

// Pool 3 (LIQUIDITY)
pools[POOL_LIQUIDITY].poolType    = PoolType.LIQUIDITY;
pools[POOL_LIQUIDITY].destination = _liqDestination;

// Pool 4 (CONTRIBUTOR)
pools[POOL_CONTRIBUTOR].poolType    = PoolType.CONTRIBUTOR;
pools[POOL_CONTRIBUTOR].destination = _contributorDest;

// Pool 5 (FUTURE_LIQUIDITY) — 7-year lock
uint64 pool5Activation = _genesisTime + uint64(POOL5_LOCK_YEARS * PROTOCOL_YEAR);
Pool storage p5 = pools[POOL_FUTURE_LIQ];
p5.poolType          = PoolType.FUTURE_LIQUIDITY;
p5.destination       = _futureLiqDest;
p5.nextExecutionTime = pool5Activation;
p5.activationTime    = pool5Activation;
```

### Remediation Comment

**ACKNOWLEDGED:** The `Blue Lighthouse` team made a business decision to acknowledge this finding and not alter the contracts, stating:

Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

## 7.13 REPEATED STORAGE READS OF GENESISTIME MAY INTRODUCE UNNECESSARY GAS OVERHEAD

// INFORMATIONAL

### Description

The `ScheduleController.startSchedule` function accesses the `genesisTime` storage variable multiple times throughout execution instead of caching the value locally.

Repeated storage reads may introduce unnecessary gas overhead due to multiple `SLOAD` operations, particularly when the same value is reused several times within the function.
Caching frequently accessed storage variables in a local variable improves readability and provides a more gas-efficient execution pattern.

### BVSS

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

### Recommendation

Consider caching `genesisTime` in a local variable at the beginning of the function and reusing the cached value throughout execution to reduce repeated storage reads.

Copy Code

```
function startSchedule() external onlyRole(DEFAULT_ADMIN_ROLE) {
        if (scheduleStarted) revert ScheduleAlreadyStarted();

        uint64 gt = genesisTime; // cache
        if (block.timestamp < gt) {
            revert BeforeGenesisTime(gt, uint64(block.timestamp));
        }
        if (block.timestamp > gt + INITIALIZATION_WINDOW) {
            revert InitializationWindowExpired(uint64(gt + INITIALIZATION_WINDOW), uint64(block.timestamp));
        }

        scheduleStarted = true;

        uint256 contractBalance = token.balanceOf(address(this));
        require(
            contractBalance == ALLOC_POOL2 + ALLOC_POOL3 + ALLOC_POOL4 + ALLOC_POOL5,
            "SC: unexpected balance at genesis"
        );

        pools[POOL_BURN].balance        = uint128(ALLOC_POOL2);
        pools[POOL_LIQUIDITY].balance   = uint128(ALLOC_POOL3);
        pools[POOL_CONTRIBUTOR].balance = uint128(ALLOC_POOL4);
        pools[POOL_FUTURE_LIQ].balance  = uint128(ALLOC_POOL5);

        _startNewCycle(POOL_BURN, gt);
        _startNewCycle(POOL_LIQUIDITY, gt);
        _startNewCycle(POOL_CONTRIBUTOR, gt);

        emit ScheduleStarted(gt);
}
```

### Remediation Comment

**ACKNOWLEDGED:** The `Blue Lighthouse` team made a business decision to acknowledge this finding and not alter the contracts, stating:

> Both contracts are already deployed on Ethereum mainnet and the schedule is live. SIMN is non-upgradeable (ownership renounced at genesisMint), and the ScheduleController findings either target one-time setup code that has already executed (initializeController/startSchedule) or are Low/Informational hardening, gas, or refactor items affecting admin-gated or non-fund-critical paths. None pose a risk to user funds or to the correctness of the running schedule. We accept the residual risk rather than upgrade a live, fund-custodying contract, and will incorporate the relevant suggestions into any future deployment or major version.

---

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.