SALUS SECURITY

SEP 2025

CODE SECURITY ASSESSMENT

RAINLINK

Overview

Project Summary

- Name: Rainlink - contract
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/hello-rainlink/contract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Rainlink - contract
Version	v8
Type	Solidity
Dates	Sep 04 2025
Logs	Apr 07 2025; Apr 09 2025; Apr 10 2025; Apr 14 2025; Apr 15 2025; Apr 17 2025; Apr 18 2025; Sep 04 2025

Vulnerability Summary

Total High-Severity issues	4
Total Medium-Severity issues	6
Total Low-Severity issues	5
Total informational issues	2
Total	17

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction **4**
1.1 About SALUS 4
1.2 Audit Breakdown 4
1.3 Disclaimer 4
Findings **5**
2.1 Summary of Findings 5
2.2 Notable Findings 6
1. BaseComn does not follow the ERC1967 standard 6
2. Calculation error leads to infinite withdraw 8
3. Dos risk via front-running consume_bridge_msg function 10
4. WithdrawBonusFromPool send incorrect value of bonus 12
5. Unprotected Fees may lead to significant losses for users 14
6. Logic error in onlyAdmin modifier 15
7. Pool can't send wrapped-tokens’s bonus 16
8. Bridge fee will lock in Messager contract 17
9. Users can call bridgetoken with only 1wei-upload_gas_fee 18
10. Untracked residual fee in pool fee distribution 19
11. Lack of removal mechanism for staking pools 20
12. Missing events for functions that change critical state 21
13. Unchecked token address in sendTokenFee function 22
14. Lack of lock period on staked tokens allows potential bonus exploitation 23
15. Potential withdrawal failure 24
2.3 Informational Findings 25
16. Missing zero address checks 25
17. Use of floating pragma 26
Appendix **27**
Appendix 1 - Files in Scope 27

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	BaseComn does not follow the ERC1967 standard	High	Business Logic	Resolved
2	Calculation error leads to infinite withdraw	High	Business Logic	Resolved
3	Dos risk via front-running consume_bridge_msg function	High	Denial of Service	Resolved
4	WithdrawBonusFromPool send incorrect value of bonus	High	Business Logic	Resolved
5	Unprotected Fees may lead to significant losses for users	Medium	Business Logic	Resolved
6	Logic error in onlyAdmin modifier	Medium	Business Logic	Resolved
7	Pool can't send wrapped-tokens's bonus	Medium	Business Logic	Resolved
8	Bridge fee will lock in Messager contract	Medium	Business Logic	Resolved
9	Users can call bridgetoken with only 1wei-upload_gas_fee	Medium	Business Logic	Acknowledged
10	Untracked residual fee in pool fee distribution	Medium	Business Logic	Resolved
11	Lack of removal mechanism for staking pools	Low	Business Logic	Resolved
12	Missing events for functions that change critical state	Low	Logging	Resolved
13	Unchecked token address in sendTokenFee function	Low	Business Logic	Resolved
14	Lack of lock period on staked tokens allows potential bonus exploitation	Low	Business Logic	Acknowledged
15	Potential withdrawal failure	Low	Business Logic	Resolved
16	Missing zero address checks	Informational	Data Validation	Resolved
17	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. BaseComn does not follow the ERC1967 standard

Severity: High	Category: Business Logic

Target:
- BaseComn.sol
- proxy/MyProxy.sol

Description

The ERC1967 splits the admin address and an implementation address to two slots. According to the `BaseComn` contract codebase, the project designed a `core/Admin` contract to control the admin part. At the same time, the function of changing the admin address is missing, such as _setAdmin and changeAdmin.

The `MyProxy` contract's constructor function is used to set the implementation contract address and execute initialization data during initialization. Obviously, the admin address is not set which will cause all admin function dos.

proxy/MyProxy.sol:L13-L23

```
contract MyProxy is BaseComn, Proxy {
   ...
    constructor(address impl, bytes memory data) {
        // Call the upgradeToAndCall function from ERC1967Utils to set the
implementation and execute the provided data.
        ERC1967Utils.upgradeToAndCall(impl, data);
    }
```

Attach Scenario:
In `onlyAdmin` execute process:

1. Read `__ADMIN_SLOT`'s slot.value for calling the corresponding contract's mustMaster function.
2. The admin address is not set during initialization and not a function for changing it. So the `__ADMIN_SLOT`'s slot.value is zero address.
3. Call the zero address will revert.

BaseComn.sol::L11-L33

```
abstract contract BaseComn {
    bytes32 constant _ADMIN_SLOT =
        0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;
    modifier onlyAdmin() {
        IAdmin(getAdminAddr()).mustAdmin(msg.sender);
        _;
    }

    function getAdminAddr() public view returns (address) {
```

```
        return StorageSlot.getAddressSlot(_ADMIN_SLOT).value;
    }
}
```

Recommendation

Add the admin address set during initialization and complete the function of `BaseComn` contract.

Status

The team has resolved this issue in commit [a7c6ea1](a7c6ea1).

2. Calculation error leads to infinite withdraw

Severity: High	Category: Business Logic

Target:
- token/Pool.sol

Description

The `withdrawBonusFromPool` function for withdrawing rewards from the staking pool will check whether the pool exists, calculate the user's reward, reset the reward information, update the total amount of the pool, and transfer the reward tokens to the user.

token/Pool.soll:L156-L178

```
function withdrawBonusFromPool(address stakeToken, uint amount) public {
    ....
    if (bonus > 0) {
        emit Types.Log("wd bonus", bonus);
        // Reset the bonus information.
        _resetBonus(stakeToken, amount);

        // Decrease the total amount in the pool.
        poolMap[stakeToken].inAmount -= amount;
        // Transfer the bonus tokens to the user.
        SafeERC20.safeTransfer(IToken(stakeToken), msg.sender, bonus);
    }
}
```

According to the `comn/Types` contract, we know that `PoolInfo.acc` is based on the number of Q64.64. In the highlighted expression below, the `userStakeAmountMap[user][token].debt` and `userStakeAmountMap[user][token].remainReward` are not based on the number of Q64.64. The impact of adding and subtracting these two variables on `totalReward` is essentially insignificant due to the order of magnitude difference. This also applies to `amount_`. Therefore, the check always passes and the `remainReward` is almost unchanged.

token/Pool.soll:L492-L507

```
function _resetBonus(address token, uint amount_) private {
    address user = msg.sender;
    uint amount = userStakeAmountMap[user][token].amount;
    uint newDebt = (amount * poolMap[token].acc) >> 64;

    // Calculate the total reward the user has earned.
    uint totalReward = (amount * poolMap[token].acc) -
        userStakeAmountMap[user][token].debt +
        userStakeAmountMap[user][token].remainReward;
    require(totalReward >= amount_, "not enough reward");
    uint newReward = (totalReward - amount_) >> 64;

    // Update the user's remaining reward and debt.
    userStakeAmountMap[user][token].remainReward = newReward;
    userStakeAmountMap[user][token].debt = newDebt;
}
```

Normally, the part withdrawn by the user should be subtracted in `remainReward`. Due to the above error, the `remainReward` is unchanged. Therefore, attackers can repeatedly withdraw rewards until the pool is exhausted.

Recommendation

Override the expression for making the variables in the same base number.

Status

The team has resolved this issue in commit af1638d.

3. Dos risk via front-running consume_bridge_msg function

Severity: High	Category: Denial of Service

Target:
- token/Executor.sol
- core/Messager.sol

Description

All Ethereum-like blockchain transactions and their parameters are visible in the mempool before execution. When the relayers call the `processMsg` function with the `message` and `signature`, these parameters are publicly accessible to anyone.

token/Executor.sol:L203-L282

```
function processMsg(
    Types.Message memory message,
    bytes[] memory signature // 65bytes for one signature
) public returns (bool) {
    ...
    bool consume_success = IMessager(MessagerAddr).consume_bridge_msg(message,
signature);
    if (!consume_success && msg.sender !=
address(0x0000000000000000000000000000000000000001)) {
        revert("nonce has been uploaded");
    }
    ...
}
```

In the current design, `consume_bridge_msg` can be called by anyone, an attacker can front-run the original transaction by submitting the same parameters in a higher-fee transaction. This would consume the nonce first, marking it as used and invalidating the legitimate transaction.

core/Messager.sol:L175-L188

```
function consume_bridge_msg(Types.Message memory messageDec, bytes[] memory signature)
public returns (bool) {
    (bool rs,) = _verify_msg(messageDec, signature);

    uint256 from_chain_id = ComFunUtil.combainChain(messageDec.msg_header.from_chain);
    uint256 nonce = messageDec.msg_header.nonce;

    // not empty key
    if (nonceStateMap[from_chain_id][nonce]) {
        emit Types.Log("nonce has been used");
        return false;
    } else {
        nonceStateMap[from_chain_id][nonce] = true;
    }

    return rs;
}
```

Users attempting to finalize their cross-chain messages can be blocked by malicious actors who preemptively consume the message, effectively forcing users to re-initiate the process or the validator to re-sign the message (DoS).

Recommendation

Implement an access control in the `consume_bridge_msg` function(e.g., onlyExecutor).

Status

The team has resolved this issue in commit 63ac084.

4. WithdrawBonusFromPool send incorrect value of bonus

Severity: High	Category: Business Logic

Target:
- token/Pool.sol

Description

The `withdrawBonusFromPool` function transfers the user's entire bonus amount instead of the partial sum indicated by the `amount` parameter. If the user has a bonus balance of 3, but calls `withdrawBonusFromPool(stakeToken, 1)`, the contract still transfers the full 3 tokens.

token/Pool.sol:L147-L169

```
function withdrawBonusFromPool(address stakeToken, uint256 amount) public {
    // Check if the pool for the given token exists.
    if (poolMap[stakeToken].token == address(0)) {
        revert("pool not found");
    }

    require(amount > 0, "amount is zero");
    address user = msg.sender;
    // Calculate the bonus amount.
    uint256 bonus = calBonusFromPool(user, stakeToken);

    // If there is a bonus, proceed with the withdrawal.
    if (bonus > 0) {
        emit Types.Log("wd bonus", bonus);
        // Reset the bonus information.
        _resetBonus(stakeToken, amount);

        // Decrease the total amount in the pool.
        poolMap[stakeToken].inAmount -= amount;
        // Transfer the bonus tokens to the user.
        SafeERC20.safeTransfer(IToken(stakeToken), msg.sender, bonus);
    }
}
```

Although the `_resetBonus` function attempts to reduce the user's recorded bonus or debt by `amount_`, the actual on-chain transfer (`SafeERC20.safeTransfer(...)`) sends the entire bonus instead of the partial amount.

token/Pool.sol:L435-L449

```
function _resetBonus(address token, uint256 amount_) private {
    address user = msg.sender;
    uint256 amount = userStakeAmountMap[user][token].amount;
    uint256 newDebt = (amount * poolMap[token].acc) >> 64;

    // Calculate the total reward the user has earned.
    uint256 totalReward = (amount * poolMap[token].acc) -
userStakeAmountMap[user][token].debt
        + userStakeAmountMap[user][token].remainReward;
    require(totalReward >= amount_, "not enough reward");
    uint256 newReward = (totalReward - amount_) >> 64;

    // Update the user's remaining reward and debt.
```

```
    userStakeAmountMap[user][token].remainReward = newReward;
    userStakeAmountMap[user][token].debt = newDebt;
}
```

As a result, a user can repeatedly call `withdrawBonusFromPool` with small `amount_` values, each time receiving the full bonus, thereby extracting more tokens than their actual bonus balance.

Recommendation

Adjust the transfer logic so it only sends `amount_`.

Status

The team has resolved this issue in commit 63ac084.

5. Unprotected Fees may lead to significant losses for users

Severity: Medium	Category: Business Logic
Target: - token/Pool.sol	

Description

When the executor performs a `Types.TokenType.pool` token transfer, a `LpFee` will be charged. The minimum fee rate is 0.03%. it will be charged in proportion to the amount in the pool.

token/Pool.sol : L270-L284

```
function getLpFee(address token, uint amount) public view returns (uint) {
    uint pool_fee_all = Math.mulDiv(amount, POOL_FEE, 1000000);
    uint all_cal_amount = poolMap[token].inAmount;
    require(
        all_cal_amount > 0,
        "Total pool amount must be greater than zero"
    );

    uint ratio = Math.mulDiv(amount, 1000, all_cal_amount);
    if (ratio < 3) {
        return pool_fee_all;
    } else {
        return Math.mulDiv(amount, ratio, 1000);
    }
}
```

Attach Scenario:

1. Alice owns 90% of a pool of 100 ether in Blockchain B.
2. Bob bridge 1 ether from Blockchain A to Blockchain B. The estimated fee at this time is 1%, which is equal to 0.01 ether.
3. Before the bridge transaction is executed, Alice withdraws all ether.
4. The actual fee soared to 10%, , which is equal to 0.1 ether.
5. This situation is unacceptable to Bob.

This strategy can be used to target specific users. And it is easier to achieve when there is insufficient liquidity in the early stages of the protocol, which will undermine user confidence in the protocol.

Recommendation

Adding a slippage mechanism.

Status

The team has resolved this issue in commit 63ac084.

6. Logic error in onlyAdmin modifier

Severity: Medium	Category: Business Logic
Target: - core/Admin.sol	

Description

The current implementation requires that the caller must be the contract itself. This actually blocks any external calls because the `msg.sender` of the external caller will never be equal to the contract address. This may cause all functions that require the `onlyAdmin` modifier to not be called.

core/Admin.sol:L31-L34

```
modifier onlyAdmin() {
    require(address(this) == msg.sender, "Must admin");
    _;
}
```

Recommendation

Fix example:

```
modifier onlyAdmin() {
    require(isAdmin(msg.sender), "Must admin");
    _;
}
```

Status

The team has resolved this issue in commit 63ac084.

7. Pool can't send wrapped-tokens's bonus

Severity: Medium	Category: Business Logic
Target: - token/Pool.sol	

Description

In the current implementation, when users stake wrapped tokens into the pool, they send native currency (e.g., ETH) rather than ERC20 tokens. However, during bonus withdrawal via `withdrawBonusFromPool`, the bonus is transferred using the ERC20 `safeTransfer` function, preventing users from receiving their bonus.

token/Pool.solL147-L169

```
function withdrawBonusFromPool(address stakeToken, uint256 amount) public {
    // Check if the pool for the given token exists.
    if (poolMap[stakeToken].token == address(0)) {
        revert("pool not found");
    }

    require(amount > 0, "amount is zero");
    address user = msg.sender;
    // Calculate the bonus amount.
    uint256 bonus = calBonusFromPool(user, stakeToken);

    // If there is a bonus, proceed with the withdrawal.
    if (bonus > 0) {
        emit Types.Log("wd bonus", bonus);
        // Reset the bonus information.
        _resetBonus(stakeToken, amount);

        // Decrease the total amount in the pool.
        poolMap[stakeToken].inAmount -= amount;
        // Transfer the bonus tokens to the user.
        SafeERC20.safeTransfer(IToken(stakeToken), msg.sender, bonus);
    }
}
```

Recommendation

Add the logic in `withdrawBonusFromPool` if the staked token is a wrapped token.

Status

The team has resolved this issue in commit 63ac084.

8. Bridge fee will lock in Messager contract

Severity: Medium	Category: Business Logic
Target: - token/Executor.sol - core/Messager.sol	

Description

When user bridge token, `msger_value` will be sent to the `IMessager` contract. After the `Messager` contract review, we did not find any handle function for `msger_value` which will lead to the fee lock in the Messager contract.

token/Executor.sol:L185-L256

```
function bridgeToken(
    ...) public payable {
    ...

    IMessager(MessagerAddr).emit_msg{value: msger_value}(
        0,
        to_chain,
        receiver,
        messageBody,
        upload_gas_fee
    );
}
```

Recommendation

Add function for admin for receiving fee.

Status

The team has resolved this issue in commit 63ac084.

9. Users can call bridgetoken with only 1wei-upload_gas_fee

Severity: Medium	Category: Business Logic
Target: - token/Executor.sol	

Description

In the `bridgeToken` function, the only check regarding the `upload_gas_fee` is to ensure that `msg.value >= upload_gas_fee`. As long as `upload_gas_fee` is set to any value greater than zero (for example, 1 wei), the function will proceed with the cross-chain operation.

token/Executor.sol:L142-L201

```
function bridgeToken(
    bytes32 source_token,
    Types.Chain memory to_chain,
    bytes32 to_who,
    bytes32 receiver, // destination bridger
    uint128 all_amount,
    uint128 upload_gas_fee // convert target platform token to source platform token
) public payable {
    uint256 all_value = msg.value;
    uint256 msger_value;

    // transfer gas fee to pool
    require(all_value >= upload_gas_fee, "please send enough gas");
    msger_value = all_value - upload_gas_fee;
    IPool(PoolAddr).sendEthFee{value: upload_gas_fee}(WTOKEN_ADDRESS);
    ...
}
```

This lack of a minimum fee requirement allows users to set the fee extremely low, which may result in the system not collecting the expected fee revenue and relayers not receiving adequate incentives to process cross-chain transactions on the destination chain.

Recommendation

Implement a minimum bound for `upload_gas_fee`.

Status

This issue has been acknowledged by the team and states that it is a protocol design.

10. Untracked residual fee in pool fee distribution

Severity: Medium	Category: Business Logic
Target: - token/Pool.sol	

Description

Within the `transferFromPool` function, the LP fee (`lp_fee`) is computed by calling `getLpFee(destToken, allAmount)`. The Pool contract then deducts this `lp_fee` from the total bridged amount, sending `allAmount - lp_fee` to the recipient. A portion of the LP fee is recalculated as `pool_fee` using the formula:

token/Pool.sol:L178-L221

```
function transferFromPool(address destToken, address toWho, uint256 allAmount) public
payable onlyExecutor {
    // Calculate the LP fee.
    uint256 lp_fee = getLpFee(destToken, allAmount);
    // Calculate the actual amount to be transferred after deducting the fee.
    uint256 amount = allAmount - lp_fee;
    ...
    // Calculate the pool fee.
    uint256 pool_fee = Math.mulDiv(lp_fee, poolMap[destToken].amount,
poolMap[destToken].inAmount);
    // Get a reference to the pool information.
    Types.PoolInfo storage poolInfo = poolMap[destToken];
    // If there is a staked amount in the pool, update the reward amount and APY.
    if (poolInfo.stakeAmount > 0) {
        poolInfo.rewardAmount += pool_fee;
        if (pool_fee > 0 && poolInfo.stakeAmount > 0) {
            ...
        }
    }
}
```

This `pool_fee` is added to the pool's `rewardAmount` for distribution among stakers. However, the remaining portion of the LP fee, calculated as `lp_fee - pool_fee`, is not explicitly accounted for or stored in a dedicated variable; it remains implicitly in the pool's asset balance.

The residual fee remains locked within the pool without clear tracking or a mechanism for extraction.

Recommendation

Introduce a dedicated variable or mechanism to record the residual fee and provide a mechanism for the platform to extract or reallocate.

Status

The team has resolved this issue in commit f700565.

11. Lack of removal mechanism for staking pools

Severity: Low	Category: Business Logic
Target: - token/Pool.sol	

Description

The `stakeIntoPool` function in the Pool contract requires that a pool already exists for the token specified by the user. This is verified by checking if `poolMap[stakeToken].token` is not the zero address. However, there is no mechanism provided for the administrator to remove or disable an existing staking pool.

As a result, if a token later is found to have security issues or other concerns, the admin cannot shut down its staking pool to prevent new users from staking insecure tokens.

Recommendation

Implement an admin function that allows for the removal or deactivation of staking pools.

Status

The team has resolved this issue in commit de8e0fa.

12. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - core/Messager.sol - token/Executor.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the Messager contract, the event is lacking in the `set_bridge_fee` function.

In the Executor contract, events are lacking in the `setTokenRelationship`, `removeTokenRelationship` and `removeTokenRelationship` functions.

Recommendation

It is recommended to emit events for critical state changes.

Status

The team has resolved this issue in commit 63ac084.

13. Unchecked token address in sendTokenFee function

Severity: Low	Category: Business Logic
Target: - token/Pool.sol	

Description

The `sendTokenFee` function in Pool contract allows a user to specify a token address as an input parameter, and then calls functions on that address without first verifying that a staking pool for the specified token exists. This can lead to potential security issues if an incorrect or malicious token address is provided, as the function will proceed to transfer tokens and update pool state using that address.

```
function sendTokenFee(address token, uint256 amount) public {
    // Transfer the tokens from the user to the contract.
    SafeERC20.safeTransferFrom(IToken(token), msg.sender, address(this), amount);
    // Add the locked amount to the pool.
    _addLockAmount(token, amount);
}
```

Recommendation

Add a check to ensure that a pool for the specified token exists.

Status

The team has resolved this issue in commit 63ac084.

14. Lack of lock period on staked tokens allows potential bonus exploitation

Severity: Low	Category: Business Logic
Target: - token/Pool.sol	

Description

Currently, in Pool contract, the bonus for staking is calculated based on the user's staked amount and the pool's accumulated reward coefficient (acc). In the function `calBonusFromPool`, the bonus is computed as:

token/Pool.sol:L229-L236

```
function calBonusFromPool(address user, address stakeToken) public view returns
(uint256) {
    // Get the user's staked amount information.
    Types.UserAmountInfo memory uInfo = userStakeAmountMap[user][stakeToken];
    // Calculate the new reward.
    uint256 newReward = (uInfo.amount * poolMap[stakeToken].acc) >> 64;
    // Calculate the total bonus.
    return newReward + uInfo.remainReward - uInfo.debt;
}
```

Because there is no minimum lock period enforced after staking, a user (or attacker using their own funds) could execute a sandwich-style attack as follows:

1. **First Transaction:** The attacker stakes a large amount of tokens into the pool.
2. **Second Transaction:** An executor's transaction (triggered by a legitimate cross-chain message) increases the pool's acc, thereby boosting the bonus calculation.
3. **Third Transaction:** The attacker calls `withdrawBonusFromPool` to withdraw the bonus, and calls the `withdrawFromPool` to withdraw the staked token.

However, the potential profit is mitigated by the gas cost and the incremental nature of the acc increase, making the attack low-risk in practice.

Recommendation

Implement a minimum lock period for staked tokens that prevents immediate unstaking after a stake operation.

Status

This issue has been acknowledged by the team.

15. Potential withdrawal failure

Severity: Low	Category: Business Logic
Target: - token/Pool.sol	

Description

When tokens are transferred out of the pool, the `poolMap[destToken].amount` will be reduced by the ratio of the transferred amount to the total amount. This will cause `poolMap[destToken].amount` to be less than or equal to `poolMap[destToken].stakeAmount`. During the withdrawal process, it will be checked that the amount withdrawn by the user cannot exceed the actual amount of assets in the pool and the total stake amount.

token/Pool.sol:L186-L242,L442-L471

```
function transferFromPool() public payable onlyExecutor {
    ...
    uint stakedDecrease = (allAmount * poolMap[destToken].amount) /
        poolMap[destToken].inAmount;
    // Decrease the staked amount in the pool.
    poolMap[destToken].amount -= stakedDecrease;
    ...
}
function _removeStakeAmount(address token, uint amount) private {
    ...

    require(amount <= poolMap[token].amount, "not enough pool assets");
    require(
        amount <= poolMap[token].stakeAmount,
        "not enough pool stakeAmount assets"
    );
    ...
}
```

Attach Scenario:

1. Alice is the only staker in the pool. At this time, `poolMap[destToken].stakeAmount` is equal to `poolMap[destToken].amount`.
2. The Executor calls `transferFromPool` to perform normal operations. Each transfer will reduce the `poolMap[destToken].amount` but keep the `poolMap[destToken].stakeAmount` unchanged.
3. Alice wants to withdraw all stake tokens. The action will fail due to failure to pass this check.

Recommendation

Remove strict checks on `_removeStakeAmount`.

Status

The team has resolved this issue in commit 63ac084.

2.3 Informational Findings

16. Missing zero address checks

Severity: Informational	Category: Data Validation
Target: - core/Admin.sol	

Description

It is considered a security best practice to verify addresses against the zero address. However, this precautionary step is absent in the WETH9 contract.

core/Admin.sol:L56-L60

```
function setAdmin(address newAdmin) public onlyMaster {
    emit AdminChanged(admin, newAdmin);
    admin = newAdmin;
}
```

Recommendation

Consider adding zero address checks for above address variables.

Status

The team has resolved this issue in commit 63ac084.

17. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.20;
```

The Rainlink uses a floating compiler version ^0.8.20.

Using a floating pragma ^0.8.20 statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit c39ddd8:

File	SHA-1 hash
comn/TokenBatch.sol	ee47dc8a484ca89041442e85fc1a4498b8adcc5c
comn/ComFunUtil.sol	4f56b4b0a71c057bd2cfbb83bc9bbc566e60675d
comn/SafeERC20.sol	09ebab309ac4b9280b3801dfb4838273047e4422
comn/Types.sol	496faf913076d5f6d40cb2e71581ed49fd2296e6
core/Validator.sol	b3292a713951190aa21c203ddc918f1eee849785
core/Comn.sol	9feb0f428c3365cb9c1b01a9b84bca1e070b6733
core/Messager.sol	54407233e563fcbd1d6d6f833e28f38033ac9c5f
core/Admin.sol	246202aed8529ae7c17458d28b4066d1dc2c2a82
proxy/MyProxy.sol	5750791a87895e4d33aab7dcc138137c76cb6699
token/Token.sol	5a7c0dbe5326cda080629402343068e1deac89af
token/Pool.sol	1e87926c132cd50469159074dc6ba8b93df34274
token/Comn.sol	ed2bee7db425cf95fbc767a6f89a719f913eb684
token/Executor.sol	8cc42caca7792cd9b8d2c5744f4c798b5f521c7b
BaseComn.sol	101fc7038280b24664b74c877a50a83925842289

And we audited the commit de8e0fa that introduced new features hello-rainlink/contract repository:

File	SHA-1 hash
comn/WToken.sol	cdd6c1eb31dfa11884b58787d03c25079b0b0e6b
core/Admin.sol	a9493822b8faeccb7c19f26a132510cce95fe433
token/Pool.sol	296a661bc15b0f2718aa6ceed8498c900f19af78
token/Executor.sol	613acd09c9307ce8ad32776760d7e73e7ac9323c