# SMART CONTRACT AUDIT REPORT

for

# BRIDGE CONTRACT

**Prepared By: Keane Ho**

**Singapore**
**May 29, 2024**

## Document Properties

| | |
|---|---|
| Client | Bridge Contract |
| Title | Smart Contract Audit Report |
| Target | Bridge Contract v1 |
| Version | 1.0 |
| Author | Keane Ho |
| Auditors | Polaristow, ACaiSec, tanliwei, 7g, bsssss, Alex, Dicanoex |
| Reviewed by | F1 |
| Approved by | Seek |
| Classification | Public |

## Version Info

| Version | Date | Author(s) | Description |
|---|---|---|---|
| 0.1 | May 30, 2024 | Keane Ho | Initial Draft |
| 1.0 | June 24, 2024 | Keane Ho | Final Release |

## Contact

For more information about this document and its contents, please contact Solid-Rock-Security .



| | |
|---|---|
| Name | Keane Ho |
| Phone | +86 17208278520 |
| Email | keane@dapplink.xyz |

# 1 Introduction

Given the opportunity to review the **Bridge Contract v1** design document and related smart contract source code, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between the smart contract code and the design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of the smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

## 1.1 About Bridge Contract V1

First of all, Dapplink is a modular, combinable Layer3 AppChain protocol, created for decentralized large-scale application scenarios, with high security, low cost, and integration with AI. Dapplink supports deployment on any Layer2, including Bitcoin and Ethereum chains. The Layer3 AppChain supporting EVM will be launched on the test network in May this year. Developers can freely combine different Dapplink Layer3 modules to serve upper-layer applications according to their needs.

In order to support DappLink's Layer3 multi-staking protocol and allow assets to be staked across chains, Dapplink designed and developed its own cross-chain interoperability protocol. At present, cross-chain interoperability between Ethereum and Ethereum-Layer2 and EVM chains has been realized, and cross-transfer between Bitcoin, Bitocin-Layer2 and EVM chains will also be supported in the future. In addition, the DappLink Bridge test network has been launched, and the main network will be launched soon.The basic information of Bridge Contract V1 is as follows:

Table 1.1: Basic Information of Bridge Contract V1

| Item | Description |
|---|---|
| Issuer | Bridge Contract |
| Website | https://bridge.testnet.dapplink.xyz/bridge |
| Type | Ethereum Smart Contract |
| Platform | Solidity |
| Audit Method | Whitebox |
| Latest Audit Report | May 29, 2024 |

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/eniac-x-labs/bridge-contracts(965d18f)

And this is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/eniac-x-labs/bridge-contracts(2f5952b)

## 1.2 About Solid Rock Security

Solid Rock Security Labs focus on Web3 project attack and defense, Web3 project audit and Web3 project security analysis. We are reachable at Telegram(todo),Twitter (https://x.com/0xsolidrock), or Email (todo).

## 1.3 Disclaimer

Note that this audit does not give any warranties on finding all possible security issues of the given smart contract(s), i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Lastly, this security audit should not be used as investment advice.

# 2 | Findings

## 2.1 Summary

Here is a summary of our findings after analyzing the Bridge Contract v1 Protocol design and implementation. During the first phase of our audit, we studied the smart contract source code and ran our in-house static code analyzer through the codebase. The purpose is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool.

| Severity | # of Findings | |
|---|---|---|
| Critical | 15 | ■■■ |
| High | 11 | ■■ |
| Medium | 19 | ■■■■ |
| Low | 19 | ■■■■ |
| Total | 64 | |

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section .

## 2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 15 critical-severity vulnerabilities, 11 high-severity vulnerabilities , 19 medium-severity vulnerabilities, and 19 low-severity vulnerabilities.

| ID | Severity | Title | Status |
|---|---|---|---|
| SRS-001 | High | Handling Fee-on-Transfer Tokens in ERC20 Staking Function | fixed |
| SRS-002 | Medium | call() Should be Used Instead of transfer() on An Address Payable | fixed |
| SRS-003 | High | Handling Chain ID Changes to Prevent Withdrawal Failures | reject |
| SRS-004 | Low | Underflow Bug in Pool Index Calculation | fixed |
| SRS-005 | Medium | MessageManager.claimMessage function lacks a check to prevent replay | fixed |
| SRS-006 | Medium | The setSupportToken function parameter check is missing | reject |
| SRS-007 | Low | Redundant check for uint256 type variables | fixed |
| SRS-008 | Low | WithdrawOrClaimBySimpleAsset function parameter type optimization | fixed |
| SRS-009 | Critical | The calculation method of updating the TotalAmount variable is wrong | fixed |
| SRS-010 | Critical | Loses precision in calculating Reward, which will cause user asset loss | fixed |
| SRS-011 | Medium | The DepositAndStaking type function does not fully check the Pools status | fixed |
| SRS-012 | Low | DepositAndStaking function check is not rigorous | reject |
| SRS-013 | High | The pause function of the L2Pool Manager contract is missing | fixed |
| SRS-014 | Low | The FundingPoolBalance data structure has security risks | reject |
| SRS-015 | Low | TokenBridgeBase.setPerFee function parameter check missing | fixed |
| SRS-016 | Medium | The updates of the two key data structure s IsSupportToken and SupportTokens are inconsistent | acknown |
| SRS-017 | Low | TokenBridgeBase.setPerFee function parameter check missing | fixed |
| SRS-018 | Critical | The stakingMessageNumber usage error | fixed |
| SRS-019 | Low | TokenBridgeBase.PerFee incorrect comment | reject |
| SRS-020 | Medium | Mising invoking the initial functions of parent contracts | fixed |
| SRS-021 | Critical | Users can drain tokens from the L1Pool Manager contract | fixed |
| SRS-022 | Critical | Users will lost the WETH after the deposit | fixed |
| SRS-023 | Medium | The Withdraw event is emitted incorrectly | fixed |
| SRS-024 | Medium | Wrong argument in the LessThanMin StakeAmount event | fixed |
| SRS-025 | High | Wrongly set Users[_user][index]. isWithdrawed if the IsWithdraw is false | reject inexistence The Reward is withdrawn regardless of whether the IsWithdraw is true |
| SRS-026 | Critical | Incompatible With the Deflationary or Fee-on-Transfer Tokens | inexistence(reject) |

| | | | |
|---|---|---|---|
| SRS-027 | Low | Potential Duplicated Items in the SupportTokens | acknown(low) |
| SRS-028 | Low | The word Assert should be Asset | inexistence(reject) |
| SRS-029 | Low | Checks-Effects-Interactions Pattern is not adopted | inexistence(reject) |
| SRS-030 | Medium | Missing passing the bridge token address to the messageManager | reject |
| SRS-031 | Medium | There is no minimum amount for the bridge amount | reject |
| SRS-032 | Medium | There is a risk that users cannot receive ETH | fixed |
| SRS-033 | Low | Gas optimization: cache array length | acknown |
| SRS-034 | Medium | The method signature is used incorrectly | fixed |
| SRS-035 | Low | SupportTokens does not clean up unsupported tokens | acknown |
| SRS-036 | Low | NewPoolIsNotCreate prompt is inconsistent | fixed |
| SRS-037 | Medium | Users cannot withdraw funds or receive earnings | inexistence |
| SRS-038 | Critical | Users can withdraw ETH repeatedly indefinitely | fixed |
| SRS-039 | Low | Protocol Rely on Administrator Actions | reject |
| SRS-040 | Low | Gas Optimization:Redundant Implementaion | fixed |
| SRS-041 | Medium | Potential Failed Transfer ETH to User | fixed |
| SRS-042 | Medium | Lack of Storage Gaps in the Contract TokenBridgeBase | reject |
| SRS-043 | Medium | Incorrect Update isWithdrawed Flag | fixed |
| SRS-044 | High | Potential Dos in Reward Calculation Loop | Not fixed (That's it for now.) |
| SRS-045 | Critical | Incorrect initialization of the TotalAmount | inexistence |
| SRS-046 | Critical | User unable withdraw asset after cliam reward | fixed |
| SRS-047 | Critical | Incorrect Reward Calculation Logic-I | inexistence |
| SRS-048 | Critical | Incorrect Reward Calculation Logic-II | inexistence |
| SRS-049 | Low | Use safeApprove() instead approve() | reject |
| SRS-050 | Low | Underpaying Optimism l2gas may lead to loss of funds | Not fixed (acknown) |
| SRS-051 | High | The _l2TxGasLimit is set incorrectly | Not fixed (acknown) |
| SRS-052 | High | The gasPerPubdataByte is incorrect | Not fixed (acknown) |
| SRS-053 | Critical | If the L2 deposit finalization transaction fails, the protocol will lose funds | Not fixed (acknown) |
| SRS-054 | Critical | Refunded funds from cross-chain transactions will be lost | Not fixed (acknown) |
| SRS-055 | Medium | The IsCompleted check should be applied to staking ERC20 and ETH | fixed |
| SRS-056 | Critical | Users cannot bridge WETH from the ETH chain to the target chain | No WETH for L1 |
| SRS-057 | Medium | Users cannot bridge WETH from the ETH chain to the target chain | fixed |
| SRS-058 | Critical | Missing isWithdrawed check | fixed |
| SRS-059 | Low | Missing sourceChainId != destChainId check | fixed |

| | | | |
|---|---|---|---|
| SRS-060 | High | Missing send value for TransferAssertToMantleBridge on ETH | fixed |
| SRS-061 | Low | Missing payable for IMantleL1Bridge.depositETHTo | fixed |
| SRS-062 | High | Missing fee for TransferAssertToScrollBridge | inexistence(erc20 and weth is token,token transfer no eth fee) |
| SRS-063 | High | Missing length != 0 check for DepositAndStakingERC20/WETH | fixed |
| SRS-064 | High | Incorrect usage of safeTransfer(From) for ERC20 | reject(too older versions of token are not supported) |

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

# 3 Detailed Results

## 3.1 Handling Fee-on-Transfer Tokens in ERC20 Staking Function

- ID: SRS-001
- Auditors: Polaristow,Alex
- Impact: N/A
- Severity: High
- Likelihood: N/A

### Description

The DepositAndStakingERC20 function does not account for fee-on-transfer tokens, which can cause discrepancies between the amount transferred from the user and the amount actually received by the contract. This can lead to inaccurate accounting, as the contract assumes the full amount _amount is received and updates balances and pool amounts accordingly.

If the ERC-20 token being transferred charges a fee on transfer, the contract will not receive the full _amount. The function safeTransferFrom will transfer _amount from the user, but the contract will receive _amount - fee. However, the function incorrectly updates the user's staked amount and the pool's total amount by _amount, not accounting for the fee.

```
function DepositAndStakingERC20(
        address _token,
        uint256 _amount
    ) public override nonReentrant whenNotPaused {
        if (!IsSupportToken[_token]) {
            revert TokenIsNotSupported(_token);
        }
        if (_amount < MinStakeAmount[_token]) {
            revert LessThanMinStakeAmount(MinStakeAmount[_token], _amount);
        }

        IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
        uint256 PoolIndex = Pools[_token].length - 1;
        if (Pools[_token][PoolIndex].startTimestamp > block.timestamp) {
            Users[msg.sender].push(
                User({
                    isWithdrawed: false,
                    StartPoolId: PoolIndex,
                    EndPoolId: 0,
                    token: _token,
```

```
                    Amount: _amount
                })
            );
            Pools[_token][PoolIndex].TotalAmount += _amount;
        } else {
            revert NewPoolIsNotCreate(PoolIndex);
        }
        FundingPoolBalance[_token] += _amount;
        emit StarkingERC20Event(msg.sender, _token, _amount);
    }
```

**Recommendation**

Check the actual received amount after the transfer by comparing the contract's balance before and after the transfer. Use this actual received amount to update the user's staked amount and the pool's total amount.

## 3.2 call() Should be Used Instead of transfer() on An Address Payable

- ID: SRS-002
- Severity: Medium
- Auditors: Polaristow,Alex
- Likelihood: N/A
- Impact: N/A

### Description

The transfer() and send() functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

```
    function SendAssertToUser(
    address _token,
    address to,
    uint256 _amount
) internal returns (bool) {
    if (!IsSupportToken[_token]) {
        revert TokenIsNotSupported(_token);
    }
    FundingPoolBalance[_token] -= _amount;
    if (_token == address(ContractsAddress.ETHAddress)) {
        if (address(this).balance < _amount) {
```

```
            revert NotEnoughETH();
        }
        payable(to).transfer(_amount);
    } else {
        if (IERC20(_token).balanceOf(address(this)) < _amount) {
            revert NotEnoughToken(_token);
        }
        IERC20(_token).safeTransfer(to, _amount);
    }
    return true;
}
```

**Recommendation**

It is recommended to use call() instead of transfer(), but be sure to respect the CEI pattern and/or add re-entrancy guards, as several hacks already happened in the past due to this recommendation not being fully understood.

## 3.3 Handling Chain ID Changes to Prevent Withdrawal Failures

- ID: SRS-003
- Auditors: Polaristow
- Impact: N/A
- Severity: High
- Likelihood: N/A

### Description

The current implementation relies on hardcoded chain IDs to determine the appropriate bridge for withdrawing ETH to Layer 1. If the blockchain undergoes a hard fork that changes the chain ID, the function may fail to identify the correct bridge, leading to an inability to withdraw funds.

```
function WithdrawETHtoL1(
    address _to,
    uint256 _amount
 }external payable onlyRole(ReLayer) returns (bool){
    uint256 Blockchain = block.chainid;
    if (_amount > address(this).balance) {
        revert NotEnoughETH();
    }
    if (Blockchain == 0x82750) {
        //Scroll https://chainlist.org/chain/534352
        IScrollStandardL2ETHBridge(
```

```
            ContractsAddress.ScrollL2StandardETHBridge
        ).withdrawETH{gas: MAX_GAS_Limit, value: _amount}(
            _to,
            _amount,
            uint256(MAX_GAS_Limit)
        );
    ...
    }else {
        revert ErrorBlockChain();
    }
    FundingPoolBalance[ContractsAddress.ETHAddress] -= _amount;
    emit WithdrawETHtoL1Success(
        block.chainid,
        block.timestamp,
        _to,
        _amount
    );
    return true;
}
```

**Recommendation**

To mitigate this issue, introduce a configurable variable for the chain ID, allowing the system to adapt to changes in the chain ID without requiring code modifications.

## 3.4 Underflow Bug in Pool Index Calculation

- ID: SRS-004
- Severity: low
- Auditors: Polaristow, tanliwei
- Likelihood :N/A
- Impact: N/A

### Description

The line uint256 PoolIndex = Pools[address(ContractsAddress.ETHAddress)].length - 1; will cause an underflow and revert if the length is 0, so the condition if (Pools[address(ContractsAddress.ETHAddress)].length == 0)will never be executed.

```
uint256 PoolIndex = Pools[address(ContractsAddress.ETHAddress)].length -
    1;
if (Pools[address(ContractsAddress.ETHAddress)].length == 0) {
    revert NewPoolIsNotCreate(1);
}
```

**Recommendation**

Check the length of the array before performing the subtraction to ensure it is not zero.

```
if (Pools[address(ContractsAddress.ETHAddress)].length == 0) {
    revert NewPoolIsNotCreate(1);
}
uint256 PoolIndex = Pools[address(ContractsAddress.ETHAddress)].length - 1;
```

By performing the length check first, we ensure that the subtraction is only executed when the length is greater than zero, thus preventing the underflow and unintended revert.

## 3.5 MessageManager.claimMessage function lacks a check to prevent replay

- ID: SRS-005
- Auditors: ACaiSec
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

In the function, there is no check to see if cliamMessageStatus[messageHash] has been used, which may result in the same messageHash being used multiple times, thus generating multiple MessageClaimed messages.

```
        function claimMessage(
        uint256 sourceChainId,
        uint256 destChainId,
        address _to,
        uint256 _fee,
        uint256 _value,
        uint256 _nonce
    ) external onlyTokenBridge nonReentrant {
        bytes32 messageHash = keccak256(
            abi.encode(sourceChainId, destChainId, _to, _fee, _value, _nonce)
        );
        cliamMessageStatus[messageHash] = true;
        emit MessageClaimed(sourceChainId, destChainId, messageHash);
    }
```

### Recommendation

It is recommended to add checks to prevent reuse

```
    function claimMessage(
        uint256 sourceChainId,
        uint256 destChainId,
        address _to,
        uint256 _fee,
```

```
        uint256 _value,
        uint256 _nonce
    ) external onlyTokenBridge nonReentrant {
        bytes32 messageHash = keccak256(
            abi.encode(sourceChainId, destChainId, _to, _fee, _value, _nonce)
        );
        require(!cliamMessageStatus[messageHash], "Has been used!");
        cliamMessageStatus[messageHash] = true;
        emit MessageClaimed(sourceChainId, destChainId, messageHash);
    }
```

## 3.6 The setSupportToken function parameter check is missing

- ID: SRS-006
- Auditors: ACaiSec
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

L1PoolManager.setSupportToken does not check the _isSupport parameter. Improper input by the administrator may harm the normal operation of the project.

For the input parameters, assume IsSupportToken[_token] == false, and _isSupport == false. After the assignment of IsSupportToken[_token] = _isSupport;, IsSupportToken[_token] == false is still obtained. However, after that, the function will create a Pool data structure for this _token.

```
     function setSupportToken(
        address _token,
        bool _isSupport,
        uint32 startTimes
    ) external override onlyRole(DEFAULT_ADMIN_ROLE) {
        if (IsSupportToken[_token]) {
            revert TokenIsAlreadySupported(_token, _isSupport);
        }
        IsSupportToken[_token] = _isSupport;
        //genesis pool
        ...
        SupportTokens.push(_token);
        emit SetSupportTokenEvent(_token, _isSupport);
    }
```

### Recommendation

Performing the following operations will endanger the normal operation of the project.

```
setSupportToken(_token, false, startTimes)
```

Suggest to remove _isSupport parameter

```
    function setSupportToken(
        address _token,
        uint32 startTimes
    )external override onlyRole(DEFAULT_ADMIN_ROLE) {
        if (IsSupportToken[_token]) {
            revert TokenIsAlreadySupported(_token);
        }
        IsSupportToken[_token] = true;
        ...
    }
```

## 3.7 Redundant check for uint256 type variables

- ID: SRS-007
- Auditors: ACaiSec
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

The value range of uint256 type variables is [0, 2 ** 256], so the judgment condition (_amount < 0) will never be met. This check is redundant and it is recommended to modify it.

```
    function setMinStakeAmount(
        address _token,
        uint256 _amount
    ) external override onlyRole(DEFAULT_ADMIN_ROLE) {
        if (_amount < 0) {  // @note 不必要的比较
            revert LessThanZero(_amount);
        }
        MinStakeAmount[_token] = _amount;
        emit SetMinStakeAmountEvent(_token, _amount);
    }
```

### Recommendation

```
    function setMinStakeAmount(
        address _token,
        uint256 _amount
```

```
    ) external override onlyRole(DEFAULT_ADMIN_ROLE) {
        if (_amount == 0) {
            revert Zero(_amount);
        }
        MinStakeAmount[_token] = _amount;
        emit SetMinStakeAmountEvent(_token, _amount);
    }
```

## 3.8 WithdrawOrClaimBySimpleAsset function parameter type optimization

- ID: SRS-008
- Auditors: ACaiSec
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

L1PoolManager.WithdrawOrClaimBySimpleAsset function uses a parameter of type int256 and then changes its value to uint256. It is recommended to use the uint256 parameter type directly.

```
for (int256 i = 0; uint256(i) < Users[_user].length; i++) {
    uint256 index = uint256(i);
    ...
}
```

### Recommendation

```
for (uint256 index = 0; index < Users[_user].length; index++) {
    ...
}
```

## 3.9 The calculation method of updating the TotalAmount variable is wrong

- ID: SRS-009
- Auditors: ACaiSec, tanliwei
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

### Description

This problem occurs in both the WithdrawOrClaimBySimpleID function and the WithdrawOrClaimBySimpleAsset function. Here,the WithdrawOrClaimBySimpleID function is used as an example.

The function first subtracts the Amount number of tokens in the last pool through Pools[_token][EndPoolId].TotalAmount -= Users[_user][index].Amount;. If IsWithdraw = false, the user only takes away the Reward and does not withdraw the principal Amount. At this time, the value of Pools[_token][EndPoolId].TotalAmount is smaller than the actual situation.

```
function WithdrawOrClaimBySimpleID(
        address _user,
        uint index,
        bool IsWithdraw
    ) internal {
        address _token = Users[_user][index].token;
        uint256 EndPoolId = Pools[_token].length - 1;
        Pools[_token][EndPoolId].TotalAmount -= Users[_user][index].Amount;

        uint256 Reward = 0;
        uint256 Amount = Users[_user][index].Amount;
        uint256 startPoolId = Users[_user][index].StartPoolId;
        if (startPoolId > EndPoolId) {
            revert NoReward();
        }

        for (uint256 j = startPoolId; j < EndPoolId; j++) {
            if (j > Pools[_token].length - 1) {
                revert NewPoolIsNotCreate(j);
            }
            uint256 _Reward = (Amount * Pools[_token][j].TotalFee) /
                Pools[_token][j].TotalAmount;
            Reward += _Reward;
            Pools[_token][j].TotalFeeClaimed += _Reward;
        }
        require(Reward > 0, "No Reward");
        Amount += Reward;
        Users[_user][index].isWithdrawed = true;
        if (IsWithdraw) {
            ...
        }
        else {
            Users[_user][index].StartPoolId = EndPoolId;
            SendAssertToUser(_token, _user, Reward);
            emit ClaimReward(_user, startPoolId, EndPoolId, _token, Reward);
        }

    }
```

When the administrator calls the CompletePoolAndNew function to update, according to the code TotalAmount: Pools[_token][PoolIndex].TotalAmount, it can be known that the value of TotalAmount of the new pool is equal to the TotalAmount of the previous pool. If the WithdrawOrClaimBySimpleID function has been called in the previous pool to perform the Claim operation, it will cause errors in the subsequent pool status updates, resulting in asset losses for users and project parties.

```
function CompletePoolAndNew(
        Pool[] memory CompletePools
    ) external payable onlyRole(ReLayer) {
        for (uint256 i = 0; i < CompletePools.length; i++) {
            address _token = CompletePools[i].token;
            uint PoolIndex = Pools[_token].length - 1;
            Pools[_token][PoolIndex-1].IsCompleted = true;
            if (PoolIndex-1 != 0){
                Pools[_token][PoolIndex-1].TotalFee = FeePoolValue[_token];
                FeePoolValue[_token] = 0;
            }
            uint32 startTimes = Pools[_token][PoolIndex].endTimestamp;
            Pools[_token].push(
                Pool({
                    startTimestamp: startTimes,
                    endTimestamp: startTimes + periodTime,
                    token: _token,
                    TotalAmount: Pools[_token][PoolIndex].TotalAmount,
                    TotalFee: 0,
                    TotalFeeClaimed: 0,
                    IsCompleted: false
                })
            );
            emit CompletePoolEvent(_token, PoolIndex);
        }
    }
```

**Poc**

Scenario 1:

User A has 5000 / 10000 = 50% of the total pool share

In Pools[n]:

User A Amount = 5000, TotalFee = 10000, TotalAmount = 10000

User A Claim reward => Pools[n].TotalAmount = 10000 - 5000 = 5000

Call CompletePoolAndNew => Pools[n + 1].TotalAmount = Pools[n].TotalAmount

In Pools[n + 1]:

Amount = 5000, TotalFee = 10000, TotalAmount = 50000

At this point, User A has occupied all the shares, 5000 / 5000 = 100%, and all TotalFee will be obtained by him.

Modification suggestions:

```
function WithdrawOrClaimBySimpleID(
        address _user,
        uint index,
        bool IsWithdraw
    ) internal {
        address _token = Users[_user][index].token;
        uint256 EndPoolId = Pools[_token].length - 1;
        uint256 Reward = 0;
        uint256 Amount = Users[_user][index].Amount;
        uint256 startPoolId = Users[_user][index].StartPoolId;
        if (startPoolId > EndPoolId) {
            revert NoReward();
        }

        for (uint256 j = startPoolId; j < EndPoolId; j++) {
            if (j > Pools[_token].length - 1) {
                revert NewPoolIsNotCreate(j);
            }
            uint256 _Reward = (Amount * Pools[_token][j].TotalFee) /
                Pools[_token][j].TotalAmount;
            Reward += _Reward;
            Pools[_token][j].TotalFeeClaimed += _Reward;
        }
        require(Reward > 0, "No Reward");
        Amount += Reward;
        Users[_user][index].isWithdrawed = true;
        if (IsWithdraw) {
            Pools[_token][EndPoolId].TotalAmount -= Users[_user][index].Amount;
            ...
        }
        else {
            Users[_user][index].StartPoolId = EndPoolId;
            SendAssertToUser(_token, _user, Reward);
            emit ClaimReward(_user, startPoolId, EndPoolId, _token, Reward);
        }

    }
```

## 3.10 Loses precision in calculating Reward, which will cause user asset loss

- ID: SRS-010
- Severity: Critical
- Auditors: ACaiSec
- Likelihood: N/A

- Impact: N/A

### Description

In the WithdrawOrClaimBySimpleID function and the WithdrawOrClaimBySimpleAsset function, the following code snippet is used to calculate the user's Reward. In the calculation process of the _Reward variable, due to the rounding down feature of solidity, this part of the calculation will lose precision, resulting in user asset loss.

```
for (uint256 j = startPoolId; j < EndPoolId; j++) {
    if (j > Pools[_token].length - 1) {
        revert NewPoolIsNotCreate(j);
    }
    uint256 _Reward = (Amount * Pools[_token][j].TotalFee) /
        Pools[_token][j].TotalAmount;
    Reward += _Reward;
    Pools[_token][j].TotalFeeClaimed += _Reward;
}
```

### Recommendation

Scenario 1:

TotalFee is too small, causing some users to not be able to get Reward

In Pools[n]:

TotalFee = 100, TotalAmount = 10000

In this scenario, only when the user's Amount >= 100 can the reward be obtained. If the user's Amount < 100, the funds corresponding to this part of the user will be locked in the contract. For example, if there are 100 users with Amount = 50, then there will be 5000 * 100 / 10000 = 50 (TotalFee * 50%) of Reward locked in the contract.

Scenario 2:

The loss of precision during the calculation process causes the user to receive less Reward than the actual amount.

In Pools[n]:

Amount = 100, TotalFee = 199, TotalAmount = 10000

In Pools[n + 1]:

Amount = 10, TotalFee = 199, TotalAmount = 10000

Since the solidity language rounds down when performing division, the following scenario will occur when calculating Reward:

In Pools[n]:

Reward = 100 * 199 / 10000 = 19900 / 10000 = ⌊ 1.99⌋ = 1

In Pools[n + 1]:

Reward = 10 * 199 / 10000 = 1990 / 10000 = ⌊ 0.199⌋ = 0

The proportion of Reward received by the user is only 1 / (1.99 + 0.199) = 0.456

Suggested change:

Add 1e18 precision to reward calculation

```
    for (uint256 j = startPoolId; j < EndPoolId; j++) {
        if (j > Pools[_token].length - 1) {
            revert NewPoolIsNotCreate(j);
        }
        uint256 _Reward = (Amount * Pools[_token][j].TotalFee) * 1e18 /
            Pools[_token][j].TotalAmount;
        Reward += _Reward;
        Pools[_token][j].TotalFeeClaimed += _Reward;    // While using TotalFeeClaime
d, remember to div 1e18 to recover the actually value.
    }
    require(Reward > 0, "No Reward");
    Amount += Reward / 1e18;    // div 1e18 of the sum Reward
```

When the following scenario is encountered again

In Pools[n]:

Amount = 100, TotalFee = 199, TotalAmount = 10000

In Pools[n + 1]:

Amount = 10, TotalFee = 199, TotalAmount = 10000

In Pools[n]:

Reward = 100 * 199 * 1e18 / 10000 = 100 * 199 * 1e16

In Pools[n + 1]:

Reward = 10 * 199 * 1e18 / 10000 = 10 * 199 * 1e16

Reward / 1e18 = ((100 * 199 * 1e14) + (10 * 199 * 1e14)) / 1e18

= (19900 + 1990) * 1e14 / 1e18

= 21890 * 1e14 / 1e18

= 2

The proportion of Reward obtained by the user reached 2 / (1.99 + 0.199) = 0.913

## 3.11 The DepositAndStaking type function does not fully check the Pools status

- ID: SRS-011
- Auditors: ACaiSec
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

The following three functions do not fully check Pools and cannot cover the two special scenarios of NewPoolIsNotCreate and PoolIsCompleted at the same time.

- DepositAndStakingERC20
- DepositAndStakingETH
- DepositAndStakingWETH

**Recommendation**

It is recommended to use a unified check method to check the two special scenarios of NewPoolIsNotCreate and PoolIsCompleted, taking DepositAndStakingWETH as an example

```
function DepositAndStakingWETH(
        uint256 amount
    ) public override nonReentrant whenNotPaused {
        if (amount < MinStakeAmount[address(ContractsAddress.WETH)]) {
            revert LessThanMinStakeAmount(MinStakeAmount[address(0)], amount);
        }
        uint256 PoolIndex = Pools[address(ContractsAddress.WETH)].length - 1;
        if (Pools[address(ContractsAddress.WETH)].length == 0) {
            revert NewPoolIsNotCreate(1);
        }
        if (Pools[address(ContractsAddress.WETH)][PoolIndex].IsCompleted) {
            revert PoolIsCompleted(PoolIndex);
        }
        IWETH(ContractsAddress.WETH).transferFrom(
            msg.sender,
            address(this),
            amount
        );
        ...
    }
```

## 3.12 DepositAndStaking function check is not rigorous

- ID: SRS-012
- Auditors: ACaiSec
- Impact: N/A
- Severity: Low
- Likelihood: N/A

**Description**

It is recommended to double-check the branch that executes DepositAndStakingETH to ensure that users will not lose assets due to inaccurate parameter configuration.

```
function DepositAndStaking(
```

```
        address _token,
        uint256 _amount
    ) public payable override whenNotPaused {
        if (msg.value > 0) {     // @note 建议进行双重检查，避免用户造成资产损失
            DepositAndStakingETH();
        } else if (_token == ContractsAddress.WETH) {
            DepositAndStakingWETH(_amount);
        } else if (IsSupportToken[_token]) {
            DepositAndStakingERC20(_token, _amount);
        }
    }
```

**Recommendation**

```
    function DepositAndStaking(
        address _token,
        uint256 _amount
    )public payable override whenNotPaused {
        if (msg.value > 0 && _token == ContractsAddress.ETHAddress) {
            DepositAndStakingETH();
        } else if (_token == ContractsAddress.WETH && msg.value == 0) {
            DepositAndStakingWETH(_amount);
        } else if (IsSupportToken[_token] && msg.value == 0) {
            DepositAndStakingERC20(_token, _amount);
        } else {
            revert();
        }
    }
```

```
// If the WETH address is passed in and ETH is sent, the user will incur a loss.
L1PoolManager(target).DepositAndStaking{value: 1 eth}(ContractsAddress.WETH, 1e18);
```

## 3.13 The pause function of the L2Pool Manager contract is missing

- ID: SRS-013
- Severity: High
- Auditors: ACaiSec
- Likelihood: N/A
- Impact: N/A

### Description

The L2PoolManager contract inherits the PausableUpgradeable contract, but the whenNotPaused modifier is not used in the code implementation of the L2PoolManager contract. This means that the pause function of the L2PoolManager contract is missing. This means that when an emergency occurs and some functions of the contract need to be

suspended, the functions in the contract cannot be suspended. The lack of this function will make the contract unable to cope with special scenarios, exposing the assets of users and project parties to risks.

```
contract L2PoolManager is IL2PoolManager, PausableUpgradeable, TokenBridgeBase {}
```

**Recommendation**

It is recommended to add modifier whenNotPaused() to the following function

- WithdrawETHtoL1
- WithdrawWETHToL1
- WithdrawERC20ToL1

```
    function WithdrawETHtoL1(
        address _to,
        uint256 _amount
    ) external payable whenNotPaused() onlyRole(ReLayer) returns (bool) {...}
```

## 3.14 The FundingPoolBalance data structure has security risks

- ID: SRS-014
- Auditors: ACaiSec
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

Updating FundingPoolBalance by passing parameters through the administrator may have security risks. The value of the parameter amount may be too large or too small for the actual token balance of the contract. In either case, the system bonus distribution will be incorrect, resulting in asset losses for users or project parties.

```
function UpdateFundingPoolBalance(address token, uint256 amount) external onlyRole(ReLayer) {
    FundingPoolBalance[token] = amount; // @note 建议改为 IERC20(token).balanceOf(this) 的形式更新
}
```

**Recommendation**

It is recommended to use the balance method to update the corresponding value in the FundingPoolBalance data structure

```
function UpdateFundingPoolBalance(address token, uint256 amount) external onlyRole(RelayerLayer) {
    if(token == ContractsAddress.ETHAddress){
        FundingPoolBalance[token] = this.balance();
    } else {
        FundingPoolBalance[token] = IERC20(token).balanceOf(this);
    }
}
```

## 3.15 TokenBridgeBase.setPerFee function parameter check missing

- ID: SRS-015
- Auditors: ACaiSec
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

setPerFee function does not check the passed _PerFee parameter value to ensure it is less than 1_000_000. Once the administrator sets its value to greater than 1_000_000, it will cause user funds to lose.

### Recommendation

The following changes are recommended

```
function setPerFee(uint256 _PerFee) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_PerFee < 1000000);
    PerFee = _PerFee;
}
```

## 3.16 The updates of the two key data structures IsSupportToken and SupportTokens are inconsistent

- ID: SRS-016
- Auditors: ACaiSec,Dicanoex
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

When the administrator enters (ERC20Address, false), the corresponding ERC20Address token will be removed from IsSupportToken, but the ERC20Address token will not be removed from SupportTokens. This leads to inconsistent updates of the two key data structures IsSupportToken and SupportTokens.

```
    function setSupportERC20Token(
        address ERC20Address,
        bool isValid
    )external onlyRole(DEFAULT_ADMIN_ROLE) {
        IsSupportToken[ERC20Address] = isValid;
        if (isValid) {
            SupportTokens.push(ERC20Address);    // @note isValid = false 时，没有将代币从 SupportTokens 中剔除
        }
    }
```

**Recommendation**

```
function setSupportERC20Token(
    address ERC20Address,
    bool isValid
)external onlyRole(DEFAULT_ADMIN_ROLE) {
    IsSupportToken[ERC20Address] = isValid;
    if (isValid) {
        SupportTokens.push(ERC20Address);
    }
    else {
        remove(ERC20Address);
    }
}


function remove(address token) {
    for (uint i = 0; i < SupportTokens.length - 1; i++){
        if(SupportTokens[i] == token){
            SupportTokens[i] = SupportTokens[length - 1];
            SupportTokens.pop();
            break;
        }
    }
}
```

## 3.17 TokenBridgeBase.setPerFee function parameter check missing

- ID: SRS-017
- Severity: Low

- Auditors: ACaiSec
- Likelihood: N/A
- Impact: N/A

### Description

TokenBridgeBase.SendAssertToUser function needs to check that the value of FundingPoolBalance[_token] is greater than the value of _amount when performing subtraction operations, otherwise a rollback will occur outside the expected scenario (rollback without error information), which is not conducive to understanding the error situation.

```
function SendAssertToUser(
        address _token,
        address to,
        uint256 _amount
    ) internal returns (bool) {
        if (!IsSupportToken[_token]) {
            revert TokenIsNotSupported(_token);
        }
        FundingPoolBalance[_token] -= _amount;  // @note 减法之前需要进行检查
        if (_token == address(ContractsAddress.ETHAddress)) {
            if (address(this).balance < _amount) {
                revert NotEnoughETH();
            }
            payable(to).transfer(_amount);
        } else {
            if (IERC20(_token).balanceOf(address(this)) < _amount) {
                revert NotEnoughToken(_token);
            }
            IERC20(_token).safeTransfer(to, _amount);
        }
        return true;
    }
```

### Recommendation

```
function SendAssertToUser(
        address _token,
        address to,
        uint256 _amount
    ) internal returns (bool) {
        if (!IsSupportToken[_token]) {
            revert TokenIsNotSupported(_token);
        }
        require(FundingPoolBalance[_token] >= _amount, "Insufficient token balance")
        FundingPoolBalance[_token] -= _amount;
    }
```

## 3.18 The stakingMessageNumber usage error

- ID: SRS-018
- Severity: Critical
- Auditors: ACaiSec
- Likelihood: N/A
- Impact: N/A

### Description

In the function TokenBridgeBase.BridgeInitiateStakingMessage , stakingMessageHash is constructed by stakingMessageNumber, while st akingMessageHash and stakingMessage Number + 1 are recorded as associated items in emit. Incorrect data recording directly affects the function of staking initialization and has a fatal impact on the data security of the contract.

```
1.     function BridgeInitiateStakingMessage(
2.         address from,
3.         address to,
4.         uint256 shares
5.     ) external returns (bool) {
6.         bytes32 stakingMessageHash = keccak256(
7.             abi.encode(
8.                 from,
9.                 to,
10.                shares,
11.                stakingMessageNumber
12.            )
13.        );
14.        stakingMessageNumber++;
15.        emit InitiateStakingMessage(
16.            from,
17.            to,
18.            shares,
19.            stakingMessageNumber,    // @note 错误的 stakingMessageNumber
20.            stakingMessageHash
21.        );
22.        return true;
23.    }
```

### Recommendation

```
1.  bytes32 stakingMessageHash = keccak256(
2.             abi.encode(
3.                 from,
4.                 to,
5.                 shares,
6.                 stakingMessageNumber
7.             )
8.         );
```

```
9.
10.         emit InitiateStakingMessage(
11.             from,
12.             to,
13.             shares,
14.             stakingMessageNumber,
15.             stakingMessageHash
16.         );
17.         stakingMessageNumber++;
```

## 3.19 TokenBridgeBase.PerFee incorrect comment

- ID: SRS-019
- Auditors: ACaiSec, tanliwei
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

The PerFee variable is annotated with 0.1% when it is defined, but the actual value assigned in the TokenBridgeBase.__TokenBridge_init function is 1%.

```
1. uint256 public PerFee; // 0.1%
2.
3. function __TokenBridge_init(
4. address _MultisigWallet,
5. address _messageManager
6. ) internal onlyInitializing {
7. MinTransferAmount = 0.1 ether;
8. PerFee = 10000; // 1%
9. _grantRole(DEFAULT_ADMIN_ROLE, _MultisigWallet);
10. messageManager = IMessageManager(_messageManager);
11. stakingMessageNumber = 1;
12. }
```

### Recommendation

```
1.  PerFee = 1000;
```

## 3.20 Mising invoking the initial functions of parent contracts

- ID: SRS-020
- Severity: Medium
- Auditors: tanliwei
- Likelihood: N/A
- Impact: N/A

### Description

In the MessageManager contract, the initialize function lacks invoking the __AccessControl_init(), and the __ReentrancyGuard_init() function.

```
1.  function initialize(address _poolManagerAddress) public initializer {
2.      poolManagerAddress = _poolManagerAddress;
3.      nextMessageNumber = 1;
4.  }
```

The __ReentrancyGuard_init function is key funciton to initialize the $._status as shown below:

```
1.  function __ReentrancyGuard_init() internal onlyInitializing {
2.      __ReentrancyGuard_init_unchained();
3.  }
4.
5.  function __ReentrancyGuard_init_unchained() internal onlyInitializing {
6.      ReentrancyGuardStorage storage $ = _getReentrancyGuardStorage();
7.      $._status = NOT_ENTERED;
8.  }
```

### Recommendation

Invoking the __AccessControl_init(), and the __ReentrancyGuard_init() function from the initialize function.

## 3.21 Users can drain tokens from the L1Pool Manager contract

- ID: SRS-021
- Severity: Critical
- Auditors: tanliwei
- Likelihood: N/A
- Impact: N/A

### Description

In the L1PoolManager contract, the function WithdrawOrClaimBySimpleID only pop the Users[_user] if the Users[_user].length > 0, as shown below:

```
    function WithdrawByID(uint i) external nonReentrant whenNotPaused {
        if (i >= Users[msg.sender].length) {
            revert OutOfRange(i, Users[msg.sender].length);
        }
        WithdrawOrClaimBySimpleID(msg.sender, i, true);
    }

    function WithdrawOrClaimBySimpleID(
        address _user,
        uint index,
        bool IsWithdraw
    ) internal {
        ...
        Amount += Reward;
        Users[_user][index].isWithdrawed = true;
        if (IsWithdraw) {
            Users[_user][index].isWithdrawed = true;
            SendAssertToUser(_token, _user, Amount);
            if (Users[_user].length > 1) {
                Users[_user][index] = Users[_user][Users[_user].length - 1];
                Users[_user].pop();
                ...
            }
        }
        ...
```

As a result, a user can repeatedly withdraw tokens if the user has only one User item, due to the conditionUsers[_user].length > 1 will always be false. So, the pop operation Users[_user].pop() is by-passed.

**Poc**

- Alice deposits 1 WETH, and Users[Alice].length becomes 1,
- Alice withdraw 1 WETH twice or third times and sucessfully, due to the Users[Alice].length > 1 is always false, the deposit record is never removed.

**Recommendation**

Always pop the corresponding User item no matter Users[Alice].length is greater than 1 or not.

## 3.22 Users will lost the WETH after the deposit

- ID: SRS-022
- Auditors: tanliwei,bsssss
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

### Description

In the L1PoolManager contract, the DepositAndStakingWETH transfers WETH from users to the contract L1PoolManager, as shown below:

```
1.  function DepositAndStakingWETH(
2.      uint256 amount
3.  ) public override nonReentrant whenNotPaused {
4.      if (amount < MinStakeAmount[address(ContractsAddress.WETH)]) {
5.          revert LessThanMinStakeAmount(MinStakeAmount[address(0)], amount);
6.      }
7.
8.      IWETH(ContractsAddress.WETH).transferFrom(
9.          msg.sender,
10.         address(this),
11.         amount
12.     );
13.
14.     uint256 PoolIndex = Pools[address(ContractsAddress.WETH)].length - 1;
15.     if (Pools[address(ContractsAddress.WETH)][PoolIndex].IsCompleted) {
16.         revert PoolIsCompleted(PoolIndex);
17.     }
18.     if (
19.         Pools[address(ContractsAddress.WETH)][PoolIndex].startTimestamp >
20.         block.timestamp
21.     ) {
22.         Users[msg.sender].push(
23.             User({
24.                 isWithdrawed: false,
25.                 StartPoolId: PoolIndex,
26.                 EndPoolId: 0,//@audit todo why EndPoolId is 0?
27.                 token: ContractsAddress.WETH,
28.                 Amount: amount
29.             })
30.         );
31.         Pools[address(ContractsAddress.WETH)][PoolIndex]
32.             .TotalAmount += amount;
33.     }
34.     FundingPoolBalance[ContractsAddress.WETH] += amount;
35.     emit StakingWETHEvent(msg.sender, amount);
```

```
36.     }
```

But, the deposit will always success even if the Pools[address(ContractsAddress.WETH)][PoolIndex].startTimestamp is less than the block.timestamp.

When the Pools[address(ContractsAddres s.WETH)][PoolIndex]. startTimestamp is less than the block.timestamp, the DepositAndStakingWETH neither insert a new User item into the Users[msg.sender] array nor revert like functions DepositAndStakingETH, and DepositAndStakingERC20 do. As a result, after the deposit of  WETH at this scenario, the  user unable to withdraw the corresponding WETH.

Note that the withdrawal should be done with corresponding User item:

```
1.     function WithdrawOrClaimBySimpleID(
2.         address _user,
3.         uint index,
4.         bool IsWithdraw
5.     ) internal {
6.         address _token = Users[_user][index].token;
7.         uint256 EndPoolId = Pools[_token].length - 1;
8.         Pools[_token][EndPoolId].TotalAmount -= Users[_user][index].Amount;
9.
10.        uint256 Reward = 0;
11.        uint256 Amount = Users[_user][index].Amount;
```

**Poc**

- Deposit WETH success when the Pools[address(ContractsAddress.WETH)][PoolIndex] .startTimestamp is less than the block.timestamp, and no item inserted into the Users [msg.sender] array.
- The user can not withdaw the WETH which deposited before, due to there is correspoding User item.

**Recommendation**

Deposit WETH shoud be reverted if the Pools[address(ContractsAddress.WETH)] [PoolIndex].startTimestamp is less than or equal to the block.timestamp.

## 3.23 The Withdraw event is emitted wrongly

- ID: SRS-023
- Auditors: tanliwei
- Impact:  N/A
- Severity:  Medium
- Likelihood:  N/A

**Description**

In the L1PoolManager contract, the WithdrawOrClaimBySimpleID function emits the Withdraw function if the IsWithdraw is true and the Users[_user].length is greater than 1, as shown below:

```
function WithdrawOrClaimBySimpleID(
    address _user,
    uint index,
    bool IsWithdraw
) internal {
...
    if (IsWithdraw) {
        Users[_user][index].isWithdrawed = true;
        SendAssertToUser(_token, _user, Amount);
        if (Users[_user].length > 1) {
            Users[_user][index] = Users[_user][Users[_user].length - 1];
            Users[_user].pop();

            emit Withdraw(
                _user,
                startPoolId,
                EndPoolId,
                _token,
                Amount - Reward,
                Reward
            );
        }
    }
    ...
```

The point is that the Withdraw event should be emitted once the IsWithdraw is true, no matter the Users[_user].length is greater than 1 or not.

**Recommendation**

Emitting the Withdraw event once the IsWithdraw is true, no matter the Users [_user] .length is greater than 1 or not.

## 3.24 Wrong argument in the LessThanMin StakeAmount event

- ID: SRS-024
- Auditors: tanliwei
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

In the L1PoolManager contract, the DepositAndStakingWETH emits the event LessThanMinStakeAmount if the amount is less than the MinStakeAmount [address(ContractsAddress.WETH)], it is shown below:

```
    function DepositAndStakingWETH(
        uint256 amount
    ) public override nonReentrant whenNotPaused {
        if (amount < MinStakeAmount[address(ContractsAddress.WETH)]) {
            revert LessThanMinStakeAmount(MinStakeAmount[address(0)], amount);
        }
```

But the argument in the LessThanMinStakeAmount event should be MinStakeAmount [ContractsAddress.WETH] rather than MinStakeAmount[address(0)].

**Recommendation**

Updating the argument passed to the LessThanMinStakeAmount event.

## 3.25 Wrongly set Users[_user][index].isWithdrawed if the IsWithdraw is false

- ID: SRS-025
- Auditors: tanliwei,Dicanoex
- Impact: N/A
- Severity: High
- Likelihood: N/A

### Description

In the TokenBridgeBase contract, the WithdrawOrClaimBySimpleID still sets the Users[_user][index].isWithdrawed as true if the parameter IsWithdraw is false, because the Users[_user][index].isWithdrawed is set as ture before and after the if branch.

When the parameter IsWithdraw is false, the Users[_user][index].isWithdrawed is marked as true but the token does not transfer to the user, because the transfer is done in the if branch and the condition requires the IsWithdraw should be true. It is harmful to the user and protocol due to the wrong user state.

```
    function WithdrawOrClaimBySimpleID(
        address _user,
        uint index,
        bool IsWithdraw
    ) internal {
        ...
        require(Reward > 0, "No Reward");
        Amount += Reward;
```

```
        Users[_user][index].isWithdrawed = true;
        if (IsWithdraw) {
            Users[_user][index].isWithdrawed = true;
            SendAssertToUser(_token, _user, Amount);
            ...
        }
```

**Poc**

If IsWithdraw is false, the first unconditional Users[_user][index].isWithdrawed = true is still executed. Marking the item as isWithdrawed effectively prevents users from withdrawing, resulting in a loss of user funds.

**Recommendation**

Only updating the Users[_user][index].isWithdrawed as true inner the if branch.

## 3.26 Incompatible With the Deflationary or Fee-on-Transfer Tokens

- ID: SRS-026
- Auditors: tanliwei
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

**Description**

In the L1PoolManager contract, the DepositAndStakingERC20 function contains a critical flaw that fails to correctly handle fee-on-transfer tokens. The contract assumes that the full amount of tokens will be transferred to the contract. However, if the token deducts a fee on the transfer, the contract will not receive the full amount. As a result, when the function attempts to withdraw the same amount from the _token, the contract may not have enough tokens to perform all the withdrawals.

```
        function DepositAndStakingERC20(
        address _token,
        uint256 _amount
    ) public override nonReentrant whenNotPaused {
        if (!IsSupportToken[_token]) {
            revert TokenIsNotSupported(_token);
        }
        if (_amount < MinStakeAmount[_token]) {
            revert LessThanMinStakeAmount(MinStakeAmount[_token], _amount);
        }

        IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
```

```
        uint256 PoolIndex = Pools[_token].length - 1;
        if (Pools[_token][PoolIndex].startTimestamp > block.timestamp) {
            Users[msg.sender].push(
                User({
                    isWithdrawed: false,
                    StartPoolId: PoolIndex,
                    EndPoolId: 0,
                    token: _token,
                    Amount: _amount
                })
            );
            Pools[_token][PoolIndex].TotalAmount += _amount;
        }
```

**Recommendation**

Calculating the actual balance received by subtracting the balance of the token after the transfer to that of before the transfer, and applying the actual balance for the following business.

```
uint256 initialBalance = IERC20(_token).balanceOf(address(this));
IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);
uint256 finalBalance =  IERC20(_token).balanceOf(address(this));
uint256 actualReceived = finalBalance - initialBalance;
```

## 3.27 Potential Duplicated Items in the SupportTokens

- ID: SRS-027
- Auditors: tanliwei
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

In the TokenBridgeBase contract, the DEFAULT_ADMIN_ROLE role can use the setSupportERC20Token function to insert token address into the array SupportTokens.

```
    function setSupportERC20Token(
        address ERC20Address,
        bool isValid
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        IsSupportToken[ERC20Address] = isValid;
        if (isValid) {
            SupportTokens.push(ERC20Address);
        }
```

```
9.     }
```

But, the function does not check if the token exist in the SupportTokens array before insert it, as a result, the same token could be inserted into the array multiple times.

**Recommendation**

Checking if the token exists in the array SupportTokens or not before insert the token.

## 3.28 The word Assert should be Asset

- ID: SRS-028
- Auditors: tanliwei
- Impact: N/A
- Severity: Low
- Likelihood: N/A

**Description**

In the TokenBridgeBase contract, the functions QuickSendAssertToUser, and SendAssertToUser should be renamed as QuickSendAssetToUser, and SendAssetToUser.Note that it is Asset rather than Assert.

**Recommendation**

Correcting the typos

## 3.29 Checks-Effects-Interactions Pattern is not adopted

- ID: SRS-029
- Auditors: tanliwei,Dicanoex
- Impact: N/A
- Severity: Low
- Likelihood: N/A

**Description**

In the TokenBridgeBase contract, the BridgeFinalizeERC20 firstly tranfers tokens to the to address, then update the storage state FundingPoolBalance [ERC20Address], which violate the Checks-Effects-Interaction pattern.

Reference: Use the Checks-Effects-Interactions Pattern

```
    function BridgeFinalizeERC20(
        uint256 sourceChainId,
        uint256 destChainId,
        address to,
        address ERC20Address,
        uint256 amount,
        uint256 _fee,
        uint256 _nonce
    ) external onlyRole(ReLayer) returns (bool) {
        if (destChainId != block.chainid) {
            revert sourceChainIdError();
        }
        if (!IsSupportChainId(sourceChainId)) {
            revert ChainIdIsNotSupported(sourceChainId);
        }
        if (!IsSupportToken[ERC20Address]) {
            revert TokenIsNotSupported(ERC20Address);
        }
        IERC20(ERC20Address).safeTransfer(to, amount);
        FundingPoolBalance[ERC20Address] -= amount;
```

**Recommendation**

Update the state firstly, then transfer tokens to the to address, to match the Checks-Effects-Interactions.

## 3.30 Missing passing the bridge token address to the messageManager

- ID: SRS-030
- Severity: Medium
- Auditors: tanliwei,bsssss,Alex
- Likelihood: N/A
- Impact: N/A

### Description

In the TokenBridgeBase contract, users can bridge different tokens with functions BridgeInitiateETH, BridgeInitiateWETH, and BridgeInitiateERC20, however, none of them pass the key parameter, token address, to the contract messageManager.

```
    function BridgeInitiateETH(
        uint256 sourceChainId,
        uint256 destChainId,
        address to
    ) external payable returns (bool) {
```

```
        ...
        messageManager.sendMessage(block.chainid, destChainId, to, amount, fee);
        ...
    }
    function BridgeInitiateWETH(
        uint256 sourceChainId,
        uint256 destChainId,
        address to,
        uint256 value
    ) external returns (bool) {
        ...
        messageManager.sendMessage(sourceChainId, destChainId, to, amount, fee);
        ...
    }
    function BridgeInitiateERC20(
        uint256 sourceChainId,
        uint256 destChainId,
        address to,
        address ERC20Address,
        uint256 value
    ) external returns (bool) {
    ...
        messageManager.sendMessage(sourceChainId, destChainId, to, amount, fee);
    ...
    }
```

```
//MessageManager.sol
    function sendMessage(
        uint256 sourceChainId,
        uint256 destChainId,
        address _to,
        uint256 _value,
        uint256 _fee
    ) external onlyTokenBridge {
        if (_to == address(0)) {
            revert ZeroAddressNotAllowed();
        }
        uint256 messageNumber = nextMessageNumber;
        bytes32 messageHash = keccak256(
            abi.encode(
                sourceChainId,
                destChainId,
                _to,
                _fee,
                _value,
                messageNumber
            )
```

```
        );
        nextMessageNumber++;
        sentMessageStatus[messageHash] = true;
        emit MessageSent(
            sourceChainId,
            destChainId,
            msg.sender,
            _to,
            _fee,
            _value,
            messageNumber,
            messageHash
        );
    }
```

As a result, it is possible to mess up the token and the amount of token users bridged.

**Recommendation**

Passing and recording the key parameter to the messageManager, token address.

## 3.31 There is no minimum amount for the bridge amount

- ID: SRS-031
- Auditors: tanliwei,Alex
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

**Description**

The BridgeInitiateERC20 function of the TokenBridgeBase contract charges fee from user when they brige ERC20 tokens as shown below:

```
    function BridgeInitiateERC20(
        uint256 sourceChainId,
        uint256 destChainId,
        address to,
        address ERC20Address,
        uint256 value
    ) external returns (bool) {
        if (sourceChainId != block.chainid) {
            revert sourceChainIdError();
```

```
        }
        if (!IsSupportChainId(destChainId)) {
            revert ChainIdIsNotSupported(destChainId);
        }
        if (!IsSupportToken[ERC20Address]) {
            revert TokenIsNotSupported(ERC20Address);
        }

        uint256 BalanceBefore = IERC20(ERC20Address).balanceOf(address(this));
        IERC20(ERC20Address).safeTransferFrom(msg.sender, address(this), value);
        uint256 BalanceAfter = IERC20(ERC20Address).balanceOf(address(this));
        uint256 amount = BalanceAfter - BalanceBefore;
        FundingPoolBalance[ERC20Address] += value;
        uint256 fee = (amount * PerFee) / 1_000_000;
        amount -= fee;
```

However, there is no minimum amount for the bridge amount, which results in the fee could be zero if the amount is less than 100 when the PerFee is 10000, or if the amount is less than 10000 when the PerFee is set to 100.

**Poc**

- The decimal of WBTC is 8
- 10000 wei WBTC worth $6.9 since price of BTC is $69000
- if a user bridge 9999 wei WBTC worth ~$6.9 when the PerFee is 100, the fee charge would be zero, 9999*100/1000000 = 0

**Recommendation**

Adding minimum amount check for the bridge amount.

## 3.32 There is a risk that users cannot receive ETH

- ID: SRS-032
- Auditors: 7g
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

```
function SendAssertToUser(
        address _token,
        address to,
        uint256 _amount
    ) internal returns (bool) {
        if (!IsSupportToken[_token]) {
```

```
            revert TokenIsNotSupported(_token);
        }
        FundingPoolBalance[_token] -= _amount;
        if (_token == address(ContractsAddress.ETHAddress)) {
            if (address(this).balance < _amount) {
                revert NotEnoughETH();
            }
            payable(to).transfer(_amount);
        } else {
            if (IERC20(_token).balanceOf(address(this)) < _amount) {
                revert NotEnoughToken(_token);
            }
            IERC20(_token).safeTransfer(to, _amount);
        }
        return true;
    }
```

payable(to).transfer(_amount);

The transfer function will send 2300 gas, which is enough for basic operations on the receiving side (such as event logging), but if the operations performed by the receiving contract require more gas, the transfer will fail, causing the transaction to roll back.

**Recommendation**

The fixed gas limit of transfer may cause problems when network conditions change. Using call can provide more flexibility, allowing you to specify a gas limit or no gas at all. It is recommended to use call instead of transfer, combined with appropriate error handling to ensure safety.

## 3.33 Gas optimization: cache array length

- ID: SRS-033
- Auditors: 7g,Dicanoex
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

```
function getPrincipal() public view returns (KeyValuePair[] memory){
        KeyValuePair[] memory result = new KeyValuePair[](SupportTokens.length);
        for (uint256 i = 0; i < SupportTokens.length; i++) {
            //...
        }
    }
```

**Recommendation**

Cache array length: Cache **SupportTokens.length** into the local variable length to avoid reading SupportTokens.length** in each loop and reduce Gas consumption.

## 3.34 The method signature is used incorrectly

- ID: SRS-034
- Auditors: 7g,Dicanoex
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

```
1.  bool success = SafeCall.callWithMinGas(
2.              shareAddress,
3.              gasLimit,
4.              0,
5.              abi.encodeWithSignature("TransferShareTo(address,address,uint256, uint256)
", from, to, shares, stakeMessageNonce)
6.          );
```

**Recommendation**

TransferShareTo(address,address,uint256, uint256) There is an extra space at the end of the uint256 signature, which will cause the call to always fail.

## 3.35 SupportTokens does not clean up unsupported tokens

- ID: SRS-035
- Auditors: 7g,Dicanoex
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

```
1.  function setSupportERC20Token(
2.          address ERC20Address,
3.          bool isValid
4.      ) external onlyRole (DEFAULT_ADMIN_ROLE) {
5.         IsSupportToken[ERC20Address] = isValid;
6.          if (isValid) {
```

```
            SupportTokens.push(ERC20Address);
        }
    }
```

**Recommendation**

If USDCe cross-chain is supported, it is pushed to the SupportTokens list, but when it is subsequently offline and no longer supported, it is not removed from the SupportTokens in a timely manner, resulting in users being able to continue to cross the asset.

## 3.36 NewPoolIsNotCreate prompt is inconsistent

- ID: SRS-036
- Auditors: 7g
- Impact: N/A
- Severity: Low
- Likelihood: N/A

**Description**

The prompt for staking ERC20 is revert NewPoolIsNotCreate(PoolIndex); while the prompt for ETH and WETH is revert NewPoolIsNotCreate(PoolIndex+1);

## 3.37 Users cannot withdraw funds or receive earnings

- ID: SRS-037
- Auditors: 7g
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

**Description**

```
uint256 _Reward = (Amount * Pools[_token][j].TotalFee) /
                Pools[_token][j].TotalAmount;
            Reward += _Reward;
            Pools[_token][j].TotalFeeClaimed += _Reward;
```

**Recommendation**

Because Pools[_token][j].TotalAmount is a division function, it may be 0 for a period of time, and no one can receive the income during this period of time.

## 3.38 Users can withdraw ETH repeatedly indefinitely

- ID: SRS-038
- Auditors: 7g
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

### Description

```
    if (Users[_user].length > 1) {
                Users[_user][index] = Users[_user][Users[_user].length - 1];
                Users[_user].pop();
```

In WithdrawByID, the attacker only needs to deposit 10 ETH in the first cycle, and then can withdraw 10 ETH in each cycle. Because Users[_user].length > 1, it fails to account for the situation of only depositing once. This leads to the attack.

### Poc

```
function testAttackWithdrawByID() public {
        L1PoolManager pool = L1PoolManager(address(l1Poolproxy));
        CompleteETHPoolAndNew(address(ETHAddress), 0);

        deal(address(this), 100 ether);
        pool.DepositAndStakingETH{value: 10 ether}();
        address alice = makeAddr("alice");
        deal(alice, 1 ether);

        vm.prank(alice);

        // stake 1 eth
        pool.DepositAndStakingETH{value: 1 ether}();

        CompleteETHPoolAndNew(address(ETHAddress), 0);

        for (uint256 i = 0; i < 5; i++) {
            // create trade fee
            pool.BridgeInitiateETH{value: 1 ether}(block.chainid, 42161, makeAddr("bob"));
            CompleteETHPoolAndNew(address(ETHAddress), 0);

            vm.prank(alice);
            pool.WithdrawByID(0); // withdraw 1 eth
        }
        console.log("balance3:", alice.balance);
        assertLe(alice.balance, 1.1 ether, "expect alice balance= 1eth + some fee");
```

```
27.     }
```

## 3.39 Protocol Relyer on Administrator Actions

- ID: SRS-039
- Auditors: bsssss
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

The protocol relies heavily on administrator actions. Such as if the administrator fails to call the CompletePoolAndNew() function in a timely manner to add a new staking period before the latest Pools[_token][index].endTimestamp expires, users will be unable to make new deposits and stakes(The check if (Pools[address (ContractsAddress.ETHAddress)][PoolIndex].startTimestamp > block.timestamp)). Or the unexpected calling the fucntion UpdateFundingPoolBalance(). There creates a risk to user funds.

### Recommendation

The administrator should ensure the protocol operates correctly.

## 3.40 Gas Optimization:Redundant Implementaion

- ID: SRS-040
- Auditors: bsssss
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

The statement if (_amount < 0) { revert LessThanZero(_amount); } in the setMinStakeAmount() function is redundant because the parameter _amount is of type uint256.

The statement Users[_user][index].isWithdrawed = true; in the WithdrawOrClaimBySimpleID() function (line 236) is redundant because

the Users[_user][index] element will be removed. The same issue also exists in the function WithdrawOrClaimBySimpleAsset().

The statement if (j > Pools[_token].length - 1) {revert NewPoolINotCreate (j); } in the WithdrawOrClaimBySimpleID() function is redundant because the statement j < EndPoolId and uint256 EndPoolId = Pools[_token].length - 1;. The same issue also exists in the function WithdrawOrClaimBySimpleAsset().

The statement if (startPoolId > EndPoolId) { revert NoReward(); } in the WithdrawOrClaimBySimpleID() function is redundant because it is impossible. The same issue also exists in the function WithdrawOrClaimBySimpleAsset().

The statement if (Pools[address(ContractsAddress.WETH)][PoolIndex]. IsCompleted) {revert PoolIsCompleted(PoolIndex); } in the DepositAndStakingWETH () function is redundant because it is impossible.

### Code Snippet

https://github.com/solid-rock-security/bssssss-bridge-contracts/blob/main/src/core/L1/L1PoolManager.sol#L748-L750

https://github.com/solid-rock-security/bssssss-bridge-contracts/blob/main/src/core/L1/L1PoolManager.sol#L236-L240

https://github.com/solid-rock-security/bssssss-bridge-contracts/blob/main/src/core/L1/L1PoolManager.sol#L224-L226

https://github.com/solid-rock-security/bssssss-bridge-contracts/blob/main/src/core/L1/L1PoolManager.sol#L282-L284

https://github.com/solid-rock-security/bssssss-bridge-contracts/blob/main/src/core/L1/L1PoolManager.sol#L158-L160

### Recommendation

Rmove redundant code.

## 3.41 Potential Failed Transfer ETH to User

- ID: SRS-041
- Auditors: bsssss
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

In the function BridgeFinalizeETH(), the statement payable(to).transfer(amount) is used to send ether to the user. However, due to the transfer's 2300 gas limit, it will fail if the recipient address is a multisig wallet or an Account Abstraction Wallet.

```
payable(to).transfer(amount);
```

**Poc**

See the discussion here.

**Recommendation**

Use Address.call with a gas limit instead of transfer.

## 3.42 Lack of Storage Gaps in the Contract TokenBridgeBase

- ID: SRS-042
- Auditors: bsssss
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

According the OpenZeppelin Docs, it should add storage gaps in base contract.

**Recommendation**

Add storage gaps in base contract.

## 3.43 Incorrect Update isWithdrawed Flag

- ID: SRS-043
- Auditors: bsssss,Dicanoex
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

### Description

In the WithdrawOrClaimBySimpleID() function, the statement Users[_user][index]. isWithdrawed = true; on line 234 incorrectly updates the isWithdrawed flag. It should update the isWithdrawed flag within the if (IsWithdraw) code branch.

The same issue also exists in the function WithdrawOrClaimBySimpleAsset().

```
Users[_user][index].isWithdrawed = true;
        if (IsWithdraw) {
```

```
3.              Users[_user][index].isWithdrawed = true;
4.              SendAssertToUser(_token, _user, Amount);
5.  ...
```

**Recommendation**

Remove the statement Users[_user][index].isWithdrawed = true; on line 234.

## 3.44 Potential Dos in Reward Calculation Loop

- ID: SRS-044
- Auditors: bsssss
- Impact: N/A
- Severity: High
- Likelihood: N/A

**Description**

The function WithdrawOrClaimBySimpleID() calculates the user reward by looping through the PoolId. When the history of reward periods is very large, it may revert due to insufficient gas or the transaction gas cost may exceed the reward value.

The same issue also exist in funcation WithdrawOrClaimBySimpleAsset().

```
1.  for (uint256 j = startPoolId; j < EndPoolId; j++) {
2.              if (j > Pools[_token].length - 1) {
3.                  revert NewPoolIsNotCreate(j);
4.              }
5.              uint256 _Reward = (Amount * Pools[_token][j].TotalFee) /
6.                  Pools[_token][j].TotalAmount;
7.              Reward += _Reward;
8.              Pools[_token][j].TotalFeeClaimed += _Reward;
9.          }
```

## 3.45 Incorrect initialization of the TotalAmount

- ID: SRS-045
- Auditors: bsssss
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

### Description

The Relayer adds a new staking period through the function CompletePoolAndNew(). The new period's TotalAmount is initialized to zero. However, in the function Withdraw OrClaimBySimpleID(), the statement Pools[_token][EndPoolId] .TotalAmount -= Users[_user] [index].Amount; subtracts the requested amount from the latest period's TotalAmount. If the sum of the latest period's staked amount is less than a past staked amount, the user will be unable to withdraw or claim rewards (which is highly likely).

The same issue also exists in the function WithdrawOrClaimBySimpleAsset().

```
Pools[_token][EndPoolId].TotalAmount -= Users[_user][index].Amount;

Pools[_token].push(
                Pool({
                    startTimestamp: startTimes,
                    endTimestamp: startTimes + periodTime,
                    token: _token,
                    TotalAmount: Pools[_token][PoolIndex].TotalAmount,
                    TotalFee: 0,
                    TotalFeeClaimed: 0,
                    IsCompleted: false
                })
```

### Poc

Consider this scenario:

Alice deposits 100 tokens in the 2nd period. In the 3rd period, the total staked amount by all users is 99. Alice is unable to withdraw or claim rewards in the entire 3rd period.

### Recommendation

Update the initialization logic of the TotalAmount.

## 3.46 User unable withdraw asset after cliam reward

- ID: SRS-046
- Auditors: bsssss
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

### Description

As per the protocol design, users can choose to only claim rewards in the function WithdrawOrClaimBySimpleID(), and withdraw assets in the future. However, the statement require(Reward > 0, "No Reward"); makes it impossible for users to withdraw assets when the reward is zero. Once a user's stake reward is zero (which is highly likely due to the absence of new reward periods), their staked assets will be stuck in the contract.

The same issue also exists in the function WithdrawOrClaimBySimpleAsset().

**Recommendation**

Remove the statement require(Reward > 0, "No Reward");.

## 3.47 Incorrect Reward Calculation Logic-I

- ID: SRS-047
- Severity: Critical
- Auditors: bsssss
- Likelihood: N/A
- Impact: N/A

### Description

In the WithdrawOrClaimBySimpleID() function, the statement Pools[_token][EndPoolId].TotalAmount -= Users[_user][index].Amount incorrectly updates Pools[_token][EndPoolId].TotalAmount. It should update Pools[_token][index].TotalAmount instead of Pools[_token] [EndPoolId].TotalAmount.

The same issue also exist in function WithdrawOrClaimBySimpleAsset().

```
Pools[_token][EndPoolId].TotalAmount -= Users[_user][index].Amount;

        uint256 Reward = 0;
        uint256 Amount = Users[_user][index].Amount;
        uint256 startPoolId = Users[_user][index].StartPoolId;
        if (startPoolId > EndPoolId) {
            revert NoReward();
        }

        for (uint256 j = startPoolId; j < EndPoolId; j++) {
            if (j > Pools[_token].length - 1) {
                revert NewPoolIsNotCreate(j);
            }
            uint256 _Reward = (Amount * Pools[_token][j].TotalFee) /
                Pools[_token][j].TotalAmount;
```

### Poc

Consider this scenario:

Alice deposits 100 tokens in the 2nd period, making Pools[_token][2].TotalAmount now 100. Bob deposits 101 tokens in the 3rd period, making Pools[_token][3].TotalAmount now 101.

When Alice withdraws the deposited tokens in the 3rd period, Pools[_token][3].TotalAmount will be updated to 101 - 100 = 1. Then, in the 4th period, Bob's _Reward calculated as (Amount * Pools[_token][3].TotalFee) / Pools[_token][3].TotalAmount; will be 101 * TotalFee / 1, which results in 101 * TotalFee.

**Recommendation**

Update the reward calculation logic.

## 3.48 Incorrect Reward Calculation Logic-II

- ID: SRS-048
- Auditors: bsssss
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

**Description**

In the WithdrawOrClaimBySimpleID() function, the statement uint256 _Reward = (Amount * Pools[_token][j].TotalFee) / Pools[_token][j].TotalAmount; incorrectly calculates the user reward. The Amount is the user deposit amount in a past period, and the TotalAmount is the total stake amount for each period. The reward calculation logic is unreasonable.

The same issue also exists in the function WithdrawOrClaimBySimpleAsset().

**Poc**

Consider this scenario:

Alice deposits 100 tokens in the 2nd period. Bob deposits 1 token in the 3rd period, making Pools[_token][3].TotalAmount now 1.

When Alice claims the reward in the 4th period, Alice's _Reward calculated as (Amount * Pools[_token][3].TotalFee) / Pools[_token][3].TotalAmount; will be 100 * TotalFee / 1, which results in 100 * TotalFee.

**Recommendation**

Update the reward calculation logic.

## 3.49 Use safeApprove() instead approve()

- ID: SRS-049
- Auditors: Alex
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

Tokens not compliant with the ERC20 specification could return false from the approve function call to indicate the approval fails, while the calling contract would not notice the failure if the return value is not checked.

Additionally, the protocol supports USDT. For USDT, it is necessary to approve 0 first.

```
IERC20(_token).approve(
                ContractsAddress.ArbitrumOneL1ERC20Gateway,
                _amount
            );
```

## 3.50 Underpaying Optimism l2gas may lead to loss of funds

- ID: SRS-050
- Auditors: Alex
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

In the TransferAssertToBaseBridge() function, the protocol calls OptimismL1Bridge .depositERC20To() for cross-chain transfers, with _minGasLimit using gasleft().

```
    function TransferAssertToBaseBridge(
        address _token,
        address _to,
        uint256 _amount
    ) internal {
        if (_token == address(ContractsAddress.ETHAddress)) {
            IOptimismL1Bridge(ContractsAddress.BaseL1StandardBridge)
                .depositETHTo{value: _amount}(_to, 0, "");
        } else {
            address l2token = getOPL2TokenAddress(_token);
            IERC20(_token).approve(
                ContractsAddress.BaseL1StandardBridge,
```

```
            _amount
        );
        IOptimismL1Bridge(ContractsAddress.BaseL1StandardBridge)
            .depositERC20To(
                _token,
                l2token,
                _to,
                _amount,
                uint32(gasleft()),
                ""
            );
    }
}
```

The Optimism's standard token bridge initiates the cross-chain deposit by sending a cross-chain message to L2Bridge. If the l2 Gas is underpaid, finalizeDeposit() will fail, resulting in the loss of user funds.

Please refer to this issue for reference.
https://github.com/ethereum-optimism/optimism/blob/7cbda018196b58a71e2c0b4bc9e31a289235074e/packages/contracts-bedrock/src/universal/StandardBridge.sol#L347-L392

**Recommendation**

Given the potential risks of losing users' funds, we recommend to emphasize the risks in the documents.

## 3.51 The _l2TxGasLimit is set incorrectly

- ID: SRS-051
- Auditors: Alex
- Impact: N/A
- Severity: High
- Likelihood: N/A

### Description

In the TransferAssertToZkSyncBridge() function, the protocol sets _l2TxGasLimit to 0, which could result in insufficient gas, leading to transaction failure.

```
IZkSyncBridge(ContractsAddress.ZkSyncL1Bridge).deposit(
                _to,
                _token,
                _amount,
                0,
                0,
                address(this)
```

```
            );

///_l2TxGasLimit The L2 gas limit to be used in the corresponding L2 transaction
    function deposit(
        address _l2Receiver,
        address _l1Token,
        uint256 _amount,
        uint256 _l2TxGasLimit,
        uint256 _l2TxGasPerPubdataByte,
        address _refundRecipient
    )public payable nonReentrant returns (bytes32 l2TxHash) {
        require(_amount != 0, "2T"); // empty deposit amount
        uint256 amount = _depositFunds(msg.sender, IERC20(_l1Token), _amount);
        require(amount == _amount, "1T"); // The token has non-
standard transfer logic
```

There's a discussion within the zksync community regarding this issue.
[zkSync-Community-Hub/zksync-developers#79]

The _l2TxGasLimit should be computed using zks_estimateGasL1ToL2(), where the gas per pubdata is used as a parameter.

**Recommendation**

The suggestion is to dynamically calculate this value.

## 3.52 The gasPerPubdataByte is incorrect

- ID: SRS-052
- Auditors: Alex
- Impact: N/A
- Severity: High
- Likelihood: N/A

### Description

In the L1PoolManager.TransferAssertToZkSyncBridge() function, the protocol calls ZkSyncBridge.deposit() to transfer funds from the Ethereum chain to the zkSync chain. It's important to note that _l2TxGasPerPubdataByte is set to 0 in the protocol.

```
IZkSyncBridge(ContractsAddress.ZkSyncL1Bridge).deposit(
                _to,
                _token,
                _amount,
                0,
                0,
                address(this)
```

```
            );

///_l2TxGasLimit The L2 gas limit to be used in the corresponding L2 transaction
    function deposit(
        address _l2Receiver,
        address _l1Token,
        uint256 _amount,
        uint256 _l2TxGasLimit,
        uint256 _l2TxGasPerPubdataByte,
        address _refundRecipient
    ) public payable nonReentrant returns (bytes32 l2TxHash) {
        require(_amount != 0, "2T"); // empty deposit amount
        uint256 amount = _depositFunds(msg.sender, IERC20(_l1Token), _amount);
        require(amount == _amount, "1T"); // The token has non-
standard transfer logic
```

According to the description of the deposit() function,
the gasPerPubdataByteLimit parameter is used in the corresponding L2 transaction.
https://github.com/matter-labs/era-contracts/blob/f3630fcb01ad8b6e2e423a6f313abefe8502c3a2/l1-contracts/contracts/bridge/L1ERC20Bridge.sol#L162C39-L162C112

This parameter is crucial for every transaction on zkSync, and the official recommendation is to set it to 800.

https://docs.zksync.io/build/sdks/js/utils.html#gas

https://github.com/lambdaclass/zksync-web3-rs/blob/main/src/zks_utils.rs#L39

export const REQUIRED_L1_TO_L2_GAS_PER_PUBDATA_LIMIT = 800;
The best practices also emphasize consideringgasPerPubdataByte`.

https://docs.zksync.io/build/quick-start/best-practices.html#gasperpubdatabyte-should-be-taken-into-account-in-development

Setting it to 0 in the protocol might lead to unexpected issues with the transaction.

**Recommendation**

The suggestion is to use the correct value.

## 3.53 If the L2 deposit finalization transaction fails, the protocol will lose funds

- ID: SRS-053
- Auditors: Alex
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

### Description

In the L1PoolManager.TransferAssertToZkSyncBridge() function, the protocol calls ZkSyncBridge.deposit() to transfer funds from the Ethereum chain to the zkSync chain, specifying address(this) as the _refundRecipient.

```
IZkSyncBridge(ContractsAddress.ZkSyncL1Bridge).deposit{value: _amount}(
                _to,
                address(0),
                _amount,
                0,
                0,
                address(this)
            );

    function deposit(
        address _l2Receiver,
        address _l1Token,
        uint256 _amount,
        uint256 _l2TxGasLimit,
        uint256 _l2TxGasPerPubdataByte,
        address _refundRecipient
    ) public payable nonReentrant returns (bytes32 l2TxHash) {
        require(_amount != 0, "2T"); // empty deposit amount
        uint256 amount = _depositFunds(msg.sender, IERC20(_l1Token), _amount);
        require(amount == _amount, "1T"); // The token has non-
standard transfer logic
```

Let's take a closer look at the deposit() function. In this function, the protocol explicitly specifies that _refundRecipient is the address that will receive refund funds on L2 if the L2 deposit finalization transaction fails.
https://github.com/matter-labs/era-contracts/blob/f3630fcb01ad8b6e2e423a6f313abefe8502c3a2/l1-contracts/contracts/bridge/L1ERC20Bridge.sol#L163-L165

However, the protocol passes address(this), which is not controlled by the project on L2 and is likely someone else's address. This means that any refund funds will be sent to someone else's address, leading to a loss of funds for the protocol.

**Recommendation**

Replace address(this) with the designated address.

## 3.54 Refunded funds from cross-chain transactions will be lost

- ID: SRS-054
- Auditors: Alex
- Severity: Critical
- Likelihood: N/A

- Impact: N/A

### Description

In the L1PoolManager.TransferAssertToArbitrumOneBridge() function, the protocol calls outboundTransferCustomRefund() for cross-chain transfers.

```
function TransferAssertToArbitrumOneBridge(
        address _token,
        address _to,
        uint256 _amount
    }internal {
        if (_token == address(ContractsAddress.ETHAddress)) {
            IArbitrumOneL1Bridge(ContractsAddress.ArbitrumOneL1GatewayRouter)
                .outboundTransferCustomRefund{value: _amount}(
                ContractsAddress.ETHAddress,
                address(this),
                _to,
                _amount,
                0,
                0,
                ""
            );
        ...
```

Reviewing the outboundTransferCustomRefund() function, we see that it calls super.outboundTransferCustomRefund().

https://github.com/OffchainLabs/arbitrum-classic/blob/551a39b381dcea81e03e7599fcb01fddff4fe96c/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ERC20Gateway.sol#L58

```
function outboundTransferCustomRefund(
        address _l1Token,
        address _refundTo,
        address _to,
        uint256 _amount,
        uint256 _maxGas,
        uint256 _gasPriceBid,
        bytes calldata _data
    ) public payable override nonReentrant returns (bytes memory res) {
        return
            super.outboundTransferCustomRefund(
                _l1Token,
                _refundTo,
                _to,
                _amount,
                _maxGas,
                _gasPriceBid,
                _data
            );
```

```
    }
```

In the L1GatewayRouter.outboundTransferCustomRefund() function, the documentation clearly states that _refundTo is an address on L2 used for refunding in case of a failure or excess funds.
https://github.com/OffchainLabs/token-bridge-contracts/blob/92c3caba883c057c41461162d1795723b1c35986/contracts/tokenbridge/ethereum/gateway/L1GatewayRouter.sol#L298

```
function outboundTransferCustomRefund(
        address _token,
        address _refundTo,
        address _to,
        uint256 _amount,
        uint256 _maxGas,
        uint256 _gasPriceBid,
        bytes calldata _data
    ) public payable override returns (bytes memory) {
        address gateway = getGateway(_token);
        bytes memory gatewayData = GatewayMessageHandler.encodeFromRouterToGateway(
            msg.sender,
            _data
        );

        emit TransferRouted(_token, msg.sender, _to, gateway);
        // here we use `IL1ArbitrumGateway` since we don't assume all ITokenGateway implements `outboundTransferCustomRefund`
        return
            IL1ArbitrumGateway(gateway).outboundTransferCustomRefund{ value: msg.value }(
                _token,
                _refundTo,
                _to,
                _amount,
                _maxGas,
                _gasPriceBid,
                gatewayData
            );
    }
```

However, in the protocol, the address passed is address(this), which on L2 is not controlled by the project but could be someone else's address.

```
IArbitrumOneL1Bridge(ContractsAddress.ArbitrumOneL1GatewayRouter)
                .outboundTransferCustomRefund{value: _amount}(
                ContractsAddress.ETHAddress,
                address(this),
                _to,
                _amount,
                0,
                0,
                ""
            );
```

This means that if there is a failure or excess funds on L2, the funds will be sent to this address, leading to a potential loss of funds for the protocol.

**Recommendation**

Use an address controlled by the project team to receive refund funds.

## 3.55 The IsCompleted check should be applied to staking ERC20 and ETH

- ID: SRS-055
- Severity: Medium
- Auditors: Alex
- Likelihood: N/A
- Impact: N/A

### Description

The L1PoolManager.DepositAndStakingWETH() function allows users to stake WETH and then receive fees. The protocol includes a check for Pools[address(ContractsAddress.WETH)][PoolIndex].IsCompleted.

```
uint256 PoolIndex = Pools[address(ContractsAddress.WETH)].length - 1;
        if (Pools[address(ContractsAddress.WETH)][PoolIndex].IsCompleted) {
            revert PoolIsCompleted(PoolIndex);
        }
```

However, this corresponding check is missing in the DepositAndStakingETH() and DepositAndStakingERC20() functions.

**Recommendation**

The IsCompleted check should be applied to staking ERC20 and ETH.

## 3.56 Users cannot bridge WETH from the ETH chain to the target chain

- ID: SRS-056
- Severity: Critical
- Auditors: Alex
- Likelihood: N/A
- Impact: N/A

### Description

The L1PoolManager inherits from the TokenBridgeBase contract.

The TokenBridgeBase.BridgeInitiateWETH() function allows users to bridge funds from the source chain to the target chain. In this function, the protocol first retrieves the L2 WETH address and then calls WETH.transferFrom() to transfer funds from the user, recording the amount in the FundingPoolBalance.

```
IWETH WETH = IWETH(L2WETH());

        uint256 BalanceBefore = WETH.balanceOf(address(this));
        WETH.transferFrom(msg.sender, address(this), value);
        uint256 BalanceAfter = WETH.balanceOf(address(this));
        uint256 amount = BalanceAfter - BalanceBefore;
        if (amount < MinTransferAmount) {
            revert LessThanMinTransferAmount(MinTransferAmount, amount);
        }
        FundingPoolBalance[ContractsAddress.WETH] += amount;
```

However, when users bridge funds from the ETH chain to the L2 chain, the protocol still retrieves the L2 WETH address. The L2 WETH might not exist on L1, preventing users from bridging WETH from the ETH chain to L2.

```
function L2WETH() public view returns (address) {
        uint256 Blockchain = block.chainid;
        if (Blockchain == 0x82750) {
            // Scroll: https://chainlist.org/chain/534352
            return (ContractsAddress.ScrollWETH);
        } else if (Blockchain == 0x44d) {
            // Polygon zkEVM https://chainlist.org/chain/1101
            return (ContractsAddress.PolygonZkEVMWETH);
        } else if (Blockchain == 0xa) {
            // OP Mainnet https://chainlist.org/chain/10
            return (ContractsAddress.OptimismWETH);
        } else if (Blockchain == 0xa4b1) {
            // Arbitrum One https://chainlist.org/chain/42161
            return (ContractsAddress.ArbitrumOneWETH);
        } else if (Blockchain == 0xa4ba) {
            // Arbitrum Nova https://chainlist.org/chain/42170
            return (ContractsAddress.ArbitrumNovaWETH);
```

**Recommendation**

When bridging WETH from the ETH chain to the L2 chain, the WETH address should be 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2.

## 3.57 Users cannot bridge WETH from the ETH chain to the target chain

- ID: SRS-057
- Severity: Medium

- Auditors: Alex
- Impact: N/A
- Likelihood: N/A

### Description

```
function WithdrawOrClaimBySimpleAsset(
    address _user,
    address _token,
    bool IsWithdraw
) internal {
    ...
    for (int256 i = 0; uint256(i) < Users[_user].length; i++) {
        ...
            Pools[_token][EndPoolId].TotalAmount -= Users[_user][index]
                .Amount;

            ...

            if (IsWithdraw) {
                Users[_user][index].isWithdrawed = true;
                SendAssertToUser(_token, _user, Amount);
                if (Users[_user].length > 1) {
                    Users[_user][index] = Users[_user][
                        Users[_user].length - 1
                    ];
                    Users[_user].pop();
                    i--;
                ...
                }
                }
                else {
                    Users[_user][index].StartPoolId = EndPoolId;
                    SendAssertToUser(_token, _user, Reward);
                ...
                }

        }
    }
}
```

If a user is doing only a claim instead of withdrawing, the Pools[_token][EndPoolId].TotalAmount -= Users[_user][index].Amount operation should not be executed. Otherwise would result in an inaccurate TotalAmount and incorrect reward calculation.

## 3.58 Missing isWithdrawed check

- ID: SRS-058
- Auditors: Dicanoex
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

### Description

```
function WithdrawOrClaimBySimpleID(
        address _user,
        uint index,
        bool IsWithdraw
    ) internal {
...
}
```

The function is missing the following check:

```
if (Users[_user][index].isWithdrawed) {
    continue;
}
```

### Recommendation

Although there is a swap-pop operation, it only applies when there is more than one item in Users[_user]. If there is only one item in Users[_user], the swap-pop operation won't be performed, leaving the current withdrawing item as is. User can use the only one item to withdraw infinitely, draining all funds in the pool.

Please add the isWithdrawed check as soon as possible.

## 3.59 Missing sourceChainId != destChainId check

- ID: SRS-059
- Auditors: Dicanoex
- Impact: N/A
- Severity: Low
- Likelihood: N/A

### Description

```
    function BridgeInitiateWETH(
        uint256 sourceChainId,
        uint256 destChainId,
```

```
        address to,
        uint256 value
    ) external returns (bool) {
        if (sourceChainId != block.chainid) {
            revert sourceChainIdError();
        }
        if (!IsSupportChainId(destChainId)) {
            revert ChainIdNotSupported(destChainId);
        }
        ...
    }
```

If sourceChainId == destChainId, the bridge is totally meaningless, and bad things may happen if the relayer does not arrange this correctly, resulting in a loss of funds.

**Recommendation**

Adding a sourceChainId != destChainId check is highly recommended.

## 3.60 Missing send value for TransferAssertToMantleBridge on ETH

- ID: SRS-060
- Auditors: Dicanoex
- Impact: N/A
- Severity: High
- Likelihood: N/A

**Description**

```
    function TransferAssertToMantleBridge(
        address _token,
        address _to,
        uint256 _amount
    )internal {
        if (_token == address(ContractsAddress.ETHAddress)) {
            IMantleL1Bridge(ContractsAddress.MantleL1Bridge).depositETHTo(
                _to,
                0,
                ""
            );
        } else {
            ...
        }
    }
```

The code is missing to send value to the bridge, resulting in a malfunctioning of the feature.

**Recommendation**

The code should be changed to:

```
    function TransferAssertToMantleBridge(
        address _token,
        address _to,
        uint256 _amount
    ) internal {
        if (_token == address(ContractsAddress.ETHAddress)) {
            IMantleL1Bridge(ContractsAddress.MantleL1Bridge).depositETHTo{value: _amount}(
                _to,
                0,
                ""
            );
        } else {
            ...
        }
    }
```

## 3.61 Missing payable for IMantleL1Bridge.depositETHTo

- ID: SRS-061
- Auditors: Dicanoex
- Impact: N/A
- Severity: Low
- Likelihood: N/A

**Description**

```
interface IMantleL1Bridge {
    function depositETHTo(
        address _to,
        uint32 _l2Gas,
        bytes calldata _data
    ) external;
    ...
}
```

The function is missing a payable keyword.

**Recommendation**

Add payable keyword.

## 3.62 Missing fee for TransferAssertToScrollBridge

- ID: SRS-062
- Auditors: Dicanoex
- Impact: N/A
- Severity: High
- Likelihood: N/A

### Description

```
1.  function TransferAssertToScrollBridge(
2.      address _token,
3.      address _to,
4.      uint256 _amount
5.  ) internal{
6.      if (_token == address(ContractsAddress.ETHAddress)) {
7.          uint fee = IL1MessageQueue(ContractsAddress.ScrollL1MessageQueue)
8.              .estimateCrossDomainMessageFee(170000);
9.          IScrollStandardL1ETHBridge(
10.             ContractsAddress.ScrollL1StandardETHBridge
11.         ).depositETH{value: _amount + fee}(_to, _amount, 170000);
12.     } else if (_token == address(ContractsAddress.WETH)) {
13.         uint fee = IL1MessageQueue(ContractsAddress.ScrollL1MessageQueue)
14.             .estimateCrossDomainMessageFee(20000);
15.         IERC20(_token).approve(
16.             ContractsAddress.ScrollL1StandardWETHBridge,
17.             _amount
18.         );
19.         IScrollStandardL1WETHBridge(
20.             ContractsAddress.ScrollL1StandardWETHBridge
21.         ).depositERC20(_token, _to, _amount, 20000);
22.     } else {
23.         uint fee = IL1MessageQueue(ContractsAddress.ScrollL1MessageQueue)
24.             .estimateCrossDomainMessageFee(20000);
25.         IERC20(_token).approve(
26.             ContractsAddress.ScrollL1StandardWETHBridge,
27.             _amount
28.         );
29.         IScrollStandardL1ERC20Bridge(
30.             ContractsAddress.ScrollL1StandardERC20Bridge
31.         ).depositERC20(_token, _to, _amount, 20000);
32.     }
33. }
```

Only the ETH case sets value to pay for the fee, while the other two cases do not send value, leading to a revert transaction according to the chain docs.

### Recommendation

Adding a value option is mandatory. In particular, change the code above to:

```
    function TransferAssertToScrollBridge(
        address _token,
        address _to,
        uint256 _amount
    ) internal {
        if (_token == address(ContractsAddress.ETHAddress)) {
            ...
        } else if (_token == address(ContractsAddress.WETH)) {
            ...
            IScrollStandardL1WETHBridge(
                ContractsAddress.ScrollL1StandardWETHBridge
            ).depositERC20{value: fee}(_token, _to, _amount, 20000);
        } else {
            ...
            IScrollStandardL1ERC20Bridge(
                ContractsAddress.ScrollL1StandardERC20Bridge
            ).depositERC20{value: fee}(_token, _to, _amount, 20000);
        }
    }
```

## 3.63 Missing length != 0 check for DepositAndStakingERC20/WETH

- ID: SRS-063
- Auditors: Dicanoex
- Impact: N/A
- Severity: High
- Likelihood: N/A

### Description

The DepositAndStakingETH function has a length != 0 check:

```
        if (Pools[address(ContractsAddress.ETHAddress)].length == 0) {
            revert NewPoolIsNotCreate(1);
        }
```

While DepositAndStakingERC20 and DepositAndStakingWETH do not. They do uint256 PoolIndex = Pools[].length - 1 directly. If the length equals 0, the subtraction will revert by Solidity underflow check, without a user-friendly error message.

**Recommendation**

Adding the check above to give a better user experience is recommended.

## 3.64 Incorrect usage of safeTransfer(From) for ERC20

- ID: SRS-064
- Auditors: Dicanoex
- Impact: N/A
- Severity: High
- Likelihood: N/A

### Description

```
1.      function BridgeInitiateERC20() external returns (bool) {
2.          ...
3.          IERC20(ERC20Address).safeTransferFrom(msg.sender, address(this), value);
4.          ...
5.      }
6.
7.      function BridgeFinalizeERC20() external onlyRole(ReLayer) returns (bool) {
8.          ..
9.          IERC20(ERC20Address).safeTransfer(to, amount);
10.         ...
11.     }
12.
13.     function SendAssertToUser() internal returns (bool) {
14.         ...
15.             if (IERC20(_token).balanceOf(address(this)) < _amount) {
16.                 revert NotEnoughToken(_token);
17.             }
18.             IERC20(_token).safeTransfer(to, _amount);
19.         }
20.         ...
21.     }
```

safeTransfer(From) is not a ERC20 standard function, which most ERC20 token contracts do not support. Executing such operations directly on the token contract results in a revert most time, making the protocol fail to work roughly.

**Recommendation**

The correct way should be employing the OpenZeppelin SafeERC20 util contract acting as a wrapper, which has safeTransfer(From) functions and accepts the real token contract address as its first parameter.