SALUS SECURITY

AUG 2025

# CODE SECURITY ASSESSMENT

RAINLINK

# Overview

## Project Summary

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

# Project Dashboard

## Application Summary

| | |
|---|---|
| **Name** | Rainlink - contract sol |
| **Version** | v2 |
| **Type** | Rust |
| **Dates** | Sep 09 2025 |
| **Logs** | Aug 28 2025; Sep 09 2025 |

## Vulnerability Summary

| | |
|---|---|
| **Total High-Severity issues** | 6 |
| **Total Medium-Severity issues** | 5 |
| **Total Low-Severity issues** | 5 |
| **Total informational issues** | 3 |
| **Total** | 19 |

## Contact

E-mail: support@salusec.io

# Risk Level Description

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

# Content

# Introduction

## 1.1 About SALUS

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

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

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

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

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

## 1.2 Audit Breakdown

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

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

## 1.3 Disclaimer

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

# Findings

## 2.1 Summary of Findings

| ID | Title | Severity | Category | Status |
|---|---|---|---|---|
| 1 | The withdraw instruction lacks access control | High | Business Logic | Resolved |
| 2 | The PoolNew() may fail if fund_pool ATA is pre-created | High | Denial of Service | Resolved |
| 3 | Messages can be replayed across different receivers | High | Business Logic | Resolved |
| 4 | Confirmfromotherchain() lacks a check to verify whether the message is invoked by the target program | High | Business Logic | Resolved |
| 5 | Mismatched denominator in reward | High | Business Logic | Resolved |
| 6 | LP fee may not actually be distributed due to rounding | High | Numerics | Resolved |
| 7 | The bridge_config initialization does not allocate space for the discriminator | Medium | Business Logic | Resolved |
| 8 | Off-by-one limit prevents full signer capacity in add_signers | Medium | Business Logic | Resolved |
| 9 | The remove_liquidity does not settle rewards in advance which may lead to potential losses | Medium | Business Logic | Resolved |
| 10 | Part of the lp_fee cannot be withdrawn | Medium | Business Logic | Resolved |
| 11 | Incorrect total_staked calculation | Medium | Business Logic | Resolved |
| 12 | The bridge_finish is missing a call to transfer_from_pool | Low | Business Logic | Resolved |
| 13 | Removing signers can drop the set below the threshold | Low | Business Logic | Resolved |
| 14 | The set_pool_fee_rate()does not check that the fee rate does not exceed 100 | Low | Business Logic | Resolved |
| 15 | Add access control for initsendtochainnonce | Low | Access Control | Resolved |
| 16 | The upload_fee may be used to evade the lp_fe | Low | Business Logic | Acknowledged |
| 17 | Redundant code | Informational | Redundancy | Acknowledged |
| 18 | The last_nonce is not assigned a value | Informational | Business Logic | Resolved |

| | | | | |
|---|---|---|---|---|
| 19 | Typo error | Informational | Code Quality | Resolved |

## 2.2 Notable Findings

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

### 1. The withdraw instruction lacks access control

| Severity: High | Category: Access Control |
| --- | --- |

Target:
- bridge-core\programs\bridge-core\src\instructions\message.rs

### Description

The Withdraw instruction withdraws accumulated cross-chain message fees from a PDA derived with `MESSAGE_FEE_SEED`, but the account validation does not bind the `super_admin` signer to any stored administrator.

As implemented, the accounts context only requires `super_admin: Signer<'info>` and a PDA `message_fee` validated by `seeds = [MESSAGE_FEE_SEED.as_bytes()], bump`, with no constraint such as `address = config.super_admin`, has_one, or an explicit runtime check.

bridge-core\programs\bridge-core\src\instructions\message.rs: L207-L215

```
#[derive(Accounts)]
pub struct Withdraw<'info> {
    #[account(mut)]
    pub super_admin: Signer<'info>,
    /// CHECK:
    #[account(mut, seeds = [MESSAGE_FEE_SEED.as_bytes()], bump)]
    pub message_fee: AccountInfo<'info>,
    pub system_program: Program<'info, System>,
}
```

If the handler transfers lamports from `message_fee` using invoke_signed, any arbitrary caller can present themselves as `super_admin` and trigger the withdrawal to their own address, effectively stealing all fees held by the PDA.

### Recommendation

It is recommended to check whether `super_admin` is the expected administrator address.

### Status

This issue has been resolved by the team with commit ef265c8.

## 2. The PoolNew() may fail if fund_pool ATA is pre-created

| Severity: High | Category: Denial of Service |
| --- | --- |

Target:
- bridge-token\programs\bridge-token\src\instructions\pool.rs

## Description

In the PoolNew instruction, the `fund_pool` account is expected to be initialized as the ATA of `bridge_authority`. Since ATAs are deterministic, a malicious actor can preemptively create this ATA, which causes the init constraint to fail and the instruction to revert. This makes pool creation vulnerable to a denial-of-service attack.

bridge-token\programs\bridge-token\src\instructions\pool.rs:

```
pub struct PoolNew<'info> {
    #[account(mut, constraint = payer.key() == bridge_config.admin || payer.key() ==
crate::ID)]
    pub payer: Signer<'info>,
    pub token_mint: Account<'info, Mint>,
    #[account(init, payer = payer, associated_token::mint = token_mint,
associated_token::authority = bridge_authority)]
    pub fund_pool: Box<Account<'info, TokenAccount>>,
    //...
}
```

## Recommendation

It is recommended to use `init_if_needed` to avoid the risk of DoS.

## Status

This issue has been resolved by the team with commit ef265c8.

## 3. Messages can be replayed across different receivers

| Severity: High | Category: Business Logic |
| --- | --- |

Target:
- bridge-core\programs\bridge-core\src\instructions\message.rs

## Description

During cross-chain message execution, each receiver maintains a `FromChainNonce` account for every `from_chain` to track the nonces of their own messages and prevent replay.
bridge-core\programs\bridge-core\src\instructions\message.rs: L242-L252

```
pub struct ConfirmFromOtherChain<'info> {
    #[account(init_if_needed, payer = user, seeds =
[&msg_header.from_chain.combain_chain(), FromChainNonce::SEED_SUFFIX.as_bytes(),
receiver.key().as_ref()], bump, space = 8 + FromChainNonce::LEN)]
    pub from_chain_nonce_account: Account<'info, FromChainNonce>,
    #[account(seeds = [ConfigInfo::SEEDS.as_bytes()], bump)]
    pub bridge_config: Account<'info, ConfigInfo>,
    #[account(mut)]
    pub user: Signer<'info>,
    /// CHECK:
    pub receiver: AccountInfo<'info>,
    pub system_program: Program<'info, System>,
}
```

However, in the `ConfirmFromOtherChain` instruction, there is no check to ensure that the receiver matches the `to_who` field in the message body, and anyone can call this function.

bridge-core\programs\bridge-core\src\state\config.rs:L132-L159

```
pub fn check_and_store_nonce(&mut self, nonce: u64) -> Result<bool> {
    //...
    let min_nonce = *map.keys().next().ok_or(crate::error::ErrorCode::NonceInvalid)?;

    if nonce < min_nonce {
        return Err(crate::error::ErrorCode::NonceInvalid.into());
    }
    //...
}
```

As a result, a malicious actor can execute high-nonce messages belonging to other receivers on any receiver, causing a DoS on the receiver's pending low-nonce messages.

For example:

1. user1 sends a cross-chain message with nonce = 10, and user2 sends a cross-chain message with nonce = 11.
2. A malicious actor submits user2's cross-chain message and signature early, using user1's FromChainNonce account. user1's min_nonce is now updated to 11, causing their pending cross-chain message to never be successfully executed.

## Recommendation

It is recommended to check whether the message matches the receiver.

## Status

This issue has been resolved by the team with commit ef265c8.

## 4. Confirmfromotherchain() lacks a check to verify whether the message is invoked by the target program

| Severity: High | Category: Business Logic |
| --- | --- |

Target:
- bridge-core\programs\bridge-core\src\instructions\message.rs:L242-L252

## Description

When executing the `confirm_message` instruction, there is no check to ensure that the message is invoked by the `caller_auth_pda` of the target program. This allows an attacker to frontrun the execution and consume the nonce, causing the target program's subsequent invocation of this instruction to revert because the nonce has already been used.

bridge-core\programs\bridge-core\src\instructions\message.rs:L242-L252

```
pub struct ConfirmFromOtherChain<'info> {
    #[account(init_if_needed, payer = user, seeds =
[&msg_header.from_chain.combain_chain(), FromChainNonce::SEED_SUFFIX.as_bytes(),
receiver.key().as_ref()], bump, space = 8 + FromChainNonce::LEN)]
    pub from_chain_nonce_account: Account<'info, FromChainNonce>,
    #[account(seeds = [ConfigInfo::SEEDS.as_bytes()], bump)]
    pub bridge_config: Account<'info, ConfigInfo>,
    #[account(mut)]
    pub user: Signer<'info>,
    /// CHECK:
    pub receiver: AccountInfo<'info>,
    pub system_program: Program<'info, System>,
}
```

## Recommendation

In fact, this check is already implemented in the bridge-token program, but we should move the check into the `confirm_message` instruction itself to prevent frontrunning.

## Status

This issue has been resolved by the team with commit ec9d117.

## 5. Mismatched denominator in reward

| Severity: High | Category: Business Logic |
| --- | --- |

Target:
- bridge-token\programs\bridge-token\src\state\pool.rs

## Description

The `refresh_rewards` function calculates `acc_ratio` by dividing `pool_fee` by `total_liquidity`. However, `total_liquidity` includes both user-added liquidity (via `add_liquidity`) and bridged tokens, while `pool_fee` is derived only from staked assets without considering bridged liquidity. This mismatch causes the rewards to be distributed incorrectly, as the denominator includes assets that are not contributing to fee generation. The same issue occurs in the calculation of `last_apy`, where division is also performed against `total_liquidity` instead of the actual staked liquidity.

bridge-token\programs\bridge-token\src\state\pool.rs: L46-L58

```
pub fn refresh_rewards(&mut self, balance: u64, lp_fee: u64) -> Result<()> {
    self.total_liquidity = balance;

    let pool_fee = lp_fee * self.total_staked / self.total_liquidity;
    let now_seconds = Clock::get()?.unix_timestamp;
    let delta = now_seconds - self.last_receive_rewards_time;
    self.total_earns += pool_fee;
    self.acc_ratio += pool_fee as f64 / self.total_liquidity as f64;
    self.last_apy = pool_fee as f64 / self.total_liquidity as f64 * (365 * 24 * 60 * 60)
as f64
        / delta as f64;
    self.last_receive_rewards_time = now_seconds;
    Ok(())
}
```

## Recommendation

It is recommended to use `total_staked_liquidity` instead of `total_liquidity` for calculation.

## Status

This issue has been resolved by the team with commit ef265c8.

## 6. LP fee may not actually be distributed due to rounding

| Severity: High | Category: Numerics |
| --- | --- |
| Target:<br>- bridge-token\programs\bridge-token\src\state\pool.rs: L46-L58 | |

### Description

Each time a cross-chain message is processed, an `lp_fee` is intended to be distributed pro-rata to liquidity providers by increasing a global accumulator. The update currently adds `pool_fee / total_liquidity` to `acc_ratio`:

bridge-token\programs\bridge-token\src\state\pool.rs: L46-L58

```
pub fn refresh_rewards(&mut self, balance: u64, lp_fee: u64) -> Result<()> {
    //...
    self.acc_ratio += pool_fee as f64 / self.total_liquidity as f64;
    //...
}
```

When `lp_fee` is small and `total_liquidity` is large, the increment becomes extremely tiny. Downstream, user entitlements are derived from `user_stake * Δacc_ratio` and then converted to an integer token amount. Because that multiplication is followed by truncation, the per-message increment often rounds down to zero, so no fee is credited for that step. In practice this causes systematic dust loss and delayed or permanently missing distributions.

### Recommendation

It is recommended to multiply by a scaling factor to increase the magnitude of the `lp_fee`.

### Status

This issue has been resolved by the team with commit ef265c8.

## 7. The bridge_config initialization does not allocate space for the discriminator

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- bridge-core\programs\bridge-core\src\instructions\config.rs | |

## Description

When initializing an Anchor account, an additional 8 bytes of space must be allocated for the discriminator, but this allocation was not made during the initialization of `bridge_config`.

bridge-core\programs\bridge-core\src\instructions\config.rs: L20-L29

```
pub struct ConfInitialize<'info> {
    #[account(init, payer = authority, seeds = [ConfigInfo::SEEDS.as_bytes()], bump,
space = ConfigInfo::LEN)]
    pub bridge_config: Account<'info, ConfigInfo>,
    //...
}
```

This results in a maximum of only 11 signers being added, and attempting to add the 12th signer will fail due to insufficient account space.

## Recommendation

Allocate the space as `space = 8 + ConfigInfo::LEN`.

## Status

This issue has been resolved by the team with commit ef265c8.

## 8. Off-by-one limit prevents full signer capacity in add_signers

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- bridge-core\programs\bridge-core\src\instructions\validator.rs | |

## Description

The `bridge_config` allocates space for 12 signer addresses, but because the `add_signers` function prevents the number of signers from reaching 12, only 11 addresses can actually be added.

bridge-core\programs\bridge-core\src\instructions\validator.rs: L6-L15

```
pub fn add_signers(ctx: Context<BridgeConf>, new_signers: Vec<[u8; 20]>) -> Result<()> {
    let bridge_config = &mut ctx.accounts.bridge_config;
    for signer in new_signers {
        if !bridge_config.signers.contains(&signer) {
            bridge_config.signers.push(signer);
        }
    }
    require!(bridge_config.signers.len() < 12,
crate::error::ErrorCode::ValidatorOver12);
    Ok(())
}
```

## Recommendation

Consider fixing this issue to allow correctly adding 12 signers.

## Status

This issue has been resolved by the team with commit ef265c8.

## 9. The remove_liquidity does not settle rewards in advance which may lead to potential losses

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- bridge-token\programs\bridge-token\src\instructions\pool.rs | |

## Description

The `remove_liquidity` function does not settle unclaimed rewards, which may result in reward loss.

bridge-token\programs\bridge-token\src\instructions\pool.rs: L61-L70

```
pub fn remove_liquidity(ctx: Context<PoolLiquidity>, amount: u64) -> Result<()> {
    // Change lp
    let lp_account = &mut ctx.accounts.lp_account;
    if lp_account.amount < amount {
        return Err(crate::error::ErrorCode::Illiquidity.into());
    }
    let old_amount = lp_account.amount;
    lp_account.amount -= amount;
    lp_account.debt -=
        (lp_account.debt as f64 * (lp_account.amount as f64 / old_amount as f64)) as
u64;
}
```

In the example, a user with amount=100, debt=100, and `acc_ratio` growing to 1.1 has a 10 token entitlement. Withdrawing 50 before claiming sets amount=50 and debt=50, so the later claim produces only 50×1.1−50 = 5, permanently losing 5 tokens.

## Recommendation

It is recommended to properly handle unclaimed rewards.

## Status

This issue has been resolved by the team with commit ef265c8.

## 10. Part of the lp_fee cannot be withdrawn

| Severity: Medium | Category: Business Logic |
| --- | --- |

Target:
- bridge-token\programs\bridge-token\src\state\pool.rs

## Description

The `lp_fee` is proportionally divided into `pool_fee` and distributed to liquidity providers, but the remaining portion is not handled and cannot be withdrawn or used by anyone.

bridge-token\programs\bridge-token\src\state\pool.rs: L46-L58

```
pub fn refresh_rewards(&mut self, balance: u64, lp_fee: u64) -> Result<()> {
    self.total_liquidity = balance;

    let pool_fee = lp_fee * self.total_staked / self.total_liquidity;
    let now_seconds = Clock::get()?.unix_timestamp;
    let delta = now_seconds - self.last_receive_rewards_time;
    self.total_earns += pool_fee;
    self.acc_ratio += pool_fee as f64 / self.total_liquidity as f64;
    self.last_apy = pool_fee as f64 / self.total_liquidity as f64 * (365 * 24 * 60 * 60)
as f64
        / delta as f64;
    self.last_receive_rewards_time = now_seconds;
    Ok(())
}
```

## Recommendation

It is recommended to handle the unclaimable portion of the `lp_fee`.

## Status

This issue has been resolved by the team with commit ef265c8.

## 11. Incorrect total_staked calculation

| Severity: Medium | Category: Business Logic |
| --- | --- |

Target:
- bridge-token\programs\bridge-token\src\instructions\pool.rs

## Description

`total_staked` is used to represent the total stake of all stakers. In the design, bridged tokens are partially paid by the pool's liquidity providers and partially by the incoming bridged funds. When distributing fees, the allocation should be proportional to `total_staked`. However, this part is not actually implemented in the contract, which causes `total_staked` not to decrease during bridging, resulting in an over-allocation of fees.

Additionally, during the `remove_liquidity` process, there is a logical issue in the calculation of `staked_decrease`, causing it to not correctly reduce `total_staked`.

bridge-token\programs\bridge-token\src\instructions\pool.rs: L61-L70

```
pub fn remove_liquidity(ctx: Context<PoolLiquidity>, amount: u64) -> Result<()> {
    ...
    let staked_decrease = amount * old_amount / pool_account.total_staked_liquidity;
    pool_account.total_staked -= staked_decrease;
    pool_account.total_staked_liquidity -= amount;
    pool_account.total_liquidity -= amount;
    ...
}
```

## Recommendation

It is recommended to properly handle unclaimed rewards.

## Status

This issue has been resolved by the team with commit ef265c8.

## 12. The bridge_finish is missing a call to transfer_from_pool

| Severity: Low | Category: Business Logic |
| --- | --- |

Target:
- bridge-token\programs\bridge-token\src\instructions\executor.rs

## Description

When the bridged funds are transferred from the pool to another chain, the `transfer_to_pool` function is called to increase the `platform_vault`.
When the bridged funds are transferred out of the pool, only the transfer instruction is called to move the funds to the receiver, but the `transfer_from_pool` call to update the pool's recorded state is missing, resulting in the `platform_vault` not being properly decreased.

bridge-token\programs\bridge-token\src\instructions\executor.rs: L61-L70

```
pub fn bridge_finish(
    ctx: Context<Consumption>,
    msg_header: MsgHeader,
    msg_body: MsgBody,
    accum_pk: Vec<u8>,
    signatures: Vec<[u8; 65]>,
) -> Result<()> {
    ...
    if ctx.accounts.token_relation.mint_type == MintType::Mint as u8 {
        ...
    } else {
        ...
        let cpi_ctx = CpiContext::new_with_signer(
            ctx.accounts.token_program.to_account_info(),
            anchor_spl::token::Transfer {
                from: ctx.accounts.fund_pool.to_account_info(),
                to: ctx.accounts.receiver_token_account.to_account_info(),
                authority: ctx.accounts.bridge_authority.to_account_info(),
            },
            signer_seeds,
        );
        token::transfer(cpi_ctx, final_amount as u64)?;
        ...
```

## Recommendation

It is recommended to call `transfer_from_pool` when transferring funds out of the pool.

## Status

This issue has been resolved by the team with commit ef265c8.

## 13. Removing signers can drop the set below the threshold

| Severity: Low | Category: Business Logic |
| --- | --- |
| Target:<br>- bridge-core\programs\bridge-core\src\instructions\validator.rs | |

## Description

The `remove_signers`(ctx, `signers_to_remove`) instruction deletes listed signers from `bridge_config.signers` without validating the post-update quorum. If the remaining signer count falls below `bridge_config.threshold`, any operation that requires threshold signatures becomes unexecutable until governance adds new signers or lowers the threshold.

bridge-core\programs\bridge-core\src\instructions\validator.rs: L17-L24

```
pub fn remove_signers(ctx: Context<BridgeConf>, signers_to_remove: Vec<[u8; 20]>) ->
Result<()> {
    let bridge_config = &mut ctx.accounts.bridge_config;
    bridge_config
        .signers
        .retain(|signer| !signers_to_remove.contains(signer));

    Ok(())
}
```

## Recommendation

It is recommended to add checks to ensure that the threshold is not exceeded.

## Status

This issue has been resolved by the team with commit ec9d117.

## 14. The set_pool_fee_rate()does not check that the fee rate does not exceed 100

| Severity: Low | Category: Business Logic |
| --- | --- |
| Target:<br>- bridge-token\programs\bridge-token\src\instructions\pool.rs | |

## Description

The `pool_fee_rate` cannot exceed 1,000,000; otherwise, the fee rate will exceed 100%, causing an underflow and making `bridge_finish` fail.

bridge-token\programs\bridge-token\src\instructions\pool.rs: L26-L30

```
pub fn set_pool_fee_rate(ctx: Context<PoolFeeRate>, fee_rate: u64) -> Result<()> {
    let pool_account = &mut ctx.accounts.pool_account;
    pool_account.pool_fee_rate = fee_rate;
    Ok(())
}
```

## Recommendation

It is recommended to add this check.

## Status

This issue has been resolved by the team with commit ef265c8.

## 15. Add access control for initsendtochainnonce

| Severity: Low | Category: Access Control |
| --- | --- |
| Target:<br>- bridge-core\programs\bridge-core\src\instructions\message.rs | |

## Description

The `ToChainNonce` account actually represents which chains are supported, but anyone can initialize it. This can cause users to mistakenly call `send_message` to an unsupported chain.

bridge-core\programs\bridge-core\src\instructions\message.rs: L200-L208

```
#[derive(Accounts)]
#[instruction(to_chain: Chain)]
pub struct InitSendToChainNonce<'info> {
    #[account(mut)]
    pub sender: Signer<'info>,
    #[account(init, payer = sender, seeds = [&to_chain.combain_chain(),
ToChainNonce::SEED_SUFFIX.as_bytes()], bump, space = 8 + ToChainNonce::LEN)]
    pub to_chain_nonce_account: Account<'info, ToChainNonce>,
    pub system_program: Program<'info, System>,
}
```

## Recommendation

It is recommended to add access control, allowing the admin to add supported chains to prevent users from mistakenly sending messages to unsupported chains.

## Status

This issue has been resolved by the team with commit ef265c8.

## 16. The upload_fee may be used to evade the lp_fee

| Severity: Low | Category: Business Logic |
| --- | --- |
| Target:<br>- bridge-token\programs\bridge-token\src\instructions\executor.rs | |

## Description

The `upload_fee` is intended to incentivize and compensate for the execution of messages on the target chain. However, since the `upload_fee` does not require paying the `lp_fee`, it could be used to bridge tokens to the target chain and evade the `lp_fee`, provided that the token recipient's MEV succeeds; otherwise, these tokens would be sent to the message executor.

bridge-token\programs\bridge-token\src\instructions\executor.rs: L245-265

```
else {
    let from_decimals = ctx.accounts.fee_token_relation.from_decimals;
    let to_decimals = ctx.accounts.fee_token_relation.to_decimals;
    let gas_fee = msg_header.upload_gas_fee * 10u128.pow(to_decimals.into())
        / 10u128.pow(from_decimals.into());
    let cpi_ctx = CpiContext::new_with_signer(
        ctx.accounts.token_program.to_account_info(),
        anchor_spl::token::Transfer {
            from: ctx.accounts.fee_fund_pool.to_account_info(),
            to: ctx.accounts.sender_fee_token_account.to_account_info(),
            authority: ctx.accounts.bridge_authority.to_account_info(),
        },
        signer_seeds,
    );
    token::transfer(cpi_ctx, gas_fee as u64)?;

    // transfer gas fee from pool
    ctx.accounts
        .fee_pool_account
        .transfer_from_pool(gas_fee as i64)?;
}
```

## Recommendation

It is recommended to set a maximum `upload_fee` to prevent this situation.

## Status

This issue has been acknowledged by the team.

# 2.3 Informational Findings

## 17. Redundant code

| Severity: Informational | Category: Redundancy |
| --- | --- |
| Target:<br>- bridge-token\programs\bridge-token\src\instructions\config.rs | |

### Description

1. In the `TokenRelationshipRemove` instruction, the close constraint is used, which closes the `token_relation` account and refunds the remaining lamports to the admin. However, the `token_relationship_remove` function also transfers lamports again, which is a redundant operation.

bridge-token\programs\bridge-token\src\instructions\config.rs: L41-L51

```
pub fn token_relationship_remove(
    ctx: Context<TokenRelationshipRemove>,
    _from_chain: Chain,
    _from_token: [u8; 32],
) -> Result<()> {
    //...
    **ctx.accounts.admin.to_account_info().lamports.borrow_mut() +=
token_relation.to_account_info().lamports();
    **token_relation.to_account_info().lamports.borrow_mut() = 0;
    Ok(())
}
```

2. In the `bridge_proposal` function, tokens are currently transferred from the user's tokenAccount to the `bridge_authority` and then burned. They can actually be burned directly from the user's tokenAccount.

bridge-token\programs\bridge-token\src\instructions\executor.rs: L18-82

```
pub fn bridge_proposal(
    ctx: Context<Proposal>,
    to_chain: Chain,
    _to_token: [u8; 32],
    to_who: [u8; 32],
    all_amount: u64,
    upload_gas_fee: u64,
) -> Result<()> {
    //...
    if token_relation.mint_type == MintType::Mint as u8 {
        anchor_spl::token::transfer(
            CpiContext::new(
                ctx.accounts.token_program.to_account_info(),
                anchor_spl::token::Transfer {
                    from: ctx.accounts.sender_token.to_account_info(),
                    to: ctx.accounts.fund_pool.to_account_info(),
                    authority: ctx.accounts.sender.to_account_info(),
                },
            ),
            all_amount,
        )?;
```

```
        anchor_spl::token::burn(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info(),
                anchor_spl::token::Burn {
                    mint: ctx.accounts.token_mint.to_account_info(),
                    from: ctx.accounts.fund_pool.to_account_info(),
                    authority: ctx.accounts.bridge_authority.to_account_info(),
                },
                signer_seeds,
            ),
            all_amount,
        )?;
    }
```

## Recommendation

Consider removing the redundant code.

## Status

This issue has been acknowledged by the team.

## 18. The last_nonce is not assigned a value

| Severity: Informational | Category: Business Logic |
| --- | --- |
| Target:<br>- bridge-core\programs\bridge-core\src\instructions\message.rs | |

## Description

The `last_nonce` field is used to check initialization, initialization events, and to preemptively verify whether a nonce has been reused. However, the function does not update the `last_nonce` field, causing it to remain 0.

bridge-core\programs\bridge-core\src\instructions\message.rs: L114-L153

```
/// Confirm a message of another chain
pub fn confirm_message(
    ctx: Context<ConfirmFromOtherChain>,
    msg_header: MsgHeader,
    msg_body: Vec<u8>,
    _accum_pk: Vec<u8>,
    signatures: Vec<[u8; 65]>,
) -> Result<()> {
    //...

    // update message nonce
    let from_chain_nonce_account = &mut ctx.accounts.from_chain_nonce_account;
    if from_chain_nonce_account.last_nonce == 0 {
        from_chain_nonce_account.chain = msg_header.from_chain.clone();
    }

    msg!(
        "msg_header.nonce {}, from_chain_nonce_account.last_nonce {}",
        msg_header.nonce,
        from_chain_nonce_account.last_nonce
    );


    // check if the message is already confirmed
    require!(
        from_chain_nonce_account.last_nonce != msg_header.nonce,
        crate::error::ErrorCode::NonceConsumed
    );
```

## Recommendation

Consider updating the `last_nonce` field.

## Status

This issue has been resolved by the team with commit ef265c8.

## 19. Typo error

| Severity: Informational | Category: Code Quality |
| --- | --- |

Target:
- bridge-token\programs\bridge-token\src\lib.rs
- bridge-token\programs\bridge-token\src\instructions\config.rs
- bridge-token\programs\bridge-token\src\instructions\pool.rs

## Description

1. Chian should be chain.

bridge-token\programs\bridge-token\src\lib.rs: L55

```
ctx: Context<ChianRelationship>
```

bridge-token\programs\bridge-token\src\instructions\config.rs:L54,L123

```
ctx: Context<ChianRelationship>
```

```
pub struct ChianRelationship<'info> {
```

2. Reword should be reward

bridge-token\programs\bridge-token\src\instructions\pool.rs: L102-105, L140

```
let reword = (lp_account.amount as f64 * pool_account.acc_ratio) as u64 -
lp_account.debt
    + lp_account.remaining;
require!(
    reword >= amount,
    crate::error::ErrorCode::WithdrawalAmountExceedReword
);
```

```
lp_account.remaining = reword - amount;
```

## Recommendation

Consider fixing the typos.

## Status

This issue has been resolved by the team with commit ef265c8.

# Appendix

## Appendix 1 - Files in Scope

This audit covered the following files in commit 53a0ad5

| File | SHA-1 hash |
|---|---|
| bridge-core/programs/bridge-core/src/instructions/config.rs | 45b437fcf38c0b6fb688f54626b896a530b21095 |
| bridge-core/programs/bridge-core/src/instructions/message.rs | 2424a808d7b7472ac0eee8bc889ba3b45f055df2 |
| bridge-core/programs/bridge-core/src/instructions/mod.rs | 9015200f0c4d3715375f176f9bff97c422fa545a |
| bridge-core/programs/bridge-core/src/instructions/validator.rs | d252cc466c31c78bbbf76e5ad237726dbfb75f33 |
| bridge-core/programs/bridge-core/src/state/config.rs | 2a2350611c6182a5a93125f5f5a154a858717f62 |
| bridge-core/programs/bridge-core/src/state/mod.rs | 22e34b1c5315894c6ef688a998f61a78f227c66a |
| bridge-core/programs/bridge-core/src/utils/mod.rs | 07acb7c48c8a7aeb57e911f611bf2a519f60809f |
| bridge-core/programs/bridge-core/src/utils/sigverify.rs | 06b107e28a3abce5c85b8e3e0f8e44d6bb666602 |
| bridge-token/programs/bridge-token/src/instructions/config.rs | ce16fe3d270b08c1d29f9878db60bc926c628b6c |
| bridge-token/programs/bridge-token/src/instructions/executor.rs | 876aeae653ea6364461e79b95fef2d29318d7b9c |
| bridge-token/programs/bridge-token/src/instructions/mod.rs | e0e4c92857e167911cd5a1eb8f0647c1412b2e3d |
| bridge-token/programs/bridge-token/src/instructions/pool.rs | 8b22fb6467ad3eff23aea67359b2f1de287d99a6 |
| bridge-token/programs/bridge-token/src/instructions/token.rs | 5fd73a6f10fdf69cffa08664aa2d0d57946e7d8a |
| bridge-token/programs/bridge-token/src/state/config.rs | 44753b2cea180a14c163367c21d3f67926ece0b8 |
| bridge-token/programs/bridge-token/src/state/executor.rs | 390978ca44938bc30ed0eadc085beccdf878792b |
| bridge-token/programs/bridge-token/src/state/mod.rs | 8370a56c836e729f3138759c4b4bd6c731d69aef |
| bridge-token/programs/bridge-token/src/state/pool.rs | 29883c22ca711553840a092298251503c8bad082 |