QuillAudits

AUDIT REPORT

November 2025

For

fly

Table of Content

Executive Summary

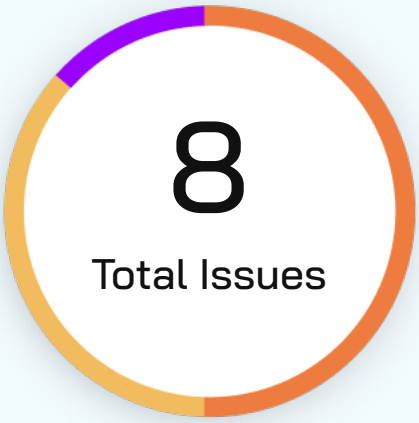
Project Name	Fly Trade
Protocol Type	Defi - Cross chain Swap
Project URL	https://www.fly.trade/
Overview	FlyTrade Protocol implement a decentralized cross-chain swapping and delegation framework.It define core functionalities such as contract initialization, user whitelisting, and secure delegation of operations.The swap module manages token swaps across chains, while registry handles asset and participant registrations.relayers coordinate and validate cross-chain transactions, ensuring reliable execution. Supporting modules like close enable controlled contract termination and settlement processes. Overall, the codebase establishes a modular, secure, and scalable foundation for multi-chain DeFi operations.
Audit Scope	The scope of this Audit was to analyze the Fly Trade Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/magpieprotocol/magpie-svm-contracts
Branch	main
Contracts in Scope	src/*
Commit Hash	e229ce8f4f238130f5a9a2b442b99a9a999e53b9
Language	Rust
Blockchain	Solana
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	10th October 2025 - 23rd October 2025
Updated Code Received	6th November 2025
Review 2	6th November 2025 - 8th November 2025

Review 3	On 10th November 2025, Fly Trade Team added pause functionality, and made config management more flexible at pull15 https://github.com/magpieprotocol/magpie-svm-contracts/pull/15/files QuillAudits reviewed these changes on 14th November 2025 and confirmed that everything is in good order, with no new issues identified.
Fixed In	https://github.com/magpieprotocol/magpie-svm-contracts **Commit hash : 563874be30d4872ad9ab76d35d0fdcdecbae77ab**

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



Critical	0 (0%)
High	0 (0%)
Medium	4 (50%)
Low	3 (37.5%)
Informational	1 (12.5%)

Severity

Issues	Critical	High	Medium	Low	Informational
Open	0	0	0	0	0
Acknowledged	0	0	**2**	**3**	**1**
Partially Resolved	0	0	0	0	0
Resolved	0	0	**2**	0	0

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Users Can Bypass Swap Fees Using is_bridge_call Parameter	Medium	Resolved
2	Users can drain the protocol vault by exploiting how token accounts are initialized, closed, and validated.	Medium	Resolved
3	Fee receivers can be manipulated by users to redirect protocol revenue	Medium	Acknowledged
4	Swap fee can be manipulated by users	Medium	Acknowledged
5	Swap fee can be manipulated by users	Low	Acknowledged
6	Registry scalability limitation due to u8 index type	Low	Acknowledged
7	Missing emergency-stop / pausable mechanism in magpie-router program	Low	Acknowledged
8	Missing Validation for Token Account and Mint Correspondence	Informational	Acknowledged

Checked Vulnerabilities

We have scanned the solana program for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Signer authorization
- Account data matching
- Sysvar address checking
- Owner checks
- Type cosplay
- Initialization
- Arbitrary cpi
- Duplicate mutable accounts
- Bump seed canonicalization
- PDA Sharing
- Incorrect closing accounts
- Missing rent exemption checks
- Arithmetic overflows/underflows
- Numerical precision errors
- Solana account confusions
- Casting truncation
- Insufficient SPL token account verification
- Signed invocation of unverified programs

Techniques and Methods

Throughout the audit of Solana Programs, care was taken to ensure:

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Severity Matrix

Impact

Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Medium Severity Issues

Users Can Bypass Swap Fees Using is_bridge_call Parameter

Resolved

Path

programs/magpie-router/src/instructions

Function Name

`swap.rs`

Description

The swap function accepts a user-supplied boolean parameter called is_bridge_call which, when set to true, allows users to completely bypass swap fees. There is no validation to ensure that only legitimate bridge calls can use this parameter.
Looking at the swap function, When is_bridge_call is set to true, the function skips the call to transfer_swap_fees(), allowing users to bypass all fee payments. This parameter is directly exposed in the program's public interface.

```
1
2
3  pub fn swap<'a, 'b, 'c: 'info, 'info>(
4      ctx: Context<'a, 'b, 'c, 'info, Swap<'info>>,
5      calldata: Vec<u8>,
6      is_bridge_call: bool,
7  ) -> Result<u64> {
8      let amount_out = instructions::swap(ctx, calldata, is_bridge_call)?;
9
10     Ok(amount_out)
11 }
```

Any user can pass is_bridge_call: true to completely avoid the swap fees

Impact

This allows any user to perform swaps without paying the required fees

Recommendation

Removethe is_bridge_call parameter entirely if it's not needed & If bridge functionality is required, implement proper authorization to make sure the call is indeed a bridge call internally and not user facing, just like how we have implemented to decide if a transaction is gasless or not, in the same way we should implement to decide whether its a bridge call or not, rather than taking it from user as input

Users can drain the protocol vault by exploiting how token accounts are initialized, closed, and validated.

Resolved

Path

programs/magpie-router/src/instructions

Function Name

`Swap, Init_multiple_token_accounts, validate_token_accounts_post_state`

Description

The attack flow is:
1. Include multiple token accounts in remaining accounts & swap data with owned_by_initiator = true
2. Have the vault pay to initialize these accounts
3. Use program's CPI to close them during execution
4. Collect rent refunds(inside cpi)
5. post cpi check is flawed and hence anyone can get away with it

Inside init_multiple_token_accounts we can see that vault pays for creation and initialization of token accounts of swap_data.token_acccounts

1. Vault pays for user-owned token accounts:

```
1
2  ...```rust
3  // In init_multiple_token_accounts function
4  let authority = if token_account.owned_by_initiator {
5      swap_accounts.from()  // User is authority
6  } else {
7      swap_accounts.vault()
8  };
9
10
11 init_associated_token_account(
12     &mint,
13     &authority,
14     &swap_accounts.vault(),  //@audit why vault everytime?
15     &token_account_to_init.to_account_info(),
16     swap_accounts,
17 )?;
```

Once these accounts are initialized we validate these accounts inside `validate_swap_accounts` , in which the final if block is allowing the token accounts whose owner is not vault, if the authority is not vault we are not throwing error, if authority is some other wallet than vault, it lets it pass

```rust
        // Only token accounts mentioned in the swap_data are allowed
        if token_account::is_owned_by_token_program(account) {
            let authority = token_account::get_authority(account);
            if authority.is_some()

            //currently we can pass accounts whose authority is not vault.
                && authority.unwrap().eq(&swap_accounts.vault().key())
                && !token_account_keys.contains(&account.key())
            {
                return err!(ErrorCode::InvalidAccount);
            }
        }
```

so basically we can pass random token accounts of ours for random tokens and let vault pay for its initialization and then those rent sol(paid by vault) can be farmed by utilizing cpi, we can cpi into token program, have this new formed account closed and the authority(from addresss, not vault) would receive the rent back, this way we could farm rent sol for so many accounts for so many tokens. we can do so because the post validation of accounts is as follow inside:

2. post validation is flawed inside validate_token_accounts_post_state we are doing

```

for token_account_key in token_account_keys {
    let token_account_info = token_accounts_map.get(token_account_key).unwrap();
    let token_account = token_account_info.get_token_account(swap_accounts).unwrap();
    let balance = spl_asset::balance(&token_account)?;
    if balance < token_account_info.balance {
        return err!(ErrorCode::TokenAccountBalanceChanged);
    }

    let current_authority = get_authority(&token_account);
    //@audit even when is none, throw an error
    if current_authority.is_some()
        && !current_authority.unwrap().eq(&swap_accounts.vault().key())
    {
        return err!(ErrorCode::TokenAccountAuthorityChanged);
    }
}
```

so if we closed that newly formed token account, the authority would be none hence the execution won't throw an error, it only throws an error why authority is some and is not vault, but since we have closed our token accounts in cpi, the authority comes out as none and we don't get error back and have successfully farmed rent sol from those random accounts supplied in remaining accounts whose initialization is done via vault.

Impact

Attackers can repeatedly:
- Create transactions with many user-owned token accounts
- Have the vault pay for initialization (~0.002 SOL per account)
- Close accounts via CPI
- Collect rent refunds

This drains the vault's SOL balance over time, with each account closure netting about 0.002 SOL.

Remediation

Inside validate_token_accounts_post_state change the check to this:

```
1
2       let current_authority = get_authority(&token_account);
3   if current_authority.is_none() || !current_authority.unwrap().eq(&swap_accounts.vault().key())
4   {
5       return err!(ErrorCode::TokenAccountAuthorityChanged);
6   }
```

Fee receivers can be manipulated by users to redirect protocol revenue

Acknowledged

Path

programs/magpie-router/src/instructions

Function Name

`swap`

Description

Users can provide their own addresses as fee receivers in swap transactions, redirecting protocol fees and slippage revenue to accounts they control. This happens because fee receiver information is parsed directly from user-provided calldata without validation.

Whenexamining the SwapData parsing logic in swap.rs, we can see that swap_fee_receivers and slippage_receivers are directly extracted from user input:

```
pub fn get_swap_data(
    calldata: &[u8],
    is_from_asset_native: bool,
    is_to_asset_native: bool,
) -> Result<SwapData> {
    // ...other parsing code...
        match ratio_type {
            RatioType::SwapFee => {
                swap_fee_ratios.push(ratio);
                swap_fee_receivers.push(receiver_index); // User-controlled
            }
            RatioType::Slippage => {
                slippage_ratios.push(ratio);
                slippage_receivers.push(receiver_index); // User-controlled
            }
        }
    }
    // ...
```

These values are later used in get_amounts_and_receivers to calculate fee distributions, a malicious user can simply pass his own accounts as receivers

Impact

Malicious users can divert protocol fees and slippage fees to themselves instead of legitimate protocol fee receivers and slippage fees receivers

Remediation

Create a global config, where we can store 2 arrays of authorized protocol fee receivers and slippage receivers, instead of taking and parsing those accounts from input.

Swap fee can be manipulated by users

Acknowledged

Path

programs/magpie-router/src/instructions

Function Name

`swap`

Description

The protocol allows users to directly specify the swap_fee amount in the calldata they submit for swaps. The protocol doesn't enforce any minimum fee requirements, which means users can arbitrarily set very small fees or completely avoid fees by setting them to zero.

```
pub fn get_swap_data(
    calldata: &[u8],
    is_from_asset_native: bool,
    is_to_asset_native: bool,
) -> Result<SwapData> {
    // ...
    let (amount_out_min, amount_out_min_size) = from_bytes::<u64>(&calldata[index..], false)?;
    index += amount_out_min_size;
    let (swap_fee, swap_fee_size) = from_bytes::<u64>(&calldata[index..], false)?; //@audit User-provided fee, on top of which we are not validating it against some minimum fee requirement
    index += swap_fee_size;
    // ...
}
```

Later, this fee is applied without any further validation:

```
    pub fn transfer_swap_fees(
    gasless: bool,
    swap_data: &SwapData,
    swap_accounts: &Box<SwapAccounts>,
) -> Result<()> {
    if gasless && swap_data.swap_fee == 0 {
        return err!(ErrorCode::GaslessMustHaveFee);
    }

    if swap_data.swap_fee > 0 { // User can set this to 0 or minimal value
        // Transfer fee logic...
    }

    Ok(())
}
```

While gasless swaps have a check that requires fees to be non-zero, regular swaps can be executed with a swap_fee of 0, allowing users to completely bypass the protocol's fee.

Impact

Users can avoid legitimate fees

Remediation

Create a global configuration account that defines the minimum swap fee required for different types of swaps and validate user supplied fees against these fees.

```
1
2     #[account(zero_copy)]
3   pub struct ProtocolConfig {
4       pub min_swap_fee_bps: u16,       // Minimum fee in basis points
5       pub min_absolute_swap_fee: u64,  // Absolute minimum fee amount
6       // Other protocol parameters
7   }
```

Low Severity Issues

Swap fee can be manipulated by users

Acknowledged

Description

The execute_math function in command.rs contains an unsafe implementation of multiplication that could cause arithmetic overflow for valid operations that produce results within the logical range of expected values but exceed the capacity of a u64.

The current implementation uses checked_mul, which returns None when an overflow occurs:

```
MathOperator::Mul => {
    let result = amount_0
        .checked_mul(amount_1)
        .ok_or(ProgramError::ArithmeticOverflow);

    return Ok(result.unwrap());
}
```

This is problematic because there are legitimate cases where multiplying two large u64 values could produce a result that:
- Is mathematically valid
- Would be needed for protocol operations
- Cannot be represented in a u64 but could be represented in a wider type like u128
- Could be safely used if intermediate calculations are performed in wider types

For example, multiplying two large token amounts (e.g., 10^12 * 10^12) would cause an overflow in u64 even though the operation itself is valid and the result might be needed for certain token calculations.

```
MathOperator::Mul => {
    // Upgrade to u128 for the multiplication
    let amount_0_wide: u128 = amount_0 as u128;
    let amount_1_wide: u128 = amount_1 as u128;

    // Perform multiplication in u128 space
    let result_wide = amount_0_wide * amount_1_wide;

    // Check if result fits in u64
    if result_wide <= u64::MAX as u128 {
        return Ok(result_wide as u64);
    } else {
        return Err(ErrorCode::ArithmeticOverflow.into());
    }
}
```

Registry scalability limitation due to u8 index type

Acknowledged

Description

The update function in registry.rs uses a u8 type for the index parameter, which limits the registry to a maximum of 256 discriminators. The registry stores discriminators that are later used during swap operations to identify and execute commands.

```
/bottlenech to 256 discriminators only because of index being u8
pub fn update(ctx: Context<Registry>, index: u8, discriminators: [u8; 8]) -> Result<()> {
    let registry_data: &mut [u8] = &mut ctx.accounts.registry.try_borrow_mut_data()?;

    let start = index as usize * 8;
    let end = start + 8;
    registry_data[start..end].copy_from_slice(&discriminators);

    Ok(())
}
```

Remediation

Change the index type from u8 to at least u16, which would allow for up to 65,536 discriminators

Missing emergency-stop / pausable mechanism in magpie-router program

Acknowledged

Path & F unction

`magpie-router/ src/`

Description

The program exposes critical operations (swaps, token transfers via CPI, delegation, closing vault/ registry balances) without any global on-chain “pause” flag or instruction that would allow maintainers (or a governance multisig) to immediately halt those operations. Access control is implemented with whitelist/relayers PDAs for privileged actions, but there is no global switch checked by regular instructions to disable normal user flows in an emergency.

Example attack path

Attacker discovers a bug in the swap CPI invocation that lets them craft an instruction sequence draining funds out of a program-owned vault (or they compromise a relayer/whitelist key). Attacker repeatedly invokes swap/delegate/other CPI flows to drain tokens or approve malicious transfers.
Because there is no on-chain pause, developers cannot flip a switch to block future calls; mitigation would require (a) changing off-chain clients (ineffective vs direct RPCusers), (b) upgrading the program (may be delayed and requires upgrade authority), or (c) draining remaining admin-controlled accounts — all slower and riskier than a pause.
(Concrete exploit specifics depend on the exact bug; absence of pause simply removes a fast response option.)

Remediation Suggestion

Add a paused flag in a program account (registry or config PDA)
For example, add pub paused: bool to a Config / registry account that is a PDA owned by the program. Restrict who can toggle it — require a multisig/whitelist signer or timelocked governance to set/unset.

At top of every critical instruction, check the flag require!(!config.paused, ErrorCode::Paused); before swap/transfer/delegate logic.

Make pause toggling multi-sig + time delay for unpausing
Usæa multisig (2/3, 3/5) or governance with a short timelock so a single key compromise cannot be used to unpause rashly.

Example minimal Anchor pattern:

```rust
#[account]
pub struct Config {
    pub admin: Pubkey,
    pub paused: bool,
    // ... other config fields
}

#[derive(Accounts)]
pub struct SetPause<'info> {
    #[account(mut, seeds=[b"config"], bump)]
    pub config: Account<'info, Config>,
    // require multisig or whitelist wallet(s) as signer(s)
    pub signer: Signer<'info>,
}

pub fn set_pause(ctx: Context<SetPause>, paused: bool) -> Result<()> {
    ctx.accounts.config.paused = paused;
    Ok(())
}

// then at top of swap / delegate handlers:
if ctx.accounts.config.paused {
    return err!(ErrorCode::Paused);
}
```

Informational Issues

Missing Validation for Token Account and Mint Correspondence

Acknowledged

Description

The protocol currently lacks explicit validation to ensure that the input_token_account and output_token_account actually correspond to their respective token mints (`from_token_mint` and `to_token_mint`). Without proper validation, users could potentially supply incorrect token accounts during a swap, leading to unexpected behavior or potential exploitation.

When examining the Swapstruct, there are no explicit constraints ensuring that the token accounts match their corresponding supplied mints:

```
#[derive(Accounts)]
pub struct Swap<'info> {
    // ...
    #[account()]
    pub from_token_mint: Option<InterfaceAccount<'info, Mint>>,

    #[account()]
    pub to_token_mint: Option<InterfaceAccount<'info, Mint>>,

    #[account(mut)]
    pub input_token_account: Option<AccountInfo<'info>>,

    #[account(mut)]
    pub output_token_account: Option<AccountInfo<'info>>,
    // ...
}
```

Remediation

```
#[account(
    mut,
    token::mint = from_token_mint,
)]
pub input_token_account: Option<Account<'info, TokenAccount>>,

#[account(
    mut,
    token::mint = to_token_mint,
)]
pub output_token_account: Option<Account<'info, TokenAccount>>,
```

Functional Tests

Initialize

- ✓ Initializes registry, whitelist, relayers, and vault accounts with correct PDA seeds.
- ✓ Sets the initializer as the first whitelisted address in index 0.
- ✓ Fails if the signer is not the upgrade authority of the program.
- ✓ Skips initialization of accounts that already exist.

Update Whitelist

- ✓ Updates address at specified index in whitelist when called by whitelisted signer.
- ✓ Fails if signer is not in the whitelist.
- ✓ Succeeds when replacing an existing address with a new one.
- ✓ Succeeds when adding a new address to an empty slot.

Update Relayers

- ✓ Updates address at specified index in relayers list when called by whitelisted signer.
- ✓ Fails if signer is not in the whitelist.
- ✓ Succeeds when replacing an existing relayer with a new one.
- ✓ Succeeds when adding a new relayer to an empty slot.

Registry Management

Fill Registry

- ✓ Allocates space and stores multiple discriminators when called by whitelisted signer.
- ✓ Fails if signer is not in the whitelist.
- ✓ Correctly transfers required rent from signer to registry account.
- ✓ Properly resizes the account to accommodate new discriminators.

Update Registry

- ✓ Updates discriminator at specified index when called by whitelisted signer.
- ✓ Fails if signer is not in the whitelist.
- ✓ Correctly updates existing discriminator without changing others.

Remove from Registry

- ✓ Zeroes out discriminator at specified index when called by whitelisted signer.
- ✓ Fails if signer is not in the whitelist.

Swap

- ✓ Succeeds when all accounts and signature are valid.
- ✓ Correctly transfers tokens from user to vault (for input) and vault to user (for output).
- ✓ Transfers correct swap fees when is_bridge_call is false.
- ✓ Bypasses swap fees when is_bridge_call is true.
- ✓ Handles gasless transactions correctly when a whitelisted magpie_wallet is the signer.
- ✓ Fails if output amount is less than minimum specified.
- ✓ Correctly initializes and closes required token accounts.
- ✓ Properly validates accounts before and after command execution.
- ✓ Fails if token accounts' state is invalid after operations.
- ✓ Emits SwapEvent with correct details after successful swap.

Delegate

- ✓ Succeeds when called by whitelisted signer.
- ✓ Transfers correct native SOL amount from vault to signer.
- ✓ Correctly approves provided token accounts to be used by signer.
- ✓ Fails if the vault would become non-rent-exempt after operation.
- ✓ Fails if signer is not in the whitelist.

Close

- ✓ Successfully closes whitelist, relayers, registry, and vault accounts when called by whitelisted signer.
- ✓ Correctly transfers all lamports from these accounts to the magpie_wallet.
- ✓ Fails if signer is not in the whitelist.

Token Account Management

- Initializes user-owned token accounts with correct parameters.
- Initializes vault-owned token accounts with correct parameters.
- Correctly validates token accounts against their respective mints.
- Successfully closes token accounts and refunds rent to appropriate recipient.
- Fails if attempting to initialize a token account with an invalid mint.

Command Execution

- Successfully executes Call command with correct parameters.
- Successfully executes Transfer command and moves tokens between accounts.
- Successfully executes Wrap command to convert SOL to wrapped SOL.
- Successfully executes Unwrap command to convert wrapped SOL to SOL.
- Successfully executes Balance command and returns correct balances.
- Successfully executes Math command with correct arithmetic operations.
- Successfully executes Comparison command with correct comparison logic.
- Successfully executes TransferSurplus command and correctly handles slippage.
- Fails if vault's balance decreases after command execution.
- Fails if token account authority changes unexpectedly.

Automated Tests

No major issues were found. Somefalse positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Threat Model

This is a threat model prepared during the audit process to map potential risks and understand protocol attack surfaces. Kindly do not consider these as issues — all verified issues are documented separately above.

Function	Threats
initialize	- Initialization control centralized to upgrade authority only. - No validation for secure or robust initial whitelist address. - Single point of failure with one initial whitelisted address. - No administrator role transition or backup mechanism.
update_whitelist / update_relayers	- No multi-sig or timelock requirements for critical whitelist changes. - Index-based updates allow overwriting existing entries without validation. - Malicious whitelisted user could compromise protocol by replacing legitimate addresses. - No event emission for tracking critical administrative changes.
fill_registry / update_registry / remove_from_registry	- Registry can grow unbounded with fill_registry (DoS risk). - Discriminator validity not verified before storage. - Limited to 256 entries due to u8 index type. - No validation of discriminator authenticity or purpose.

Function	Threats
swap	- is_bridge_call parameter can be abused to bypass swap fees entirely. - Rent farming attack possible by initializing and closing user-owned token accounts. - Fee receivers can be manipulated to redirect protocol revenue. - Swap fee can be set to zero by users to avoid fees. - Integer overflow possible in math operations during swap execution. - No validation that token accounts correspond to their declared mints. - Complex calldata processing with limited validation. - Vault could be drained through command sequence manipulation. - Token account authority validation bypasses when accounts are closed.
delegate	- Whitelisted signers can withdraw SOL from vault with minimal restrictions. - Full token balance approval to delegates without time limitations. - No mechanism to revoke delegations once granted. - Potential vault insolvency if multiple delegates request funds. - No event emission for tracking delegation activities.
close	- Irreversible protocol shutdown with no recovery mechanism. - Any whitelisted address can trigger closure of critical protocol accounts. - User funds could be stranded if closure happens during active operations. - No timelock or multi-sig requirement for this critical function.

Function	Threats
Vault	- Single PDA holds all protocol funds (centralized point of failure). - No minimum balance requirements or reserves. - Pays for user token account initialization without proper validation. - No treasury management or emergency withdrawal mechanisms.
Whitelist	- Fixed-size array limits protocol governance flexibility. - No role differentiation among whitelisted addresses. - No timelock or voting for adding/removing members. - Single compromised whitelisted address threatens entire protocol.
Relayers	- Gasless transaction mechanism can be abused if relayer is compromised. - No rate limiting for relayer operations. - No verification of relayer compensation. - Fixed-size array limits scalability.
Token Account Handling	- Post-operation validation can be bypassed when accounts are closed. - No validation for token mint/account correspondence. - Vault pays for initialization of user-owned token accounts. - Missing validation for minimum token balances. - No slippage protection beyond minimum amount check.
Command Execution	- Complex command execution with minimal security checks. - No validation of invoked program safety. - CPI calls could manipulate account state unexpectedly. - Math operations vulnerable to overflow/underflow. - No validation of reasonable operation sizes/amounts.

Closing Summary

In this report, we have considered the security of Fly Trade. We performed our audit according to the procedure described above.

issues of Medium,Low and Informational severity were found. Fly Trade fixed two and acknowledged the remaining issues mentioned in the report.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With seven years of expertise, we've secured over 1400 projects globally, averting over $3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

November 2025

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com