Smart Contract

Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.08.15, the SlowMist security team received the YIEDL team's security audit application for Yiedl Drift Delegator Bot, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Unsafe External Call Audit
- Design Logic Audit
- Scoping and Declarations Audit
- Account substitution attack Audit
- Malicious Event Log Audit

3 Project Overview

3.1 Project Introduction

Solana program for making trades on the Drift Protocol using Drift's Delegate Feature.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	`program_authority` PDA lacks program id verification	Account substitution attacks	Low	Fixed
N2	Key accounts missing program owner verification	Account substitution attacks	High	Fixed
N3	Unverified whether the account is owned by the correct Token Program	Account substitution attacks	Suggestion	Fixed
N4	`market_cache_account` lacks seed constraint validation	Account substitution attacks	High	Fixed
N5	Fields redundancy leads to data inconsistency	Others	Suggestion	Fixed
N6	`temp_data` account closure mechanism flaw	Design Logic Audit	High	Fixed
N7	`program_state` lacks seed constraint validation	Account substitution attacks	High	Fixed
N8	`PythLazerOracle` lacks price validity verification	Unsafe External Call Audit	High	Fixed
N9	Different function calls used the same log event	Malicious Event Log Audit	Low	Fixed
N10	Excessive Privilege Concentration	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/yiedl-ai/yiedl_drift_delegator_bot

Initial audit commit: d269c53df858dd79c92ca5e955f594e29314d756

Final review commit: 33bee8b84783f0977c56f5397823346ac145c13a

Audit Scope:

```
./programs/yiedl_drift_delegator/src
├── bot
│   ├── admin.rs
│   ├── bot_user.rs
│   ├── errors.rs
│   ├── market_cache.rs
│   ├── portfolio.rs
│   ├── program_state.rs
│   └── utils.rs
├── bot.rs
├── drift
│   ├── common_types.rs
│   ├── constants.rs
│   ├── cpi.rs
│   ├── oracles.rs
│   ├── perp_market.rs
│   ├── spot_market.rs
│   └── user.rs
├── drift.rs
└── lib.rs
```

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist security team analyzed the visibility of major contracts during the audit, the result as follows:

yiedl_drift_delegator			
Function Name	Account check coverage	Auth Signer	Parameters Check
admin_update_fee_admin	2/2	master_admin	1/1

yiedl_drift_delegator			
admin_update_operations_admin	2/2	master_admin	2/2
admin_update_master_admin	2/2	master_admin	1/1
admin_relinquish_fee_admin	2/2	fee_admin	-
admin_relinquish_operations_admin	2/2	operations_admin	1/1
admin_initialize_program_state	4/4	operations_admins	-
admin_update_fee_params	3/3	fee_admin	5/5
admin_update_account_creation_fee	3/3	fee_admin	1/1
admin_update_fee_collector	4/4	fee_admin	1/1
admin_update_cycle_duration	3/3	operations_admins	1/1
admin_set_delegate	5/6	operations_admins	3/3
admin_claim_fees	10/15	operations_admins	2/4
admin_record_rotation_fees	4/4	operations_admins	0/2
admin_align_status	3/3	operations_admins	5/7
admin_update_market_cache	2/3	operations_admins	2/2
admin_update_yiedl_token_mint	3/3	master_admin	1/1
admin_update_yiedl_discount_rate_and_price	3/3	fee_admin	0/2
initialize_admin	3/3	payer	-
initialize_temp_data	5/6	payer	0/2
close_temp_data	3/3	payer	0/2
initialize_market_cache	3/3	payer	-

yiedl_drift_delegator			
process_usdc_spot	1/4	-	0/2
process_one_unrealized_pnl	1/6	-	2/3
record_overall_portfolio_value	2/5	-	0/2
owner_create_user	16/20	owner&payer	2/2
owner_deposit_usdc_to_drift	13/19	owner&payer	3/3
owner_update_name	4/5	owner	2/2
owner_make_request	5/5	owner	3/3
owner_withdraw_usdc_from_drift	7/13	authority	2/4
owner_set_delegate	5/6	signer	0/3
owner_deposit_yiedl	8/8	owner&payer	0/1
owner_withdraw_yiedl	5/5	owner	0/1

4.3 Vulnerability Summary

[N1] [Low] `program_authority` PDA lacks program id verification

Category: Account substitution attacks

Content

In a multi-function account structure, the `program_authority` field uses the `AccountInfo<'info>` type, which only validates the PDA's seeds and bump, but does not verify the account's program owner. This allows attackers to pass in malicious accounts with the same seeds created by other programs.

- programs/yiedl_drift_delegator/src/bot/admin.rs

```
pub struct ClaimFees<'info> {
    /// CHECK: Program authority PDA for Drift seeds.
    #[account(
        seeds = [
            PROGRAM_AUTHORITY_SEED,
            owner_key.as_ref(),
            &bot_user_sub_id.to_le_bytes(),
        ],
        bump
    )]
    pub program_authority: AccountInfo<'info>,
```

- programs/yiedl_drift_delegator/src/bot/bot_user.rs

```
pub struct CreateUser<'info> {
pub struct DepositUsdcToDrift<'info> {
pub struct WithdrawUsdcFromDrift<'info> {
pub struct SetDelegate<'info> {
pub struct UpdateName<'info> {

#[account(
    seeds = [
        PROGRAM_AUTHORITY_SEED,
        owner.key().as_ref(),
        &bot_user_sub_id.to_le_bytes(),
    ],
    bump
)]
pub program_authority: AccountInfo<'info>,
```

Incorrect Program Authority may lead to call failure, or even privilege bypass.

Solution

Initialize this account, and add constraints `owner = &crate::ID`.

Status

Fixed

[N2] [High] Key accounts missing program owner verification

Category: Account substitution attacks

Content

These accounts are declared as the `AccountInfo<'info>` type, with no constraints to verify whether they belong to the correct program (the Drift program).

Attackers can pass in forged accounts controlled by malicious programs, thereby manipulating the results of the investment portfolio value calculation.

- programs/yiedl_drift_delegator/src/bot/portfolio.rs

```
pub struct ProcessUsdcSpot<'info> {
   //...
   pub spot_account: AccountInfo<'info>,
   //...
   pub oracle_account: AccountInfo<'info>,

//...
pub struct ProcessOneUnrealizedPnL<'info> {
   //...
   pub market_cache_account: Account<'info, MarketCache>,
   //...
   pub quote_oracle_account: AccountInfo<'info>,

//...
pub struct RecordOverallPortfolioValue<'info> {
   //...
   pub usdc_spot_account: AccountInfo<'info>,
   //...
   pub usdc_oracle_account: AccountInfo<'info>,
}
```

Solution

These accounts are not in CPI calls and will not be verified elsewhere; they need to be verified for ownership and derivation within the current program.

Status

Fixed

[N3] [Suggestion] Unverified whether the account is owned by the correct Token Program

Category: Account substitution attacks

Content

Using `AccountInfo` instead of the strongly-typed `Account<TokenAccount>` indeed does not restrict the program owner of the account, which poses serious security risks:

- programs/yiedl_drift_delegator/src/bot/admin.rs

```
#[derive(Accounts)]
#[instruction(owner_key: Pubkey, bot_user_sub_id: u16)]
pub struct ClaimFees<'info> {
   //...
   #[account(
       mut,
       seeds = [YIEDL_SEED, owner_key.as_ref()],
       bump,
   )]
   pub program_owner_yiedl_account: AccountInfo<'info>,
```

Attackers can input any account data that a program possesses, they can input forged account data.

Solution

```
#[account(
    init_if_needed,
    payer = authority,
    seeds = [YIEDL_SEED, owner_key.as_ref()],
    bump,
    token::mint = program_state.yiedl_mint,
    token::authority = program_owner_yiedl_account,
)]
pub program_owner_yiedl_account: Box<Account<'info, TokenAccount>>,
```

Status

Fixed

[N4] [High] `market_cache_account` lacks seed constraint validation

Category: Account substitution attacks

Content

In the `UpdateMarketCache` account structure, the `market_cache_account` field lacks seed constraint validation.

Although the account uses the correct seed constraints in `InitializeMarketCache`, there is no corresponding validation in the update operation.

- programs/yiedl_drift_delegator/src/bot/market_cache.rs

```
pub struct UpdateMarketCache<'info> {
    #[account(mut)]
    pub market_cache_account: Account<'info, MarketCache>,
```

Solution

```
    #[account(
        seeds = [MARKET_CACHE_SEED],
        bump
    )]
    pub market_cache_account: Account<'info, MarketCache>,
```

Status

Fixed

[N5] [Suggestion] Fields redundancy leads to data inconsistency

Category: Others

Content

In the `BotUserState` structure, a `delegate` field is defined to record the delegate address, but this field is redundant because:

1.The Drift protocol itself has a delegation feature: The contract sets the delegate by calling Drift's `update_user_delegate` function via CPI.

2.Dual recording can lead to inconsistencies: The local `delegate` field may not be consistent with the actual delegation status in the Drift protocol.

3.Lack of utility: Throughout the contract code, apart from the setting operation, there is no other place where this `delegate` field is read or used.

- programs/yiedl_drift_delegator/src/bot/bot_user.rs

```
pub fn handle_set_delegate(
    ctx: Context<SetDelegate>,
    target_owner: Pubkey, ////target_owner可以是signer或操作员
    bot_user_sub_id: u16,
    delegate: Pubkey,
) -> Result<()> {
//...
    ctx.accounts.bot_user_state.delegate = delegate;
    Ok(())
}
```

The same issue in `handle_update_name` function, the `bot_user_state.name` record is redundant.

```
pub fn handle_update_name(
   ctx: Context<UpdateName>,
   bot_user_sub_id: u16,
   new_name: String,
) -> Result<()> {
//...
   bot_user_state.name = string_to_bytes32(new_name.as_str());
```

Solution

If it is indeed necessary to delegate/name information, it should be read directly from the Drift protocol account, rather than maintaining a local copy.

Status

Fixed

[N6] [High] `temp_data` account closure mechanism flaw

Category: Design Logic Audit

Content

In the `CloseTempData` account structure, there are two issues with the mechanism to close the temp_data account:

1.Rent is refunded to the non-creator: Rent will be refunded to the payer who calls the close function, rather than to the original creator of the temporary data account.

2.Lack of data processing status validation: Anyone can close the temporary data account at any time, even if the data may still be processing or not yet completed.

- programs/yiedl_drift_delegator/src/bot/portfolio.rs

```
/// Closes the temporary data account.
pub fn handle_close_temp_data(
   ctx: Context<CloseTempData>,
   target_owner: Pubkey,
   bot_user_sub_id: u16,
) -> Result<()> {
   if ctx.accounts.temp_data.to_account_info().data_len() > 0 {
       msg!(
           "Closing temp data for owner: {}, bot_user_sub_id: {}",
           target_owner,
           bot_user_sub_id
       );
   } else {
       msg!(
           "Temp data for owner: {}, bot_user_sub_id: {} is already empty, skipping close.",
           target_owner,
           bot_user_sub_id
       );
   }
   Ok(())
}

pub struct CloseTempData<'info> {
   #[account(mut)]
   pub payer: Signer<'info>,


   #[account(
       mut,
       seeds = [TEMP_DATA_SEED, target_owner.as_ref(), &bot_user_sub_id.to_le_bytes()],
       bump,
       close = payer //@audit
   pub temp_data: Account<'info, TempData>,


   pub system_program: Program<'info, System>,
}
```

Solution

Verify whether the processing is completed, and return the rent to `temp_data.owner`.

Status

Fixed

[N7] [High] `program_state` lacks seed constraint validation

Category: Account substitution attacks

Content

In the `UpdateFeeParams / UpdateFeeCollector / UpdateCycleDuration / UpdateYiedlTokenMint / UpdateYiedlDiscountRateAndPrice` account structure, the `program_state` field lacks seed constraint validation. Although the account uses the correct seed constraints in `InitializeProgramState`, there is no corresponding validation in the update operation.

- programs/yiedl_drift_delegator/src/bot/program_state.rs

```
  128,5:     pub program_state: Account<'info, ProgramState>,
  169,5:     pub program_state: Account<'info, ProgramState>,
  206,5:     pub program_state: Account<'info, ProgramState>,
  230,5:     pub program_state: Account<'info, ProgramState>,
  262,5:     pub program_state: Account<'info, ProgramState>,
  334,5:     pub program_state: Account<'info, ProgramState>,
```

Solution

Validate seeds for `program_state`.

Status

Fixed

[N8] [High] `PythLazerOracle` lacks price validity verification

Category: Unsafe External Call Audit

Content

In `portfolio.rs`, the program directly uses the price data from `PythLazerOracle` for critical financial calculations, but there is a complete lack of validation for the validity of the oracle data. This could lead to the system making decisions based on outdated, invalid, or manipulated price data.

- programs/yiedl_drift_delegator/src/drift/oracles.rs

```
pub struct PythLazerOracle {
   pub price: i64,
   pub publish_time: u64,
   pub posted_slot: u64,
   pub exponent: i32,
   pub _padding: [u8; 4],
   pub conf: u64,
}
```

- programs/yiedl_drift_delegator/src/bot/portfolio.rs

```
  352,102:      let oracle_price_data = load_account::<PythLazerOracle>
(&ctx.accounts.oracle_account)?;
  493,33:      let oracle = load_account::<PythLazerOracle>
(&ctx.accounts.amm_oracle_account)?;
```

```
  495,45:     let quote_oracle_lazer = load_account::<PythLazerOracle>
(&ctx.accounts.quote_oracle_account)?;
  603,38:     let usdc_oracle = load_account::<PythLazerOracle>
(&ctx.accounts.usdc_oracle_account)?;
```

Solution

Check the validity of the `PythLazerOracle` data, such as ensuring that the `publish_time` is within a reasonable range.

Status

Fixed

[N9] [Low] Different function calls used the same log event

Category: Malicious Event Log Audit

Content

There are multiple instances of identical event emission issues in the Yiedl, which can make it difficult for off-chain systems to distinguish the operation functions that trigger the events, potentially leading to errors in off-chain business systems.

1. `FeeClaimed` being called in these functions:

- programs/yiedl_drift_delegator/src/bot/admin.rs

```
pub fn handle_admin_claim_fees<'info>(
//...
       emit!(FeeClaimed {
```

- programs/yiedl_drift_delegator/src/bot/bot_user.rs

```
pub fn handle_deposit_usdc_to_drift(
//...
       emit!(FeeClaimed {
```

2. `RequestMade` being called in these functions:

- programs/yiedl_drift_delegator/src/lib.rs

```
pub fn owner_withdraw_usdc_from_drift<'info>(
//...
       emit!(RequestMade {
```

- programs/yiedl_drift_delegator/src/bot/bot_user.rs

```
pub fn update_status_and_get_fee_amount(
//...
   emit!(RequestMade {
```

3. `FeeRecorded` being called in these functions:

- programs/yiedl_drift_delegator/src/bot/bot_user.rs

```
pub fn handle_deposit_usdc_to_drift(
//...
   emit!(FeeRecorded {
```

- programs/yiedl_drift_delegator/src/bot/program_state.rs

```
pub fn handle_record_rotation_fees(
//...
   emit!(FeeRecorded {
```

Solution

Customize different events for each call.

Status

Fixed

[N10] [Medium] Excessive Privilege Concentration

Category: Authority Control Vulnerability Audit

Content

The YIEDL Drift Delegator Bot contract has a centralization risk and an issue of excessive concentration of power in its administrator authority design.

For example, if the Master Admin private key is leaked, the attacker can: Replace all other administrators, control the token contract address, take over the protocol governance completely.

Here is the list of permissions owned by the administrator:

1. `fee_admin` can:

admin_relinquish_fee_admin

admin_update_fee_params

admin_update_account_creation_fee

admin_update_fee_collector

2. `master_admin` can:

admin_update_fee_admin

admin_update_operations_admin

admin_update_master_admin

admin_update_yiedl_token_mint

3. `operations_admin` can:

admin_relinquish_operations_admin

admin_initialize_program_state

admin_update_cycle_duration

admin_set_delegate

admin_claim_fees

admin_record_rotation_fees

admin_align_status

admin_update_market_cache

Solution

In the short term, transferring privileged roles to a multi-signature wallet can effectively mitigate the single point of failure risk. In the long term, transferring privileged roles to DAO governance can effectively address the risk of excessive privilege. During the transition period, managing through multi-signature with delayed transaction execution via timelock can effectively mitigate the risk of excessive privilege.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002508260001	SlowMist Security Team	2025.08.15 - 2025.08.26	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 5 high risk, 1 medium risk, 2 low risk, 2 suggestion vulnerabilities.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

SLOWMIST

Official Website

www.slowmist.com

✉

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist