OFFSIDE LABS

Reflect Delta Neutral

Smart Contract Security Assessment

September 2025

Prepared for:

Reflect

Prepared by:

Offside Labs

Ripples Wen

Sirius Xie

Contents

1 About Offside Labs

Offside Labs is a leading security research team, composed of top talented hackers from both academia and industry.

We possess a wide range of expertise in modern software systems, including, but not limited to, *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. We are also at the forefront of innovative areas like *cryptocurrencies* and *blockchain technologies*. Among our notable accomplishments are remote jailbreaks of devices such as the **iPhone** and **PlayStation 4**, and addressing critical vulnerabilities in the **Tron Network**.

Our team actively engages with and contributes to the security community. Having won and also co-organized *DEFCON CTF*, the most famous CTF competition in the Web2 era, we also triumphed in the **Paradigm CTF 2023** within the Web3 space. In addition, our efforts in responsibly disclosing numerous vulnerabilities to leading tech companies, such as *Apple*, *Google*, and *Microsoft*, have protected digital assets valued at over **$300 million**.

In the transition towards Web3, Offside Labs has achieved remarkable success. We have earned over **$9 million** in bug bounties, and **three** of our innovative techniques were recognized among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs

2 Executive Summary

Introduction

Offside Labs completed a security audit of *Reflect-Delta-Neutral* smart contracts, starting on June 13th, 2025, and concluding on July 16th, 2025.

Project Overview

Reflect is a programmable financial protocol that tokenizes on-chain yield through delta-neutral strategies, creating productive financial instruments without intermediaries. The platform captures yield from various sources and distributes it to token holders, offering an innovative alternative to DeFi lending and borrowing as well as TradFi hedge funds.

Reflect implements multiple strategies that transform raw assets into yield-generating tokens. The system architecture is modular, with separate accounts and logic for each strategy and DEX integration. These strategy controllers act as authorities for their respective token mints and manage the complex position logic required for yield generation.

The protocol currently implements two strategies: S1 (USDC Strategy) for pure USDC lending yield, and S3 (LST Strategy) for capturing LST staking rewards while opening short perpetual positions to hedge price risk. Users deposit collateral (USDC or LST tokens), receive tokenized yield-bearing assets (rUSD), and the protocol then automatically manages the delta-neutral positions to generate consistent returns from staking yields without directional price exposure.

Audit Scope

The assessment scope contains mainly the smart contracts of the reflect-main program for the *Reflect-Delta-Neutral* project.

The audit is based on the following specific branches and commit hashes of the codebase repositories:

- Reflect-Delta-Neutral
  - Codebase: `https://github.com/palindrome-eng/reflect-delta-neutral`
  - Commit Hash: 7d732a5c71f8f0269c29f105e1d68a86656caf4b

We listed the files we have audited below:

- Reflect-Delta-Neutral
  - programs/reflect-main/src/common/common.rs
  - programs/reflect-main/src/common/drift/adjust_hedge.rs
  - programs/reflect-main/src/common/drift/drift_calls.rs
  - programs/reflect-main/src/common/drift/drift_helpers.rs
  - programs/reflect-main/src/common/ids.rs
  - programs/reflect-main/src/common/load.rs
  - programs/reflect-main/src/common/math/cast.rs
  - programs/reflect-main/src/common/math/conversion.rs

- programs/reflect-main/src/common/math/rebalance.rs
- programs/reflect-main/src/common/math/rebalancing.rs
- programs/reflect-main/src/common/math/split.rs
- programs/reflect-main/src/common/math/utils.rs
- programs/reflect-main/src/instructions/admin/access/action_update.rs
- programs/reflect-main/src/instructions/admin/access/create_admin_account.rs
- programs/reflect-main/src/instructions/admin/access/role_holder_update.rs
- programs/reflect-main/src/instructions/admin/admin_context.rs
- programs/reflect-main/src/instructions/admin/dex/drift/capture_spread_drift.rs
- programs/reflect-main/src/instructions/admin/dex/drift/controller_s1/init_controller_s1.rs
- programs/reflect-main/src/instructions/admin/dex/drift/controller_s1/init_drift_accounts_s1.rs
- programs/reflect-main/src/instructions/admin/dex/drift/controller_s3/add_lst_drift.rs
- programs/reflect-main/src/instructions/admin/dex/drift/controller_s3/add_sub_account.rs
- programs/reflect-main/src/instructions/admin/dex/drift/controller_s3/init_controller_s3.rs
- programs/reflect-main/src/instructions/admin/dex/drift/controller_s3/rebalance/process_swap_deposit.rs
- programs/reflect-main/src/instructions/admin/dex/drift/controller_s3/rebalance/process_swap_withdraw.rs
- programs/reflect-main/src/instructions/admin/dex/drift/controller_s3/rebalance/swap_context.rs
- programs/reflect-main/src/instructions/admin/dex/drift/create_user_stats_account.rs
- programs/reflect-main/src/instructions/admin/dex/drift/deposit_drift.rs
- programs/reflect-main/src/instructions/admin/init_main.rs
- programs/reflect-main/src/instructions/admin/security/freeze_functionality.rs
- programs/reflect-main/src/instructions/admin/security/freeze_program.rs
- programs/reflect-main/src/instructions/admin/security/suspend_spl.rs
- programs/reflect-main/src/instructions/admin/strategy/update_attenuation.rs
- programs/reflect-main/src/instructions/admin/strategy/update_cap.rs
- programs/reflect-main/src/instructions/admin/strategy/update_recipients.rs
- programs/reflect-main/src/instructions/admin/swap/swap_orca.rs
- programs/reflect-main/src/instructions/admin/swap/swap_orca_two_hop.rs
- programs/reflect-main/src/instructions/user/drift/s1/mint_drift_s1.rs
- programs/reflect-main/src/instructions/user/drift/s1/redeem_drift_s1.rs
- programs/reflect-main/src/instructions/user/drift/s1/supply_management_context_s1.rs
- programs/reflect-main/src/instructions/user/drift/s3/mint_drift_s3.rs
- programs/reflect-main/src/instructions/user/drift/s3/redeem_drift_s3.rs
- programs/reflect-main/src/instructions/user/drift/s3/supply_management_context_s3.rs

- programs/reflect-main/src/instructions/user/drift/settle/settle_context.rs
    - programs/reflect-main/src/instructions/user/drift/settle/settle_pnl.rs
    - programs/reflect-main/src/instructions/user/drift/settle/settle_pnl_multi.rs
    - programs/reflect-main/src/state/components/access.rs
    - programs/reflect-main/src/state/components/action.rs
    - programs/reflect-main/src/state/components/attenuation.rs
    - programs/reflect-main/src/state/components/externals.rs
    - programs/reflect-main/src/state/components/killswitch.rs
    - programs/reflect-main/src/state/components/recipients.rs
    - programs/reflect-main/src/state/components/spls.rs
    - programs/reflect-main/src/state/credentials.rs
    - programs/reflect-main/src/state/dexes/drift/drift_base_controller.rs
    - programs/reflect-main/src/state/dexes/drift/jlp_controller/jlp_controller.rs
    - programs/reflect-main/src/state/dexes/drift/lst_controller/lst_controller.rs
    - programs/reflect-main/src/state/dexes/drift/lst_controller/rebalance_lst.rs
    - programs/reflect-main/src/state/dexes/drift/usdc_controller/usdc_controller.rs
    - programs/reflect-main/src/state/jlp.rs
    - programs/reflect-main/src/state/main.rs
    - programs/reflect-main/src/state/strategy.rs

Excluded Files:

- Reflect-Delta-Neutral
    - programs/reflect-main/src/common/drift/drift_components.rs
    - programs/reflect-main/src/common/oracles/pyth/pyth_components.rs
    - programs/reflect-main/src/common/drift/maps/mod.rs
    - programs/reflect-main/src/common/drift/maps/oracle_map.rs
    - programs/reflect-main/src/common/drift/maps/spot_market_map.rs
    - programs/reflect-main/src/common/drift/maps/perp_market_map.rs
    - programs/reflect-main/src/common/drift/maps/load_maps.rs
    - drift-sdk/drift_interface/src/accounts.rs
    - drift-sdk/drift_interface/src/instructions.rs
    - drift-sdk/drift_interface/src/errors.rs
    - drift-sdk/drift_interface/src/events.rs
    - drift-sdk/drift_interface/src/lib.rs
    - drift-sdk/drift_interface/src/typedefs.rs
    - jlp-sdk/perpetuals_interface/src/accounts.rs
    - jlp-sdk/perpetuals_interface/src/instructions.rs
    - jlp-sdk/perpetuals_interface/src/errors.rs
    - jlp-sdk/perpetuals_interface/src/events.rs
    - jlp-sdk/perpetuals_interface/src/lib.rs
    - jlp-sdk/perpetuals_interface/src/typedefs.rs

Findings

The security audit revealed:

- 2 critical issues
- 4 high issues
- 7 medium issues
- 21 low issues
- 16 informational issues

Further details, including the nature of these issues and recommendations for their remediation, are detailed in the subsequent sections of this report.

3 Summary of Findings

ID	Title	Severity	Status
01	Missing Drift Program ID Validation	Critical	Fixed
02	Stable Token Minting Ignores Existing Supply	Critical	Fixed
03	Lack of Ownership Verification in SpotMarketMap and PerpMarketMap Load Functions	High	Fixed
04	Unvalidated Strategy Identifier Facilitates Cross-Strategy Access Control Bypass	High	Fixed
05	Incorrect Attenuated Output Calculation Relative to Fill Rate	High	Fixed
06	Front-Running Threat in Reward Distribution	High	Fixed
07	Inconsistent Stable Token Valuation Between Mint and Redeem Operations	Medium	Fixed
08	Insufficient User Account Validation in capture_spread_drift Instruction	Medium	Fixed
09	Lack of Permission Check in add_lst_drift Instruction	Medium	Fixed
10	Insufficient Permission Verification in capture_spread_drift	Medium	Fixed
11	Type Mismatch in CPI Parameters for place_and_take_perp_order	Medium	Fixed
12	Partial Consideration of User Accounts in capture_spread_drift	Medium	Fixed
13	Stable Token Over-Minting from Excluding Unrealized Perpetual Losses	Medium	Fixed
14	Improper Action Permission Check in update_action_role_protocol	Low	Fixed
15	Insufficient Mint Account Validation in update_cap Instruction	Low	Fixed
16	Incorrect Permission Check Scope in update_action_role_strategy Instruction	Low	Fixed
17	Lack of Proper Validation for Spot Market in add_lst_drift	Low	Fixed
18	Incorrect Collateral Calculation Due to Borrowed LST Holdings	Low	Fixed
19	Insufficient Mint Account Validation in capture_spread_drift Instruction	Low	Fixed
20	Risk of Incorrect Sub-Account ID Due to Index Misalignment	Low	Fixed
21	Incorrect Order of Attenuation and Slippery Check	Low	Fixed
22	user_account_2 Not Properly Validated Against PDA Address	Low	Fixed

ID	Title	Severity	Status
23	Missing Validation of lst_mint and spot_market_lst.mint Consistency	Low	Fixed
24	Missing Strategy Controller Verification in suspend_lst_strategy	Low	Fixed
25	Perpetual Order Amount Not Aligned with Protocol Increment Constraint	Low	Fixed
26	Missing PnL Settlement Call in handle_settlement Function	Low	Fixed
27	Inconsistent Minimum Position Validation in min_position_perp Function	Low	Fixed
28	LST Token Residue from Unmatched Deposit Amounts	Low	Fixed
29	Improper Valuation Basis in capture_spread_drift Instruction	Low	Fixed
30	USDC Collateral Movement Uses Dollar Value Instead of Token Amount	Low	Fixed
31	Redemption at Stale Share Price After Pool Value Drop	Low	Fixed
32	Incorrect Calculation of Residual USDC in mint_drift_s1 and mint_drift_s3 Instructions	Low	Fixed
33	Inaccurate Account Size Calculation	Low	Fixed
34	Missing Last Active Slot Updates During Reflect-Drift Interactions	Low	Acknowledged
35	Incorrect Strategy Index in Issue Event Emission	Informational	Fixed
36	Incorrect Stable Amount in Issue Event Emission	Informational	Fixed
37	Lack of Validation for address Alignment with update_admin_permissions	Informational	Fixed
38	Unchecked strategy_id Argument in Event Logging	Informational	Fixed
39	Redundant Calculation of dollar_value_trunc in mint_math Function	Informational	Fixed
40	Improper USDC Transfer Amount Leads to Dust in Strategy Controller Account	Informational	Fixed
41	Improper Validation of user_lst_ata Mint Constraint	Informational	Fixed
42	Redundant Suspension State Check in mint_drift_s3	Informational	Fixed
43	Missing Validation of Drift Program Activation in create_user_stats_account	Informational	Fixed
44	Inconsistent Placement of Drift Program ID Verification	Informational	Fixed
45	Redundant Invocation of insert_strategy in update_role_holder_strategy	Informational	Fixed

ID	Title	Severity	Status
46	`Unnecessary Account Requirements in Instruction Definitions`	Informational	Fixed
47	`Redundant Checks in ActionMapping.add_role`	Informational	Fixed
48	`Semantic Inconsistency Between Comments and Implementation in KillSwitch`	Informational	Fixed
49	`Hardcoded Space Size`	Informational	Fixed
50	`Inconsistency Between Comments and Formula in Attenuation Parameters`	Informational	Fixed

4 Key Findings and Recommendations

4.1 Missing Drift Program ID Validation

Severity: Critical	**Status: Fixed**
Target: Smart Contract	**Category: Data Validation**

Description

```
/// CHECK: Validates that provided program matches pubkey of drift DEX.
#[account(constraint =
↪   main.externals.is_program_active(&drift_program::ID)?
↪   @ReflectErrorCodes::ProgramNotActive)]
pub drift: AccountInfo<'info>,
```

programs/reflect-main/src/instructions/user/drift/s3/supply_management_-context_s3.rs#L50-L52

Within the `mint_drift_s3` , `redeem_drift_s3` , `process_swap_deposit` , and `process_swap_withdraw` instructions, the provided Drift program account is not adequately checked to ensure that its public key matches the authorized Drift DEX program ID. The lack of a proper validation may allow an attacker to supply a malicious or incorrect program account during instruction execution.

Impact

The lack of a strict program ID validation presents a critical security vulnerability. An attacker could exploit this by providing a malicious program account in place of the legitimate Drift DEX. This could lead to the execution of logic under false assumptions, potentially resulting in unauthorized behavior or manipulation of the protocol.

Proof of Concept

An attacker can craft a transaction that passes a spoofed `drift` program account to the affected instructions. Since the current constraint does not enforce an exact match with `drift_program::ID` , the instruction would proceed. Given that the internal logic may involve the strategy account acting as a signer, the malicious program could invoke Drift with unauthorized privileges. This could allow the attacker to drain assets from the protocol and transfer them to their own control, bypassing security mechanisms.

Recommendation

Enforce a strict validation to ensure that the passed `drift` program account matches the expected `drift_program::ID` .

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.2 Stable Token Minting Ignores Existing Supply

Severity: Critical	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

When capturing yield for the s3 strategy, the quantity of stable tokens to be minted is currently calculated based solely on the total USD value of an individual sub-account's positions. This methodology does not take into account the existing supply of stable tokens, nor does it consider the positions of other sub-accounts across the protocol.

Impact

Failure to account for the current circulating supply may lead to over-minting of stable tokens. This over-minting can result in inflation of the stable token supply, cause discrepancies in accounting, and introduce significant risks to the protocol's stability and the integrity of the stable token's value.

Recommendation

It is recommended to revise the minting logic so that the total amount of stable tokens to be minted is determined by the aggregate USD value of user positions across all sub-accounts, minus the current circulating supply of the stable token. Implementing this adjustment will ensure that the supply of stable tokens accurately corresponds to the underlying assets, thereby preventing over-minting, maintaining the intended value peg, and safeguarding the overall integrity of the system.

Mitigation Review Log

Fixed in commit 1f6525bec9017115621240d617a112558477b794.

4.3 Lack of Ownership Verification in SpotMarketMap and PerpMarketMap Load Functions

Severity: High	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

The functions `SpotMarketMap::load` and `PerpMarketMap::load` do not verify the ownership of the account being loaded. This omission allows arbitrary accounts to be interpreted as valid market maps without ensuring they were created or owned by the expected program.

Impact

An attacker could supply a fake or maliciously crafted `SpotMarketMap` or `PerpMarketMap` account that mimics the expected structure but contains invalid or manipulated data. This could lead to incorrect program behavior, unauthorized access to market-related operations, or potentially financial loss due to interaction with invalid market configurations.

Recommendation

Add an explicit check to verify the owner of the account passed to `SpotMarketMap::load` and `PerpMarketMap::load` . The account's owner should be compared against the expected program ID to ensure it was created and is managed by the correct on-chain program.

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.4 Unvalidated Strategy Identifier Facilitates Cross-Strategy Access Control Bypass

Severity: High	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

```rust
pub fn update_role_holder_strategy(
    ...
    strategy_id: u8
) -> Result<()> {
        ..
        Update::Add => {
            // Add role to the user.
            update_admin_permissions.add_strategy_role(strategy_id,
 ↪  role)?;
                                ...
        },
        Update::Remove =>{
            // Remove role from the user.
            update_admin_permissions.remove_strategy_role( strategy_id,
 ↪  role)?;
                                ...
        }
```

programs/reflect-main/src/instructions/admin/access/role_holder_-
update.rs#L43-L57

In the `update_role_holder_strategy` instruction, the input argument `strategy_id` is utilized directly without verifying whether it matches with `strategy.index` . This lack of validation weakens the binding between the strategy holder and the strategy itself.

Impact

As a result, a user who possesses the `UpdateRole` privilege for one strategy may exploit this vulnerability to modify or remove roles in any other strategy, bypassing the intended access controls. This could potentially lead to unauthorized privilege escalation or manipulation of roles across unrelated strategies.

Recommendation

It is recommended to implement a check to ensure that the provided `strategy_id` argument matches the `strategy.index` of the relevant strategy account. This validation will enforce proper authorization and prevent users from modifying roles outside their designated strategy scope.

Mitigation Review Log

Fixed in commit 5439f7c022c57f6965432428f2aa1df58e83c816.

4.5 Incorrect Attenuated Output Calculation Relative to Fill Rate

Severity: High	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

The `attenuate_output` function is responsible for splitting an output SPL token amount into two components: the newly attenuated amount and the leftover amount. This mechanism is intended to address market inefficiencies by adjusting token distribution according to current market conditions. However, the current implementation produces a counter-intuitive effect. When the `fill_rate` parameter increases, which indicates improved market efficiency, the leftover amount also increases while the attenuated amount available to the user decreases. As a result, users receive fewer tokens as market efficiency rises, which may not be consistent with the intended design of the attenuation mechanism.

Impact

When the `fill_rate` is high, users may receive a significantly reduced amount of tokens. This behavior can lead to user dissatisfaction and may be perceived as unfair or as an unintended penalty.

Recommendation

It's recommended to revise the formula in the `attenuate_output` function to ensure that the attenuation correctly reflects the intended market mechanism.

Mitigation Review Log

Fixed in commit ec6923e2725929094602f1fa3a4363abb2e72269.

4.6 Front-Running Threat in Reward Distribution

Severity: High	Status: Fixed
Target: Smart Contract	Category: MEV Risk

Description

For the S1 strategy, the `capture_drift_s1` function is responsible for capturing the yield and incrementing the share price accordingly. However, there exists a potential economic attack where a malicious actor can front-run the reward deposit operation. Specifically, an attacker could make a substantial deposit immediately before the yield is captured, thereby increasing their shareholding, and then withdraw shortly after the yield has been harvested. This sequence allows the attacker to disproportionately capture a significant portion of the reward distribution, with minimal exposure or risk. Such an attack exploits the timing of the reward mechanism and may disadvantage existing participants by diluting their reward share.

Impact

If successfully executed, this attack allows a malicious actor to extract a considerable amount of unearned yield from the protocol. This result can undermine the intended fairness of the reward distribution, leading to potential financial losses for honest participants and reducing the trustworthiness of the protocol. Repeated execution of such attacks could further result in significant value leakage from the system, disincentivizing legitimate deposits and threatening the protocol's long-term sustainability.

Mitigation Review Log

Fixed in commit 1f6525bec9017115621240d617a112558477b794.

4.7 Inconsistent Stable Token Valuation Between Mint and Redeem Operations

Severity: Medium	**Status: Fixed**
Target: Smart Contract	**Category: Logic Error**

Description

```rust
let price: i64 =
    oracle_map.get_price_data(&spot_market.oracle_id())?.price;

// Calculate the value of this position.
let amount: u128 = get_token_amount(
    spot_position.scaled_balance.cast()?,
    &spot_market,
    &spot_position.balance_type,
)?;

Ok(calculate_base_asset_value(amount, &price,
    spot_market.get_precision().cast()?)?)
```

programs/reflect-main/src/common/drift/drift_helpers.rs#L78-L87

In the `mint_drift_s1` instruction, the amount of stable tokens minted is determined by the difference in the result of the `calculate_total_usdc_value` function before and after depositing into Drift. The `calculate_total_usdc_value` function calculates the

USD value of the user's USDC position. This implies that the value of stable tokens is 1:1 with USD.

```rust
let withdrawn: u64 = accounts.controller_usdc_ata.amount.safe_sub
    ↪ (controller_balance_start)?;

// Burn equivalent amount of stable.
accounts.usdc_controller.base_strategy.burn(
    withdrawn,
    &accounts.stable_mint,
    &accounts.user_reflect_ata,
    &accounts.user,
    &accounts.token_program)?;
```

programs/reflect-main/src/instructions/user/drift/s1/redeem_drift_s1.rs#L65-L73

However, in the `redeem_drift_s1` instruction, the amount of stable tokens burned corresponds directly to the amount of USDC withdrawn. This suggests that the value of the stable tokens is 1:1 with USDC.

Impact

This discrepancy leads to an inconsistency in the value of stable tokens, as their value is treated differently in minting and redeeming operations—1:1 with USD in the minting process and 1:1 with USDC in the redemption process.

Recommendation

To ensure consistency in the value of stable tokens across all operations, it is recommended that both the minting and redemption processes treat stable tokens in the same way.

Mitigation Review Log

Fixed in commit 25d5301315dd52e6d054586dfcbf9290606b33a4.

4.8 Insufficient User Account Validation in capture_spread_drift Instruction

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

```rust
// Deserialise user account.
let user_positions: UserPositions =
    get_user_positions(&accounts.user_account, 0)?;
```

programs/reflect-main/src/instructions/admin/dex/drift/capture_spread_-drift.rs#L32-L33

In the `capture_spread_drift` instruction, user positions are extracted from the provided `user_account` without proper validation. Any account owned by the Drift program is currently considered valid, regardless of whether it belongs to the intended user or not.

Impact

An attacker with access to the `Action::Capture` function may exploit this vulnerability by invoking the `capture_spread_drift` instruction with a malicious `user_account`. This could result in the manipulation of the `total_usdc_value`, leading to unintended over-minting of stable tokens. Consequently, this could cause the excessive issuance of tokens to recipients. If the attacker gains control over any recipient accounts, they could redeem the tokens, leading to potential fund loss.

Recommendation

It is strongly recommended to implement proper validation on the `user_account` within the `capture_spread_drift` instruction. This validation should include checking the account's ownership and ensuring that it is an authorized `user_account` within the system before proceeding with the operation.

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.9 Lack of Permission Check in add_lst_drift Instruction

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Access Control

Description

The `add_lst_drift` instruction is lacking a necessary permission check.

Impact

The absence of a permission check allows anyone to add LST to the strategy, potentially enabling unauthorized interactions with the system.

Recommendation

It is recommended to implement a permission check to ensure that only actors with the `Action::AddSpl` permission are authorized to execute this instruction.

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.10 Insufficient Permission Verification in capture_spread_drift

Severity: Medium | **Status: Fixed**

Target: Smart Contract | **Category: Access Control**

Description

```rust
#[account(constraint = drift.key == &drift_program::ID
    @ReflectErrorCodes::IncorrectProgramSupplied)]
pub admin_permissions: Option<Account<'info, UserPermissions>>,
```

programs/reflect-main/src/instructions/admin/dex/drift/capture_spread_-drift.rs#L89-L90

In the `capture_spread_drift` instruction, the `Action::Capture` permission is verified against the `admin_permissions` credentials. However, there is no validation to ensure that `admin_permissions` is associated with the capturer.

Impact

An unauthorized user could potentially exploit the `admin_permissions` of an admin to invoke this instruction, thereby gaining unauthorized access or control.

Recommendation

It is recommended to implement a verification step to ensure that the `admin_permissions` credentials are correctly derived from the capturer. This measure will prevent unauthorized users from misusing admin privileges and invoking the instruction without proper authorization.

Mitigation Review Log

Fixed in commit 5439f7c022c57f6965432428f2aa1df58e83c816.

4.11 Type Mismatch in CPI Parameters for place_and_take_perp_order

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `update_perp_position` function, there is a CPI call to Drift's `place_and_take_perp_order` instruction. This call passes two arguments: the order parameters and the market index. However, according to the definition of the instruction, the second argument should be `success_condition`, which is of a different type (`Option<u32>`).

```rust
pub fn place_and_take_perp_order<'c: 'info, 'info>(
    ctx: Context<'_, '_, 'c, 'info, PlaceAndTake<'info>>,
    params: OrderParams,
    success_condition: Option<u32>,
) -> Result<()> {
    handle_place_and_take_perp_order(ctx, params, success_condition)
}
```

programs/drift/src/lib.rs#L224-L230

Impact

This discrepancy in argument types may result in runtime errors or unintended behavior due to type mismatches, leading to failed instruction execution or undetermined program behaviors.

Recommendation

Ensure that the arguments provided in the CPI call to Drift's `place_and_take_perp_order` strictly match the types expected by the instruction definition.

Mitigation Review Log

Fixed in commit bec807ae986187ae10df50fdcb91683f6dd0a64f.

4.12 Partial Consideration of User Accounts in capture_spread_drift

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `capture_spread_drift` instruction, the strategy attempts to mint additional stable tokens based on the discrepancy between the value of a user's spot positions and the current supply of stable tokens. However, a strategy may be associated with multiple user accounts, yet only a single `user_account` is considered during the execution of this instruction. This may result in an inaccurate calculation or unintended minting behavior, as the valuation and supply synchronization do not fully account for all user accounts used in the strategy.

```rust
let total_usdc_value: u64 = u64::try_from(collateral_value(
    &user_positions.extract_spots_inc_usdc(),
    &spot_market_map,
    &mut oracle_map,
)?).unwrap();

match total_usdc_value.safe_sub(accounts.stable_mint.supply) {
```

programs/reflect-main/src/instructions/admin/dex/drift/capture_spread_-drift.rs#L47-L53

Impact

The failure to consider all related user accounts for a given strategy can lead to several issues, including miscalculation of the token supply adjustments, potential under-minting of stable tokens, DoS on the `capture_spread_drift` instruction.

Recommendation

It is recommended to enhance the implementation by iterating over all user accounts associated with the strategy when calculating the total value of user spot positions.

Mitigation Review Log

Fixed in commit 47c446b851c5a6c62dd2d1d2814856fd41195376.

4.13 Stable Token Over-Minting from Excluding Unrealized Perpetual Losses

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `capture_spread_drift` instruction, the strategy attempts to mint additional stable tokens based on the discrepancy between the value of a user's spot positions and the current supply of stable tokens. However, if a user holds perpetual positions with non-zero PnL, the value of the user's spot positions alone does not accurately reflect their total portfolio value. Specifically, unrealized losses in the perp positions are not accounted for, leading to an incorrect calculation of the user's net equity.

Impact

If a user's perp positions have negative PnL, the calculated value of the user's spot positions will be overestimated. This may result in the protocol minting an excessive amount of stable tokens to the recipients, which can cause inconsistencies in the protocol's accounting and may pose risks to the overall system stability.

Recommendation

It's recommended to perform a CPI to the settle instruction in order to realize any outstanding PnL from perpetual positions before proceeding with the minting process. Alternatively, the system should check the user's unrealized PnL prior to minting; if the PnL is non-zero, the operation should be halted and an appropriate error should be raised.

Mitigation Review Log

Fixed in commit 47c446b851c5a6c62dd2d1d2814856fd41195376.

4.14 Improper Action Permission Check in update_action_role_protocol

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Access Control

Description

```rust
let creds: &mut Account<UserPermissions> = &mut
    accounts.admin_permissions;
action_check_protocol(action, Some(&creds), &main.access_control)?;
```

programs/reflect-main/src/instructions/admin/access/action_update.rs
#L15-L16

In the `update_action_role_protocol` instruction, the function `action_check_protocol` is invoked to verify that the actor possesses the necessary permission for the specified `action`. However, currently, the system only permits actors with the `Action::UpdateRole` privilege to execute modifications to role actions.

Impact

In the absence of additional access control measures or restrictions, unauthorized actors who do not possess the `Action::UpdateRole` privilege could gain the ability to perform role modifications. This could result in unauthorized changes to user roles, which may compromise the security and integrity of the system.

Recommendation

It is advised to enhance the permission-checking mechanism by ensuring that only actors with the `Action::UpdateRole` privilege are allowed to perform actions related to role modifications.

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.15 Insufficient Mint Account Validation in update_cap Instruction

Severity: Low | **Status: Fixed**

Target: Smart Contract | **Category: Data Validation**

Description

```
let mint_info: &AccountInfo<'_> = &ctx.remaining_accounts[0];

// Extract mint from remaining.
let mint_struct: Mint = get_mint_from_account(mint_info)?;
```

programs/reflect-main/src/instructions/admin/strategy/update_cap.rs
#L11-L14

In the `update_cap` instruction, the provided mint account is not properly validated. As a result, any arbitrary token mint account can be passed to the instruction without restriction.

Impact

Due to the lack of proper validation, the total supply retrieved from the mint account may not correspond to the intended token associated with the strategy. Consequently, any assertions or logic relying on an accurate comparison between the new cap and the current token supply may be unreliable or invalid, potentially allowing unintended behavior or cap manipulation.

Recommendation

Implement a strict check to ensure that the mint account provided in the instruction matches the expected mint address defined in `strategy.mint` .

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.16 Incorrect Permission Check Scope in update_action_role_strategy Instruction

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Access Control

Description

```rust
action_check_protocol(
    Action::UpdateRole,
    Some(&accounts.admin_permissions),
    &accounts.main.access_control
)?;
```

programs/reflect-main/src/instructions/admin/access/action_update.rs #L52-L56

In the `update_action_role_strategy` instruction, the `action_check_protocol` function is called to ensure that the actor has permission for `Action::UpdateRole` . However, the `action_check_protocol` function is designed to check for protocol-level permissions rather than strategy-level permissions.

Impact

This discrepancy limits the scope of the action to protocol-level permissions, rather than allowing it to function within the appropriate strategy-level context. As a result, users or actors may be restricted from performing the intended action if they do not have protocol-level permissions, even though they may have the necessary strategy-level permissions.

Recommendation

It is recommended to replace the `action_check_protocol` function with `action_check_strategy` , which is designed to validate permissions at the strategy level. This change will ensure that the action is properly authorized based on strategy-specific permissions rather than broader protocol-level permissions.

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.17 Lack of Proper Validation for Spot Market in add_lst_drift

Severity: Low	**Status: Fixed**
Target: Smart Contract	**Category: Logic Error**

Description

In the `add_lst_drift` instruction, the `spot_market_lst` is not properly validated, allowing any account to be passed. This lack of validation opens the possibility for malicious actors to manipulate the system.

Impact

The `market_index` and `lst_mint` values extracted from the `spot_market_lst` can be tampered with, leading to the potential addition of fake LST to the strategy. This could result in the introduction of malicious assets into the system, potentially compromising the integrity and stability of the protocol.

Recommendation

It is recommended to implement validation for the account owner and discriminator of the `spot_market_lst` account, ensure that the account belongs to a valid Drift spot market. This validation will safeguard against the potential manipulation of the market index and minting of fake LST tokens.

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.18 Incorrect Collateral Calculation Due to Borrowed LST Holdings

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

```rust
pub fn lst_collateral_value(&self) -> u64 {
    self.lsts.iter().map(|lst| lst.dollar_value).sum()
}
```

programs/reflect-main/src/state/dexes/drift/lst_controller/rebalance_-lst.rs#L130-L132

In the rebalancing flow, the collateral value of the holdings' LST is determined by summing the dollar value of each individual LST. However, this process does not account for the `BalanceType` of the LST holdings.

Impact

LST holdings designated with `BalanceType::BORROW` are currently incorrectly included as collateral. This results in an inflated collateral value, which can significantly impact risk calculations and lead to inaccurate system assessments. Such misrepresentation may undermine the financial stability and integrity of the protocol.

Recommendation

It is recommended to adjust the collateral calculation logic to subtract the dollar value of any holdings with `BalanceType::BORROW`, rather than adding them.

Mitigation Review Log

Fixed in commit 4fce070e39e6778f45050883d801ea594dd75da2.

4.19 Insufficient Mint Account Validation in capture_spread_drift Instruction

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

The `capture_spread_drift` instruction does not perform adequate validation on the `stable_mint` account. As a result, it accepts any token mint account, regardless of whether it is the intended one.

Impact

An attacker with `Action::Capture` privileges can invoke the `capture_spread_drift` instruction using an arbitrary or valueless mint address. The instruction may complete successfully, it will emit an incorrect `SpreadCapture` event, potentially leading to misleading on-chain data and downstream accounting inconsistencies.

Recommendation

Implement a strict check to ensure that the mint account provided in the instruction matches the expected mint address defined in `strategy.mint`.

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.20 Risk of Incorrect Sub-Account ID Due to Index Misalignment

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

```rust
// Find the index in DexSplBase that has corresponding main_spl_index.
let mut strategy_base_index: Option<usize> = None;
for (i, strategy_base) in
↪   self.base_strategy.spls.get_loaded_spls().iter().enumerate() {
    if main_spl_index == strategy_base.spl_main_index {
        strategy_base_index = Some(i);
        break;
    }
}

let strategy_base_index = strategy_base_index.ok_or(ReflectErrorCodes::
↪   StrategyDoesNotSupportSpl)?;

// Use this index to get the DriftLstData entry and return the
↪   sub_account_id.
Ok(self.lsts[strategy_base_index].sub_account_id)
```

programs/reflect-main/src/state/dexes/drift/lst_controller/lst_controller.rs#L159-L171

In the `mint_to_sub_account_address` method of `DriftLstController`, the variable `strategy_base_index` is used to directly index into `controller.lsts`. However, `strategy_base_index` represents the index within `controller.base_strategy.spls`, not necessarily the same index within `controller.lsts`.

Impact

If the ordering or structure of `controller.lsts` and `controller.base_strategy.spls` diverge, using `strategy_base_index` directly as an index into `controller.lsts` may lead to retrieval of an incorrect `sub_account_id`.

Recommendation

Instead of assuming the indices are aligned, it is recommended to iterate over `controller.lsts` and locate the entry where `lst.base_strategy_spl_index` matches `strategy_base_index`. This approach ensures correctness regardless of any differences in the structure or order of the underlying collections.

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.21 Incorrect Order of Attenuation and Slippery Check

Severity: Low	**Status: Fixed**
Target: Smart Contract	**Category: Logic Error**

Description

```
// Ensure that the redeemed amount is greater than the minimum amount.
slippage_pass(burn_details.usdr_amount, min_lst_redeem)?;

// Calculate attenuated amount of LST to withdraw to the user.
let (attenuated_withdraw, _) =
    accounts.lst_controller.base_strategy.attenuation.attenuate_strategy_output(lst_available, fill_ratio)?;
```

programs/reflect-main/src/instructions/user/drift/s3/redeem_drift_-s3.rs#L153-L157

In the `mint_drift_s3` and `redeem_drift_s3` functions, the slippery check is performed prior to the application of the attenuation factor.

Impact

As a result, users may receive fewer tokens than expected.

Recommendation

It is recommended to ensure that the attenuation factor is applied before the slippery check to avoid discrepancies in token distribution.

Mitigation Review Log

Fixed in commit b5e9b562af4472c93e72e86917f2e1ddf36741b9.

4.22 user_account_2 Not Properly Validated Against PDA Address

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

```rust
/// CHECK: Drift rejects if it not derived of the authority, stratgey
↪   controller.
#[account(mut, owner = drift.key(), constraint =
↪   get_reflect_user_pda(&usdc_controller.key(), 0) ==
↪   user_account_1.key() @ReflectErrorCodes::IncorrectUserAccount)]
pub user_account_2: AccountInfo<'info>,
```

programs/reflect-main/src/instructions/user/drift/s1/supply_management_-context_s1.rs#L73-L75

In the `SupplyManagementS1` module, `user_account_2` is expected to match the PDA address calculated by the `get_reflect_user_pda` function. However, the constraint check is currently only applied to `user_account_1`, rather than `user_account_2`.

Impact

This issue results in a lack of validation for `user_account_2`'s authenticity. This undermines the integrity of the system.

Recommendation

It is recommended to update the constraint to ensure that `user_account_2` is also validated against the correct derived PDA address.

Mitigation Review Log

Fixed in commit 0044a38fb0d58ebaa53bf48ded23fe9176bf6172.

4.23 Missing Validation of lst_mint and spot_market_lst.mint Consistency

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

The `lst_mint` account is not validated to ensure it matches `spot_market_lst.mint`. This absence of validation introduces a potential attack vector whereby malicious actors could exploit the system by providing an incorrect mint address.

Impact

Since the `spot_market_lst.mint` is not necessarily the same as `lst_mint`, the `spot_market_lst.market_index` may not correspond to the actual index of `lst_mint`. This discrepancy could result in the addition of an invalid or unintended LST to the strategy. Such inconsistencies may compromise the asset composition within the protocol, undermining the protocol's integrity and stability.

Recommendation

Implement strict validation to ensure that the `lst_mint` provided is identical to `spot_market_lst.mint`.

Mitigation Review Log

Fixed in commit e1fd2c0bdd6135eddc2add64835c5fdee155cddb.

4.24 Missing Strategy Controller Verification in suspend_lst_strategy

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Data Validation

Description

In the `suspend_lst_strategy` instruction, the strategy object extracted using the `extract_strategy` method is not subsequently validated with `verify_strategy_` `controller` .

Impact

If `verify_strategy_controller` is not called, there is a risk that an incorrect or unauthorized strategy account could be utilized. This could lead to unintended logic execution, security vulnerabilities, or inconsistent protocol state.

Recommendation

It is recommended to invoke `verify_strategy_controller` immediately after extracting the strategy. This ensures that the strategy account is properly validated before any further logic is executed, thereby strengthening the security and reliability of the instruction flow.

Mitigation Review Log

Fixed in commit 298db6b579efb5890b997f175f554e4ee95e0b5e.

4.25 Perpetual Order Amount Not Aligned with Protocol Increment Constraint

Severity: Low	**Status: Fixed**
Target: Smart Contract	**Category: Data Validation**

Description

Within the `process_swap_deposit` function, a perpetual order is submitted to maintain a delta-neutral strategy. However, the amount for this order is determined directly using `calculate_asset_amount` , without adjusting the order size to be a multiple of `MIN_DRIFT_POSITION_CHANGE` as required by the protocol.

Impact

If the order size does not conform to the required multiple of `MIN_DRIFT_POSITION_CHANGE` , the order may be rejected by the protocol or result in rounding errors. This may lead to failed transactions.

Recommendation

It’s recommended to update the calculation logic to ensure that the order amount is adjusted to the nearest multiple of `MIN_DRIFT_POSITION_CHANGE` before placing the order. Doing so will ensure compatibility with the protocol and improve the reliability and effectiveness of position management operations.

Mitigation Review Log

Fixed in commit 3f3d939fd40810e87d0a3188a5333d573a424752.

4.26 Missing PnL Settlement Call in handle_settlement Function

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `handle_settlement` function of the `process_swap_withdraw` instruction, the CPI required for PnL settlement is currently absent.

Impact

If the PnL settlement is not executed prior to withdrawal, the calculations used for rebalancing may be inaccurate. This increases the risk of inconsistent accounting, potential misallocation of assets, or unintended financial imbalances within the protocol.

Recommendation

It is strongly recommended to implement the necessary CPI to the PnL settlement program within the `handle_settlement` function before any withdrawal processes are carried out. Ensuring that all profit and loss are settled before withdrawals will maintain the integrity of the protocol's accounting and help prevent discrepancies in system balances.

Mitigation Review Log

Fixed in commit b5e9b562af4472c93e72e86917f2e1ddf36741b9.

4.27 Inconsistent Minimum Position Validation in min_position_perp Function

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

```rust
if is_precise_number::<T>() {
    ...
    let min_pos =
        PreciseNumber::new(MIN_DRIFT_POSITION as
        ↪   u128).ok_or(error!(ConversionFailed))?;
    ...
    match truncated.less_than(&min_pos) {
        true => Err(BelowMinPosition.into()),
        false => Ok(unsafe { std::mem::transmute_copy::<PreciseNumber,
        ↪   T>(&truncated) }),
    }
} else if is_u64::<T>() {
    ...
    match truncated < MIN_DRIFT_POSITION_CHANGE {
        true => Err(BelowMinPositionChange.into()),
        false => Ok(unsafe { std::mem::transmute_copy::<u64,
        ↪   T>(&truncated) }),
    }
```

programs/reflect-main/src/common/math/conversion.rs#L114-L128

In the `min_position_perp` function, the input parameter `desired_position` is truncated to the nearest multiple of `MIN_DRIFT_POSITION_CHANGE` to enforce position granularity. The implementation is intended to ensure that the resultant value is also not smaller than `MIN_DRIFT_POSITION` . However, when the generic type `T` is specified as `u64` , the truncated value is compared against `MIN_DRIFT_POSITION_CHANGE` rather than the expected `MIN_DRIFT_POSITION` .

Impact

This inconsistency in the validation check can result in the acceptance of position sizes that fall below the intended minimum threshold.

Recommendation

It is recommended to update the comparison to consistently use `MIN_DRIFT_POSITION` for all type specializations to enforce minimum position requirements correctly.

Mitigation Review Log

Fixed in commit 43c0089a323ad89758762ec9d90b6d22a8754f1e.

4.28 LST Token Residue from Unmatched Deposit Amounts

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

During the process of minting stable tokens, users are required to transfer LST tokens to the strategy controller's ATA. Subsequently, the strategy controller attempts to deposit these LST tokens into Drift protocol to manage positions or provide collateral.

However, within Drift protocol's `deposit` instruction, the actual amount deposited into a user's Drift account may be less than the input `amount` parameter.

```
let amount = if (force_reduce_only || reduce_only)
    && user.spot_positions[position_index].balance_type ==
    SpotBalanceType::Borrow
{
    user.spot_positions[position_index]
        .get_token_amount(&spot_market)?
        .cast::<u64>()?
        .min(amount)
} else {
    amount
};
```

programs/drift/src/instructions/user.rs#L564-L574

Impact

A discrepancy between the `amount` parameter specified in the deposit instruction and the actual amount deposited may cause residual LST tokens to remain in the strategy controller's ATA. Consequently, the user may receive fewer stable tokens than expected, while the surplus LST tokens are not returned automatically.

Recommendation

It is recommended to implement logic that returns any residual LST tokens to the user's account in cases where the entire amount cannot be deposited into the Drift protocol. This will ensure that no excess tokens are unintentionally retained in the strategy controller.

Mitigation Review Log

Fixed in commit 7c7ad7789b1a674ee237e3a88ac00fbaa154a0a3.

4.29 Improper Valuation Basis in capture_spread_drift Instruction

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

In the `capture_spread_drift` instruction, the strategy attempts to mint additional stable tokens based on the discrepancy between the value of a user's spot positions and the current supply of stable tokens. However, in the case of the USDC strategy, the stable token is pegged to USDC, rather than to USD. This approach assumes parity between USDC and USD, which may not hold due.

Impact

If the value of USDC deviates from USD, the minting mechanism may lead to an inaccurate supply of stable tokens relative to the intended USD value.

Recommendation

It is recommended to implement separate instructions for capturing spread based on the specific pegging mechanism of each strategy.

Mitigation Review Log

Fixed in commit 47c446b851c5a6c62dd2d1d2814856fd41195376.

4.30 USDC Collateral Movement Uses Dollar Value Instead of Token Amount

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

```
usdc: ShiftCollateral {
    amount: collateral.usdc.dollar_value,
    mint: USDC_MINT,
    direction: DirectionCollateral::BUY,
},
```

programs/reflect-main/src/common/math/rebalance.rs#L112-L116

In the `rebalance_math` function, the value assigned to the `amount` field of the USDC collateral movement is set to `collateral.usdc.dollar_value`, which represents the dollar value of the collateral, instead of the actual amount of USDC tokens held.

Impact

Assigning the dollar value rather than the token amount may cause inconsistencies between the intended and actual movement of collateral. This could potentially introduce calculation inaccuracies in collateral management and protocol accounting.

Recommendation

It is recommended to ensure that the `amount` field consistently reflects the actual token amount of USDC.

Mitigation Review Log

Fixed in commit f17ebe8ea7db7e1b57610e0b079809fa80159733.

4.31 Redemption at Stale Share Price After Pool Value Drop

Severity: Low	**Status: Fixed**
Target: Smart Contract	**Category: Logic Error**

Description

```rust
pub fn calculate_capturable_yield(&self, current_value: u64) ->
    Result<u64> {
    ...
    Ok(current_value.saturating_sub(expected_value))
}
```

programs/reflect-main/src/state/components/autocompound.rs#L39-L42

In the S1 strategy, if there is a short-term decline in the pool value, the `calculate_capturable_yield` function will return 0 due to the use of `saturating_sub`. As a result, the `deposited_vault_value` does not decrease in tandem with the underlying pool value. Consequently, users who initiate a redemption during such a period are able to redeem at the previous, higher price, rather than the actual current value.

Impact

This mechanism may lead to unfair advantages for users redeeming during short-term price declines. Specifically, they may receive more value than the current underlying assets justify, effectively causing a value leakage from the protocol or other users.

Mitigation Review Log

Fixed in commit 1ac6afd69d5bfa908a12fe598ecd3ed244dcbcd9.

4.32 Incorrect Calculation of Residual USDC in mint_drift_s1 and mint_drift_s3 Instructions

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

```
accounts.controller_usdc_ata.reload()?;
let usdc_residue: u64 = usdc_balance_controller_start -
↪  accounts.controller_usdc_ata.amount;

// If there is a residual balance, transfer it back to the user.
if usdc_residue > 0 {
    accounts.usdc_controller.base_strategy.withdraw(
        usdc_residue,
        ...
    )?;
}
```

programs/reflect-main/src/instructions/user/drift/s1/mint_drift_s1.rs
#L105-L114

The `mint_drift_s1` instruction attempts to transfer any remaining USDC back to the user. The calculation for the residual amount is performed as `usdc_balance_controller_start - accounts.controller_usdc_ata.amount` . However, this logic is incorrect, the correct formula should be `accounts.controller_usdc_ata.amount - usdc_balance_controller_start` . The same issue is observed in the `mint_drift_s3` instruction.

Impact

This flaw can result in the entire instruction failing with a runtime error due to unsigned integer underflow.

Recommendation

Correct the calculation for the residual USDC as follows, to prevent arithmetic underflow and ensure correct behavior.

Mitigation Review Log

Fixed in commit fa9fc49532b50191888d938741cba2f81ae0348c.

4.33 Inaccurate Account Size Calculation

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

```rust
pub fn calculate_strategy_space(num_strategy_roles: usize) -> usize {
    let vec_overhead = size_of::<usize>(); // Vec length field
    let strategy_entry_size = size_of::<StrategyRoleEntry>();
    vec_overhead + (num_strategy_roles * strategy_entry_size)
}
```

programs/reflect-main/src/state/credentials.rs#L209-L213

The current implementation for calculating the account size of `UserPermissions` omits the consideration of the `protocol_roles` field, potentially resulting in insufficient account allocation. Furthermore, the use of Rust's native `size_of` function is not appropriate in this context, as the output does not correspond to the size that the Anchor framework uses during serialization. In Anchor, serialized sizes may differ due to data representation and padding rules.

To ensure consistent and accurate account size calculation, it is recommended to leverage Anchor's `InitSpace` macro, which provides a reliable way to determine the required space for account initialization based on Anchor serialization.

Impact

Inaccurate account size calculation can result in account reallocations or initialization failures, potentially leading to unusable accounts, loss of funds, or unintended protocol behavior. This issue might compromise the integrity or operability of user accounts due to unexpected serialization layout differences.

Recommendation

It is recommended to use Anchor's `InitSpace` macro to ensure accurate calculation of the required account size.

Mitigation Review Log

Fixed in commit 6e9d8b6233fab847089c263c6f6ca4c95c98fa65.

4.34 Missing Last Active Slot Updates During Reflect-Drift Interactions

Severity: Low	Status: Acknowledged
Target: Smart Contract	Category: Logic Error

Description

The `controller.base_drift.last_active_slot` records the last slot in which the strategy interacts with the Drift program. However, this value is only updated in the instructions `redeem_drift_s1` , `mint_drift_s3` , and `redeem_drift_s3` .

Impact

```
if can_chill {
    usdc_controller.base_drift.assert_no_fee(current_slot)?;
}
```

programs/reflect-main/src/instructions/user/drift/s1/redeem_drift_s1.rs #L34-L36

After the Reflect protocol interacts with Drift during the `redeem_drift_s1` , `process_swap_deposit` , and `process_swap_withdraw` processes, the `last_active_slot` is not updated on the Reflect side, but is updated on the Drift side. As a result, the `can_chill` check on the Reflect side may return an incorrect result.

Recommendation

It is recommended to ensure that the `last_active_slot` is consistently updated each time the Reflect protocol interacts with Drift, in order to maintain consistency between Reflect and Drift.

Mitigation Review Log

Reflect Team: Just updated the active slot inside settle_pnl. For process_swap_deposit and process_swap_withdraw, its fine to ignore it for now. Code as is will not be used for strategy 3, needs some rewrite in few places. It would be much easier to read it of drift user account over storing it anyway.

4.35 Informational and Undetermined Issues

Incorrect Strategy Index in Issue Event Emission

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Logic Error

In the `mint_drift_s1` function, the `Issue` event is emitted with an incorrect assignment of `strategy_idx` . Currently, it is assigned to `LST_STRATEGY` , but it should instead be assigned to `USDC_STRATEGY` .

Fixed in commit 25d5301315dd52e6d054586dfcbf9290606b33a4.

Incorrect Stable Amount in Issue Event Emission

Severity: Informational **Status: Fixed**

Target: Smart Contract **Category: Logic Error**

In the `mint_drift_s1` function, the `Issue` event is emitted with an incorrect assignment of the `stable_amount` . Currently, it is being assigned to the user-provided expected amount, `usdc_input` . However, it should instead reflect the actual minted stable token amount, which is `deposits_value` .

Fixed in commit 3d724c55dcb00e046abfa844aa404b8915f10a9b.

Lack of Validation for address Alignment with update_admin_permissions

Severity: Informational **Status: Fixed**

Target: Smart Contract **Category: Data Validation**

In the `update_role_holder_protocol` and `update_role_holder_strategy` instructions, the input address argument is emitted directly without validating whether it aligns with the `update_admin_permissions` . It is essential to ensure that `update_admin_permissions` is derived from the given address in order to prevent the emission of incorrect events.

Fixed in commit ccf458f5a0a9818073ef25d09d98f10a11c0be99.

Unchecked strategy_id Argument in Event Logging

Severity: Informational **Status: Fixed**

Target: Smart Contract **Category: Data Validation**

In the `update_action_role_strategy` instruction, the input `strategy_id` argument is emitted directly without validating whether it matches `strategy.index` . This could potentially result in the emission of incorrect or misleading events.

Fixed in commit ccf458f5a0a9818073ef25d09d98f10a11c0be99.

Redundant Calculation of dollar_value_trunc in mint_math Function

Severity: Informational **Status: Fixed**

Target: Smart Contract **Category: Optimization**

```
    // Calculate dollar value of the truncated perp size.
    let dollar_value_trunc: PreciseNumber =
        dollar_value(&truncated_perp_size, &price_sol, SOL_DECIMALS)?;
        ...
    // Calculate dollar value of the truncated perp size.
    let dollar_value_trunc: PreciseNumber =
        dollar_value(&truncated_perp_size, &price_sol, SOL_DECIMALS)?;
```

programs/reflect-main/src/common/math/conversion.rs#L315-L321

In the `mint_math` function, the value of `dollar_value_trunc` is calculated twice using the same expression. This results in redundant computation and leads to unnecessary consumption of compute units.

Fixed in commit ccf458f5a0a9818073ef25d09d98f10a11c0be99.

Improper USDC Transfer Amount Leads to Dust in Strategy Controller Account

Severity: Informational | **Status: Fixed**

Target: Smart Contract | **Category: Precision Issue**

In the `mint_drift_s1` instruction, the user transfers an amount of USDC, referred to as `usdc_input`, to the strategy controller's ATA. Subsequently, half of this amount is deposited into Drift in two separate CPIs. If `usdc_input` is an odd number, this process will leave one USDC token as residual dust in the strategy controller's ATA. To avoid the accumulation of such residual tokens, it is recommended to transfer exactly `2 * half_deposit` USDC tokens to the strategy controller's ATA. This approach ensures that no unintended tokens are left behind.

Fixed in commit 8736240351e291aaa9c88a3f14aaa563d35e41f9.

Improper Validation of user_lst_ata Mint Constraint

Severity: Informational | **Status: Fixed**

Target: Smart Contract | **Category: Data Validation**

```
    #[account(
        mut,
        token::authority = user,
        constraint =
↪   lst_controller.base_strategy.mint_accepted_strategy(&main.lsts_main,
↪   &controller_lst_ata.mint)
↪   @ReflectErrorCodes::StrategyDoesNotSupportSpl
      )]
    pub user_lst_ata: Box<Account<'info, TokenAccount>>,
```

programs/reflect-main/src/instructions/user/drift/s3/supply_management_-context_s3.rs#L26-L31

In the `SupplyManagement` struct, the constraint applied to `user_lst_ata` references `controller_lst_ata`, which is incorrect. The constraint should reference the mint of `user_lst_ata` to ensure that the correct token account is being validated against the accepted strategy.

Fixed in commit ccf458f5a0a9818073ef25d09d98f10a11c0be99.

Redundant Suspension State Check in mint_drift_s3

Severity: Informational | **Status: Fixed**

Target: Smart Contract | **Category: Optimization**

In the `mint_drift_s3` instruction, the suspended state of `controller_lst_ata.mint` is checked multiple times:

```
    // Deals with both strategy and main suspension.
    accounts.lst_controller.base_strategy.validate_deposit_mint(
        &accounts.main.lsts_main,
        &accounts.controller_lst_ata.mint
    )?;
```

programs/reflect-main/src/instructions/user/drift/s3/mint_drift_s3.rs #L16-L20

```
    // Ensure minting is not restricted for this LST token for drift.
    require!(!base_data.deposits_suspended,
    ↪   ReflectErrorCodes::SplMintFrozen);
```

programs/reflect-main/src/instructions/user/drift/s3/mint_drift_s3.rs #L67-L68

The first check within `validate_deposit_mint` already verifies the suspension state. The subsequent direct requirement on `base_data.deposits_suspended` is therefore redundant and does not contribute to additional safety or correctness.

Fixed in commit 99ff9e1e9b5205e2d3f4c3ed00ab7c0c11493217.

Missing Validation of Drift Program Activation in create_user_stats_account

Severity: Informational | **Status: Fixed**

Target: Smart Contract | **Category: Data Validation**

In the `create_user_stats_account` instruction, the activation state of the Drift program is not currently validated. It is recommended to perform this verification prior to invoking the Drift program in order to maintain consistency with other components of the Reflect contract.

Fixed in commit ccf458f5a0a9818073ef25d09d98f10a11c0be99.

Inconsistent Placement of Drift Program ID Verification

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Code QA

```
/// CHECK: Account must be owned by drift, otherwise execution fails.
#[account(mut, constraint = drift.key() == drift_program::id())]
pub state: AccountInfo<'info>,
```

programs/reflect-main/src/instructions/admin/dex/drift/deposit_-drift.rs#L164-L166

```
/// CHECK: Account must be owned by drift, otherwise execution fails.
#[account(mut, constraint = drift.key == &drift_program::ID
↪   @ReflectErrorCodes::IncorrectProgramSupplied)]
pub state: AccountInfo<'info>,
```

programs/reflect-main/src/instructions/user/drift/s1/supply_management_-context_s1.rs#L54-L56

```
#[account(constraint = drift.key == &drift_program::ID
↪   @ReflectErrorCodes::IncorrectProgramSupplied)]
pub admin_permissions: Option<Account<'info, UserPermissions>>,
```

programs/reflect-main/src/instructions/admin/dex/drift/capture_spread_-drift.rs#L89-L90

The verification of the Drift program ID is at times implemented in inconsistent or non-standard locations within the codebase. Instead of performing this check directly at the account definition, which is both the recommended and conventional method, the verification is sometimes conducted elsewhere or in fragmented sections of the program logic. Such decentralization reduces code clarity and may weaken security guarantees, as these critical checks are more susceptible to being inadvertently overlooked or bypassed during subsequent code maintenance or upgrades. To align with established best practices, it is strongly recommended that all account ownership verifications be explicitly enforced at the account definition level.

Fixed in commit ccf458f5a0a9818073ef25d09d98f10a11c0be99.

Redundant Invocation of insert_strategy in update_role_holder_strategy

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Optimization

At the conclusion of the `update_role_holder_strategy` instruction, the `insert_strategy` function is invoked to store the strategy object. However, throughout the execution of this instruction, the strategy instance is never actually modified. As a result, calling `insert_strategy` at this point is redundant and has no effect on the overall state.

Fixed in commit ccf458f5a0a9818073ef25d09d98f10a11c0be99.

Unnecessary Account Requirements in Instruction Definitions

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Optimization

In certain instructions, some accounts are unnecessarily required, including:

- `admin_permissions` in `settle_pnl` / `settle_pnl_multi`
- `spot_market_usdc` in `init_controller_s1`
- `main` in `create_admin_account`
- `user_account` in `create_user_stats_account`
- `usdc_market_vault` in `mint_drift_s1` / `redeem_drift_s1`
- `usdc_market_vault` in `process_swap_withdraw` / `process_swap_deposit`
- `strategy` in `update_role_holder_protocol`

Requiring these extraneous accounts not only increases transaction size but also introduces additional complexity and potential surface areas for error or misuse. It is generally recommended to limit account requirements to only those strictly necessary for the operation being performed. Including unused or unnecessary accounts may also confuse integrators and auditors, making the code less maintainable and auditable. It's recommended to remove any accounts that are not explicitly utilized by the program logic. This will enhance both efficiency and security.

Fixed in commit ccf458f5a0a9818073ef25d09d98f10a11c0be99.

Redundant Checks in ActionMapping.add_role

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Code QA

In `ActionMapping.add_role` , there is a check to verify if `ActionMapping.role_count` exceeds `MAX_ROLES` , but this check is performed twice. It is recommended to keep only one of these two checks.

Fixed in commit 7c7514f74f7d5cbce6e288283499dc31ceeccee5.

Semantic Inconsistency Between Comments and Implementation in KillSwitch

Severity: Informational	Status: Fixed
Target: Smart Contract	Category: Code QA

In the definition of `KillSwitch` , here is the comments to explain the semantics corresponding to the `frozen` value.

```rust
pub struct KillSwitch {
    /** Bool indices:
    - [0] - mint
    - [1] - redeem
    - [2] - rebalance
    - [3] - capture (print and distribute stable)
     */
    pub frozen: u8,
}
```

programs/reflect-main/src/state/components/killswitch.rs#L9-L17

However, when setting the `frozen` value, a left-shift operation is applied.

```rust
    pub fn freeze(&mut self, action: &Action) {
        let mask = 1u8 << (*action as u8);
        self.frozen |= mask;
    }
```

programs/reflect-main/src/state/components/killswitch.rs#L45-L48

The `action` values are defined as follows:

```rust
pub enum Action {
    #[default]
    // Core actions
    /** Mints stable. */
    Mint = 0,
    /** Redeems input spl. */
    Redeem = 2,
    /** Rebalances delta-neutral-position */
    Rebalance = 4,
    /** Captures the protocol yield, and distributes it to stakeholders.
     ↪  */
    Capture = 6,
```

programs/reflect-main/src/state/components/action.rs#L7-L17

The actual value stored in `frozen` does not match the values described in the `KillSwitch` comments. It is recommended to update the comments or adjust the values of the `Action` enum to ensure consistency between the documentation and the implementation.

Fixed in commit 54c8fd1d33d24dc37cfd3ff4535499b7be8a73d8.

Hardcoded Space Size

Severity: Informational	**Status: Fixed**
Target: Smart Contract	**Category: Code QA**

The following Accounts use hard-coded constant space sizes in the program during initialization.

- Main: 10_000
- UserPermissions: 10_000
- DriftUsdcController: 999
- DriftLstController: 999

For example, for `UserPermission`, if there are many `strategyRoleEntry` entries in the contract, or if future features cause the Account size to increase, it might cause the size of `UserPermission` Account to exceed this limit. It is recommended to ensure that these hard-coded sizes are greater than or equal to the theoretical maximum, or to add realloc-related operations for these Accounts.

Fixed in commit 6e9d8b6233fab847089c263c6f6ca4c95c98fa65.

Inconsistency Between Comments and Formula in Attenuation Parameters

Severity: Informational	**Status: Fixed**
Target: Smart Contract	**Category: Code QA**

According to the code comments for the `Attenuation` definition, `base_scaling` is described as fixed and `conditional_scaling` as variable. However, in the implementation of the attenuate_output function, it is base_scaling that is multiplied by `fill_ratio`, rather than `conditional_scaling`. It's recommended to reverse the comment or the formula implementation.

Fixed in commit 7c7514f74f7d5cbce6e288283499dc31ceeccee5.

5 Disclaimer

This audit report is provided for informational purposes only and is not intended to be used as investment advice. While we strive to thoroughly review and analyze the smart contracts in question, we must clarify that our services do not encompass an exhaustive security examination. Our audit aims to identify potential security vulnerabilities to the best of our ability, but it does not serve as a guarantee that the smart contracts are completely free from security risks.

We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster the security of their smart contracts.

It is important to note that the scope of our audit is limited to the areas outlined within our engagement and does not include every possible risk or vulnerability. Continuous security practices, including regular audits and monitoring, are essential for maintaining the security of smart contracts over time.

Please note: we are not liable for any security issues stemming from developer errors or misconfigurations at the time of contract deployment; we do not assume responsibility for any centralized governance risks within the project; we are not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By using this report, the client acknowledges the inherent limitations of the audit process and agrees that our firm shall not be held liable for any incidents that may occur subsequent to our engagement.

This report is considered null and void if the report (or any portion thereof) is altered in any manner.

OFFSIDE LABS

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs