Sec3™

Security Assessment Report

Carrot Protocol

August 12, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Carrot Protocol smart contracts.

The artifact of the audit was the source code of the following programs, excluding tests, in a private repository.

The initial audit focused on the following versions and revealed 14 issues or questions.

program	type	commit
Carrot Protocol	Solana	2670347a717bbfaa1293cadc661d1b3da3ee6c13

The post-audit review was conducted on the following versions to verify whether the reported issues had been addressed.

program	type	commit
Carrot Protocol	Solana	89b964af0eb5500d2b3bf4aede10e1c1979f87bf

This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
CARROT PROTOCOL		
[M-01] Potential arbitrage opportunities	Medium	Resolved
[M-02] Missing ATA check for vault asset account	Medium	Resolved
[M-03] Incorrect performance fee calculation	Medium	Resolved
[L-01] Potential outdated strategy balance	Low	Acknowledged
[L-02] Missing strategy duplication check	Low	Acknowledged
[L-03] Missing necessary checks before removing strategy	Low	Resolved
[L-04] Missing accounts consistency check between init and deposit/withdraw	Low	Resolved
[L-05] Inaccurate management fee calculation	Low	Resolved
[L-06] Arbitrary CPIs in solend init and withdraw operations	Low	Resolved
[I-01] Consider incorporating Pyth confidence interval	Info	Resolved
[I-02] Missing frozen authority check in init_vault	Info	Resolved
[I-03] Check asset decimal when adding assets	Info	Resolved
[I-04] Incorrect space calculation for Fee struct	Info	Resolved
[I-05] Inconsistent with klend implementation in klend_exchange_rate	Info	Resolved

Findings in Detail

CARROT PROTOCOL

[M-01] Potential arbitrage opportunities

Due to the architectural design of the project, keeper operations can impact the price of shares.

When the keeper performs withdraw or deposit actions, the corresponding strategy's balance is updated, increasing the interest and likely causing the share price to rise.

Conversely, when the keeper executes the distribute vault fees operation, it can lead to share inflation and devaluation.

If an arbitrageur can predict the keeper's actions, they could exploit this by performing opposite transactions before and after the keeper's actions to arbitrage or avoid some fees.

Additionally, under the current implementation, users can perform nearly lossless swaps through issuing and redeeming, with only minimal amounts lost due to rounding. Since the prices are based on EMA values, users might exploit the discrepancy between the EMA price and the actual market price to profit from almost cost-free swaps.

To reduce the potential for arbitrage, it is recommended to perform the distribute vault fees operation simultaneously with withdraw and deposit actions, and to consider introducing a withdrawal fee.

Resolution

The team has implemented the following four measures to mitigate this issue:

1. Ensuring that the distribution of vault fees is performed concurrently with withdrawal and deposit operations.

2. Introducing a redemption fee in commit `a045300ff2bf603bd6c9c49eeb7b5e6defa0f80d`.

3. Incorporating unminted fees into the share supply in commit 3e934ffec1741a002eb97e5f32cc55ff303778c6.
4. Adjusting prices based on the confidence interval provided by the price oracle in commit c70359330b0e528a1e85c0d1bff156fde27d5f40.

CARROT PROTOCOL

[M-02] Missing ATA check for vault asset account

When a user performs operations such as issue and redeem, the program only verifies that the owner of the "`vault_asset_ata`" is indeed the "`vault`" without ensuring that the "`vault_asset_ata`" is an Associated Token Account (ATA).

Since anyone can create a token account with any owner and mint, this allows malicious users to create new token accounts for the "`vault`" and transfer tokens into them. Consequently, tokens of a single mint may be stored in multiple token accounts within the vault, complicating subsequent withdrawal operations and effectively resulting in a Denial of Service (DoS) attack.

Additionally, when calculating TVL, only the balance in the token account set during the addition of the asset is considered. The lack of ATA verification may result in an underestimated TVL value, thereby affecting the price.

It is recommended to implement checks that validate "`vault_asset_ata`" as ATAs to mitigate this vulnerability.

Resolution

This issue has been resolved by commit 9a02965c9e0af35428bbeaff6b87cbc7dafd71a3.

CARROT PROTOCOL

[M-03] Incorrect performance fee calculation

The Carrot protocol collects a portion of the profits generated by the strategy as a performance fee. However, there are two issues in the calculation of the performance fee:

1. The "`calculate_performance_fee`" function returns the USD amount of the performance fee that should be collected, but this amount is subsequently used directly for minting shares without converting the USD amount to shares.

```
/* programs/carrot/src/state/data.rs */
// returns the shares amount of the performance fee that should be minted to the fee account
pub fn calculate_performance_fee(
    &self,
    net_earnings: i64,
    asset_price: i64,
    asset_price_expo: i32,
    asset_decimals: u8,
) -> u64 {
    // calculate performance fee in usd
    let fee_amount = self.calc_performance_fee(net_earnings_usd);

    fee_amount
}
```

2. When calculating the performance fee, the net earnings are directly multiplied by the performance_fee_bps without dividing by 10,000 to convert it into the correct unit.

```
/* programs/carrot/src/state/data.rs */
fn calc_performance_fee(&self, net_earnings_usd: u128) -> u64 {
    (net_earnings_usd * self.performance_fee_bps as u128)
        .try_into()
        .unwrap()
}
```

Resolution

This issue has been resolved by commits 9a02965c9e0af35428bbeaff6b87cbc7dafd71a3 and d8b367bf55656eecf625d8444b8248bcf548b604.

CARROT PROTOCOL

[L-01] Potential outdated strategy balance

When calculating TVL, the balance data recorded by each strategy is utilized. However, this balance field is only updated during deposit or withdrawal operations.

If a strategy operates infrequently, its balance field may not be updated in a timely manner, leading to an underestimation of the actual TVL.

It is recommended to introduce a crank instruction for updating the balance periodically to ensure accurate TVL calculations.

Resolution

The team acknowledged this finding and clarified that they have a crank that updates the balances through small deposit and withdrawal operations.

CARROT PROTOCOL

[L-02] Missing strategy duplication check

During the initialization of various strategies, the current implementation does not check for duplicate strategies.

This oversight could result in the double-counting of certain balances when calculating TVL.

It is recommended to ensure that no strategies with identical types and accounts used for balance calculation are added.

Resolution

The team acknowledged this finding and will take steps to avoid such corner cases in future use.

CARROT PROTOCOL

[L-03] Missing necessary checks before removing strategy

The vault owner can remove unnecessary strategies using the "`remove_strategy`" instruction.

```
/* programs/carrot/src/instructions/remove_strategy.rs */
pub fn remove_strategy_handler(ctx: Context<RemoveStrategy>) -> Result<()> {
    // remove strategy from vault record
    let vault_account = &ctx.accounts.vault.to_account_info();
    ctx.accounts.vault.rm_strategy(
        ctx.accounts.strategy.metadata.strategy_id,
        vault_account,
        ctx.accounts.authority.clone(),
        ctx.accounts.system_program.clone(),
    )?;

    // close strategy pda and return lamports to authority
    ctx.accounts
        .strategy
        .close(ctx.accounts.authority.to_account_info())
}
```

However, the current implementation does not perform any checks before deleting a strategy, which can result in the accidental removal of strategies that are still in use.

It is recommended to add necessary checks, such as ensuring the balance of the strategy to be removed is zero.

Resolution

This issue has been resolved by commit 9a02965c9e0af35428bbeaff6b87cbc7dafd71a3.

CARROT PROTOCOL

[L-04] Missing accounts consistency check between init and deposit/withdraw

During the initialization of various strategies, the required accounts for deposit and withdrawal operations are recorded in the "`strategy_type`".

However, during subsequent deposit and withdrawal operations, there is no verification to ensure that the accounts used match those declared during initialization.

Although these operations can only be performed by the keeper, it is recommended to implement the necessary checks.

Resolution

This issue has been resolved by commit 9a02965c9e0af35428bbeaff6b87cbc7dafd71a3.

CARROT PROTOCOL

[L-05] Inaccurate management fee calculation

The protocol calculates the management fee based on the TVL. However, within the redeem instruction, it incorrectly uses the TVL after redemption for this calculation, resulting in an underestimation of the management fee.

```
/* programs/carrot/src/instructions/redeem.rs */
burn(
    CpiContext::new(
        ctx.accounts.shares_token_program.to_account_info(),
        burn_shares_accounts,
    ),
    args.amount,
)?;

// reload shares mint after CPI
ctx.accounts.shares.reload().unwrap();

// recalculate tvl and nav after redeem
vault_tvl = ctx.accounts.vault.get_tvl(ctx.remaining_accounts)?;

// calculate fee
let fee_amount = ctx.accounts.vault.fee.calculate_management_fee(
    vault_tvl,
    ctx.accounts.shares.supply,
    ctx.accounts.shares.decimals,
);

let redeem_event = RedeemEvent {
    withdrawer: ctx.accounts.user.key(),
    mint: ctx.accounts.asset.key(),
    amount: asset_amount,
    shares_burned: args.amount,
    management_fee: fee_amount,
    tvl: vault_tvl,
};
```

Resolution

This issue has been resolved by commit 9a02965c9e0af35428bbeaff6b87cbc7dafd71a3.

CARROT PROTOCOL

[L-06] Arbitrary CPIs in solend init and withdraw operations

In both "`solend_supply_strategy_init`" and "`solend_supply_strategy_withdraw`" instructions, there are no constraints applied to the passed "`solend_program`" account. As a result, any program address can be passed in for CPIs.

```
/* programs/carrot/src/instructions/solend_supply_strategy_init.rs */
/// CHECK: todo
pub solend_program: UncheckedAccount<'info>,

/* programs/carrot/src/instructions/solend_supply_strategy_withdraw.rs */
/// CHECK: todo
pub solend_program: UncheckedAccount<'info>,
```

Although these two instructions require the "`vault.authority`" signature, meaning they can only be called by the Keeper, it is still recommended to use constraints to ensure the correct CPIs:

```
/* programs/carrot/src/instructions/solend_supply_strategy_deposit.rs */
/// CHECK: todo
#[account(executable, address = solend_sdk::solend_mainnet::ID)]
pub solend_program: UncheckedAccount<'info>,
```

Resolution

This issue has been resolved by commit 9a02965c9e0af35428bbeaff6b87cbc7dafd71a3.

CARROT PROTOCOL

[I-01] Consider incorporating Pyth confidence interval

When users perform issue and redeem operations, the price of the respective asset is obtained via the Pyth oracle, which is also used for calculating TVL.

```
/* programs/carrot/src/state/data.rs */
// find pyth oracle for asset
let price = get_price_usd_from_pyth_oracle(&asset.oracle, remaining_accounts)?;

let balance_usd =
    calc_usd_amount(self.balance, asset.decimals, price.price, price.expo, true)
        .ok_or(CarrotError::StrategyBalanceCalculationError)?;
```

However, the calculation currently uses only the EMA price returned by Pyth, without considering the confidence interval.

It is recommended to incorporate the confidence interval in a manner favorable to the protocol to further safeguard the protocol.

Resolution

This issue has been resolved by commits 8e81b1b65a17f86d658107e898a2626037d74bd5, c70359330b0e528a1e85c0d1bff156fde27d5f40 and 0cb1f66864bf44852cda4e4863123b5c5cfd017c.

CARROT PROTOCOL

[I-02] Missing frozen authority check in init_vault

When initializing a vault, the current implementation opts to use an existing mint rather than creating a new one. This process only verifies that the supply of the mint is zero.

```
/* programs/carrot/src/instructions/init_vault.rs */
#[account(
    mint::decimals = Vault::SHARES_DECIMALS,
    mint::authority = vault,
    constraint = shares.supply == 0,
)]
pub shares: InterfaceAccount<'info, Mint>,
```

It is recommended to also check that the mint's frozen authority is set to "None" to provide additional protection for users.

Resolution

This issue has been resolved by commit 9a02965c9e0af35428bbeaff6b87cbc7dafd71a3.

CARROT PROTOCOL

[I-03] Check asset decimal when adding assets

In the process of calculating token value, the current implementation assumes that the decimal places of asset tokens are always less than or equal to 9.

```
/* programs/carrot/src/utils/mod.rs */
// Scale the token amount to the base unit (USD cents)
let scaled_token_amount =
    token_amount.checked_mul(10_u128.pow((PRECISION - token_decimal) as u32))?;
```

However, there is no corresponding check when adding assets. It is recommended to implement this check during the asset addition process.

Resolution

This issue has been resolved by commit 9a02965c9e0af35428bbeaff6b87cbc7dafd71a3.

CARROT PROTOCOL

[I-04] Incorrect space calculation for Fee struct

The current implementation incorrectly calculates the space for the "Fee" struct as requiring 4 bytes instead of 2 bytes for a u16 type.

```
/* program/programs/carrot/src/state/data.rs */

pub struct Fee {
    pub management_fee_bps: u16,
    pub management_fee_last_update: i64,
    pub management_fee_accumulated: u64,
    pub performance_fee_bps: u16,
}

impl Fee {
    pub const SPACE: usize = 4 + 8 + 8 + 4;
```

This results in an overestimation of the required space, leading to a slight waste of rent.

Resolution

This issue has been resolved by commit 9a02965c9e0af35428bbeaff6b87cbc7dafd71a3.

CARROT PROTOCOL

[I-05] Inconsistent with klend implementation in klend_exchange_rate

In the calculation of exchange rates and balances for KLend, the Carrot Protocol implementation differs from KLend's implementation in two ways:

1. For the corner case where "`mint_total_supply`" or "`total_liquidity`" is zero, Carrot Protocol incorrectly uses 0 as the exchange rate instead of 1.
2. Carrot Protocol uses "`f64`" for subsequent calculations rather than the "`Fraction`" type employed by KLend, which may result in some inaccuracies.

Carrot Protocol:

```
/* programs/carrot/src/instructions/klend_supply_strategy_deposit.rs */
// calculate collateral:liquidity exchange rate
pub fn klend_exchange_rate(mint_total_supply: u64, total_liquidity: f64) -> f64 {
    let rate = if mint_total_supply == 0 || total_liquidity == 0.0 {
        0.0
    } else {
        mint_total_supply as f64 / total_liquidity
    };

    rate
}
```

KLend:

```
/* programs/klend/src/utils/consts.rs */
pub const INITIAL_COLLATERAL_RATE: Fraction = fraction!(1);

/* programs/klend/src/state/reserve.rs */
fn exchange_rate(&self, total_liquidity: Fraction) -> LendingResult<CollateralExchangeRate> {
    let rate = if self.mint_total_supply == 0 || total_liquidity == Fraction::ZERO {
        INITIAL_COLLATERAL_RATE
    } else {
        Fraction::from(self.mint_total_supply) / total_liquidity
    };

    Ok(CollateralExchangeRate(rate))
}
```

Resolution

This issue has been resolved by commit 9a02965c9e0af35428bbeaff6b87cbc7dafd71a3.

Appendix: Methodology and Scope of Work

The Sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in smart contract security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis and penetration testing.

Assisted by the Sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and Carrot Labs, Inc. dba Carrot Labs (the "Client''). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

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

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, Sec3 (formerly Soteria) is a leading blockchain security company. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At Sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.

Sec3™