Security Assessment Report

Hedge Vault v0.1.0

May 31st, 2022

Summary

The Sec3 (formerly Soteria) team was engaged to do a thorough security analysis of the Hedge Vault v0.1.0 Solana smart contract program. The artifact of the audit was the source code of the following on-chain smart contract excluding tests in a private repository:

- Commit 165a3ab419be68b1c25cd32a93c15492facc16e1

The audit revealed 12 issues, which were reported to the Hedge team.

The Hedge team responded with several commits and PRs for the post-audit review.

- Commit: d9a1c2914fc13dd5131238af7742a41944d7b8a7
- PR: #86, #96, #98, #99, #101, #102, #103, #104

The scope of the post-audit review is to validate if the reported issues have been addressed.

This report describes the findings and resolutions in detail.

Table of Contents

Methodology and Scope of Work

Sec3's audit team, which consists of Computer Science professors and industrial researchers with extensive experience in Solana 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.
    - Missing ownership checks
    - Missing signer checks
    - Signed invocation of unverified programs
    - Solana account confusions
    - Arithmetic over- or underflows
    - Numerical precision errors
    - Loss of precision in calculation
    - Insufficient SPL-Token account verification
    - Missing rent exemption assertion
    - Casting truncation
    - Did not follow security best practices
    - Outdated dependencies
    - Redundant code
    - Unsafe Rust code
- 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

Result Overview

In total, the audit team found the following issues.

CONTRACT HEDGE VAULT v0.1.0

Issue	Impact	Status
[M-1] Missing checks in vault linked list	Medium	Resolved
[M-2] Price cache period may be too long	Medium	Resolved
[M-3] Missing price oracle robustness checks	Medium	Resolved
[I-1] Accounting inconsistency	Informational	Resolved
[I-2] Inconsistent associated token account checks	Informational	Resolved
[I-3] Inconsistent LiquidationPoolEra checks	Informational	Resolved
[I-4] Inconsistent vault_type_account.deprecated checks	Informational	Resolved
[I-5] Inconsistent space calculation	Informational	Resolved
[I-6] Design, best practice, etc.	Informational	Resolved
[I-7] Outdated information in whitepaper	Informational	Resolved
[I-8] Partial repay	Informational	Resolved
[I-9] Inconsistent denormalized debt computation	Informational	Resolved

Findings in Detail

IMPACT – MEDIUM

[M-1] Missing checks in vault linked list

```
/* hedge-vault/programs/hedge-vault/src/processors/common.rs */
124 | pub fn update_vault_position_in_redeem_list<'a>(
125 |     vault_type_account: &mut Account<'a, VaultType>,
126 |     vault: &mut Account<'a, Vault>,
127 |     new_smaller_vault: &mut Account<'a, Vault>,
128 |     new_larger_vault: &mut Account<'a, Vault>,
129 |     old_smaller_vault: &mut Account<'a, Vault>,
130 |     previous_status: VaultStatus,
131 |     new_status: VaultStatus,
132 | ) {
```

- `old_smaller_vault` should be constrained. Its status should be `VaultStatus::Open` or it should be the `vault`.
- When removing `vault` from the linked list, reset `vault.next_vault_to_redeem` to `None`.
- `new_smaller_vault` and `new_larger_vault` should be further constrained. They should be `vault` or nodes on the linked list (the status should be open)

The following is an example showing how a linked list can be corrupted.

```
[0] Consider the following open vault linked list

vault_type_account.first_vault_to_redeem (fv) -> v0 -> v1 -> v2 -> v3 -> ... -> v4 -> v5 -> ...

**********************************************************************

[1] old_smaller_vault = v0, vault = v1, new_status != Open, previous_status = Open

    v1 is removed from the list but its "next_vault_to_redeem" still points to v2

       fv -> v0 -------> v2 -> v3 -> ... -> v4 -> v5 -> ...
                         ^
                         |
                         v1 (v1's new status is not Open anymore)
```

```
**********************************************************************

[2] old_smaller_vault = v1, vault = v2, previous_status  = Open

    It's supposed to remove the link v0 -> v2. However, since "old_smaller_vault" comes from clients
and there is no checks if it is still in the list, "v1" can be "old_smaller_vault". As a result,
"v1.next_vault_to_redeem" becomes "v3".

       fv -> v0 -------> v2 -> v3 -> ... -> v4 -> v5 -> ...
                                ^
                                |
                                v1
    new_smaller_vault = v4, new_larger_vault = v5, new_status = Open
       fv -> v0 -------> v2 -> v5 -> ...
                         ^
                         |
      v1 -> v3 -> ... -> v4
```

As a result, vaults `v3` to `v4` are not reachable

from `vault_type_account.first_vault_to_redeem` and not redeemable.

Resolution

The issue has been resolved.

IMPACT – MEDIUM

[M-2] Price cache period may be too long

```
/* hedge-vault/programs/hedge-vault/src/account_data/vault_type.rs */
045 | pub fn get_current_price(&self) -> Result<Decimal> {
046 |     let current_time = Clock::get().unwrap().unix_timestamp.to_u64().unwrap();
059 |     let time_since_refresh = current_time - self.price_last_updated_timestamp;
060 |
061 |     let max_age_in_seconds = 60;
062 |     if time_since_refresh > max_age_in_seconds {
063 |         msg!(
064 |             "HEDGE: Collateral ({:?}) price is more than {:?} seconds old and needs to be refreshed",
065 |             self.collateral_type,
066 |             max_age_in_seconds
067 |         );
068 |         return err!(HedgeErrors::CollateralPriceOutOfDate);
069 |     }
070 |     msg!("HEDGE: get_current_price: {:?}", self.recent_price);
071 |     Ok(Decimal::from_account(self.recent_price))
072 | }
```

Currently, the caching expiration window is hard-coded at 60 seconds, which is a long time for drastic price movements. For example, `redeem_get_sol` depends on `collateral_usd_price`, which could be off considerably in tail risk events.

It's helpful to query the price more frequently or use a time-weighted average price (TWAP).

Resolution

The issue has been resolved.

IMPACT – MEDIUM

[M-3] Missing price oracle robustness checks

Pyth

1. The current implementation does not check if the price status is `trading`. According to the Pyth documents "Only prices with a value of status=trading should be used. If the status is not trading but is Unknown, Halted or Auction the Pyth price can be an arbitrary value" https://docs.pyth.network/consume-data/best-practices#current-price-availability.

2. Price confidence: https://docs.pyth.network/how-pyth-works/ema-price-aggregation.

Switchboard

1. The current implementation has the Pyth slot/stale check but does not do so for Switchboard.

Resolution

These issues have been resolved. The Hedge team will further check with the Switchboard team regarding recommendations and best practices.

IMPACT – INFO

[I-1] Accounting inconsistency

In line 426, the accounting is inconsistent. It should be collateral_to_be_liquidated instead of collateral_to_send_to_liquidation_pool.

```
/* hedge-vault/programs/hedge-vault/src/processors/liquidate_vault.rs */
354 | // if liquidated_value_of_vault_in_usd is positive
355 | if market_value_in_ush > 0 {
360 |     let market_value_of_collateral =
361 |         Decimal::from_u64(market_value_in_ush).unwrap() / collateral_usd_price;
362 |     // The platform gets 20%
363 |     collateral_fee_paid_to_hedge_staking_pool = (market_value_of_collateral
364 |         * Decimal::new(2, 1))
365 |     .to_u64()
366 |     .unwrap();
367 |     // The person calling the function get 5% as a reward
368 |     collateral_fee_paid_to_person_liquidating_vault = (market_value_of_collateral
369 |         * Decimal::new(5, 2))
370 |     .to_u64()
371 |     .unwrap();
372 |     // The liquidation pool gets the rest
373 |     collateral_to_send_to_liquidation_pool = collateral_to_be_liquidated.to_u64().unwrap()
374 |         - (collateral_fee_paid_to_hedge_staking_pool
375 |             + collateral_fee_paid_to_person_liquidating_vault);
376 | } else {
377 |     msg!("Hedge: Vault underwater");
379 |     collateral_fee_paid_to_hedge_staking_pool = 0;
380 |     collateral_fee_paid_to_person_liquidating_vault =
381 |         (Decimal::from_u64(collateral_to_be_liquidated).unwrap() * Decimal::new(1, 2))
382 |             .to_u64()
383 |             .unwrap();
384 |     collateral_to_send_to_liquidation_pool = collateral_to_be_liquidated.to_u64().unwrap()
385 |         - collateral_fee_paid_to_hedge_staking_pool
386 |         - collateral_fee_paid_to_person_liquidating_vault;
387 | }
425 | // Update the total collateral held for this collateral type
426 | vault_type_account.collateral_held -= collateral_to_send_to_liquidation_pool;
```

Resolution

The issue has been resolved. Extra accountings at each step were added too.

IMPACT – INFO

[I-2] Inconsistent associated token account checks

Associated token accounts are validated in 3 different ways. Some are done via `get_associated_token_address` or `find_program_address`. Some are done by checking the owner and mint, which are less strict. Here is an example.

```
/* hedge-vault/programs/hedge-vault/src/context/deposit_vault.rs */
033 | #[derive(Accounts)]
034 | pub struct DepositVault<'info> {
048 |     #[account(mut,
049 |         address = get_associated_token_address(&vault_type_account.key(),
&vault_type_account.collateral_mint)
050 |     )]
051 |     pub vault_type_associated_token_account: Box<Account<'info, TokenAccount>>,
052 |
053 |     #[account(mut,
054 |         address =
055 |             Pubkey::find_program_address(&[
056 |                 &vault.key().to_bytes(),
057 |                 &token::ID.to_bytes(),
058 |                 &vault_type_account.collateral_mint.to_bytes(),
059 |             ], &associated_token::ID).0
060 |     )]
061 |     pub vault_associated_token_account: Box<Account<'info, TokenAccount>>,
067 |
068 |     #[account(mut,
069 |         constraint = fee_pool_associated_ush_token_account.owner == fee_pool.key(),
070 |         constraint = fee_pool_associated_ush_token_account.mint == ush_mint.key()
071 |     )]
072 |     pub fee_pool_associated_ush_token_account: Box<Account<'info, TokenAccount>>,
073 |
```

Resolution

The issue has been resolved.

IMPACT – INFO

[I-3] Inconsistent LiquidationPoolEra checks

`deposit_liquidation` requires:

- `pool_state.current_era = pool_era.key = pool_position.era`

```
/* hedge-vault/programs/hedge-vault/src/context/deposit_liquidation.rs */
018 | #[derive(Accounts)]
019 | pub struct DepositLiquidation<'info> {
026 |     #[account(mut,
027 |         seeds = [b"LiquidationPoolStateV1".as_ref()],
028 |         bump
029 |     )]
030 |     pub pool_state: Account<'info, LiquidationPoolState>,
038 |     #[account(mut,
039 |         constraint = pool_state.current_era == pool_era.key()
040 |     )]
041 |     pub pool_era: AccountLoader<'info, LiquidationPoolEra>,
043 |     #[account(init, payer = payer, space = 8 + 1176)] //8 + (32*2) + (8*5) + (16*2) + (16*64))]
044 |     pub pool_position: AccountLoader<'info, LiquidationPosition>,
```

However, `close_liquidation_pool_position` requires

- `pool_era = pool_position.load()?.era`
- no requirement on `pool_state.current_era`

This is inconsistent as closing a position associated with a previous era is allowed.

Resolution

It's confirmed that this is the intended behavior.

IMPACT – INFO

[I-4] Inconsistent vault_type_account.deprecated checks

```
/* hedge-vault/programs/hedge-vault/src/context/loan_vault.rs */
033 | #[derive(Accounts)]
034 | pub struct LoanVault<'info> {
041 |     #[account(mut,
044 |         constraint = !vault_type_account.deprecated @HedgeErrors::VaultTypeDeprecated,
046 |     )]
047 |     pub vault_type_account: Box<Account<'info, VaultType>>,
```

deprecated is checked only in

- loan_vault
- create_vault

but not checked in other instructions such as

- claim_liquidation_pool_position
- claim_staking_pool_position
- deposit_vault
- liquidate_vault
- redeem_vault
- refresh_oracel_price
- repay_valut
- withdraw_vault

Resolution

It's confirmed that they are the intended behaviors. However, deposits should not be allowed when it is deprecated. This issue has been fixed.

IMPACT – INFO

[I-5] Inconsistent space calculation

The space calculations of multiple account structs are outdated/incorrect.

However, since the contract allocates extra space for future migration, it's ok for now but may be a problem later.

```
/* hedge-vault/programs/hedge-vault/src/account_data/vault_system_state.rs */
011 | #[account]
012 | pub struct VaultSystemState {
013 |     pub last_redeem_fee_bytes: u128,    // 8 bytes --> 16 bytes
014 |     pub last_redeem_timestamp: u64,     // 8 bytes
015 |     pub total_ush_supply: u64,         // 8 bytes
016 |     pub total_vaults: u64,              // 8 bytes
017 |     pub total_vaults_closed: u64,       // 8 bytes
018 |     pub debt_redistribution_error: u64, // 8 bytes
019 |     pub authority: Pubkey,              // 32 bytes
020 |     pub hedge_staking_pool: Pubkey,     // 32 bytes
021 |     pub collateral_type_count: u64,     // 8 bytes
022 |     pub status: SystemStatus,           // 8 bytes
023 | }

/* hedge-vault/programs/hedge-vault/src/context/init_hedge_foundation.rs */
017 | #[derive(Accounts)]
018 | pub struct InitHedgeFoundation<'info> {
019 |     #[account(
020 |         init,
021 |         seeds = [b"VaultSystemStateV1"],
022 |         bump,
023 |         payer = founder,
024 |         space = 8 + (8 * 10) + 4 + (32 * 2) + 1024 // Buffer up the size for future migrations
025 |     )]
026 |     pub vault_system_state: Account<'info, VaultSystemState>,

/* hedge-vault/programs/hedge-vault/src/account_data/oracle_info_for_collateral_type.rs */
006 | #[account]
007 | pub struct OracleInfoForCollateralType {
008 |     pub collateral_type: String,  // 8 bytes - assuming UTF-8: 1 byte = 1 char --> 24
009 |     pub oracle_pyth: Pubkey,      // 32 bytes
010 |     pub oracle_chainlink: Pubkey, // 32 bytes
011 |     pub oracle_switchboard: Pubkey, // 32 bytes
012 | }
013 |
```

```
/* hedge-vault/programs/hedge-vault/src/context/add_collateral_type.rs */
016 | #[derive(Accounts)]
018 | pub struct AddCollateralType<'info> {
030 |     #[account(init,
031 |         payer = payer,
032 |         seeds = [collateral_type.as_ref(), b"Oracle".as_ref()],
033 |         bump,
034 |         space = 8 + (32*3) + 1024 // Pad up for future use
035 |     )]
036 |     pub oracle_info_account: Account<'info, OracleInfoForCollateralType>,

/* hedge-vault/programs/hedge-vault/src/account_data/liquidation_pool_state.rs */
055 | #[test]
056 | fn test_struct_size() {
057 |     use crate::account_data::vault_system_state::VaultSystemState;
058 |     use crate::account_data::oracle_info_for_collateral_type::OracleInfoForCollateralType;
059 |     eprintln!("VaultSystemState Size {:?}", core::mem::size_of::<VaultSystemState>());
060 |     eprintln!("OracleInfoForCollateralType Size {:?}",
core::mem::size_of::<OracleInfoForCollateralType>());
061 | }
```

```
VaultSystemState Size 136
OracleInfoForCollateralType Size 120
VaultType Size 272
```

Resolution

The issue has been resolved.

IMPACT – INFO

[I-6] Design, best practice, etc.

1. The constrains on the destination account of transfers require they are owned by signers. It's safer but may be too strict for users, as they need to pay fees if they want to transfer from their associated accounts to other accounts.
2. `painc` and `unwrap` can be better handled
3. Consider use `UncheckedAccount` instead of `AccountInfo`.

Resolution

These issues have been resolved.

IMPACT – INFO

[I-7] Outdated information in whitepaper

1. Sec 2.6.2 vault distribution is not what the implementation does. Instead, A.3.3 describes how the redistribution works.
2. in liquidation, it may be helpful to mention If vault underwater, the person calling only gets 1% of the total collateral and the pool gets the rest.

Resolution

These issues have been resolved.

- The redistributions have been disabled for now to incentivize depositing into stability pool.
- https://docs.hedge.so/protocol-overview/liquidation-and-liquidation-pool noted in docs that 1% of total is given to liquidator.

IMPACT – INFO

[I-8] Partial repay

In line 101, it seems to be `repay_amount >= actual_vault_debt` as "`repay_amount == actual_vault_debt`" is not a partial repay.

```
/* hedge-vault/programs/hedge-vault/src/processors/repay_vault.rs */
100 | // if the repayment amount is larger than the vault debt, just payy off the entire thing
101 | if repay_amount > actual_vault_debt {
102 |     msg!("HEDGE: Paying off entire vault debt: {:?}", actual_vault_debt);
103 |     normalized_repay_amount = actual_vault_debt;
104 |     denormalized_repay_amount = vault_account.denormalized_debt;
105 | } else {
106 |     msg!("HEDGE: Paying off a portion of vault debt. Paying {:?} of {:?} total.", repay_amount,
actual_vault_debt);
107 |     normalized_repay_amount = repay_amount;
108 |     denormalized_repay_amount = vault_type_account
109 |         .denormalize(repay_amount)?
110 |         .to_u64()
111 |         .unwrap();
112 | }
```

Resolution

The issue has been resolved.

IMPACT – INFO

[I-9] Inconsistent denormalized debt computation

In redeem_vault, it seems inconsistent vault_account.denormalized_debt and vault_type_account.denormalized_debt_extended are deducted by two different numbers. In other instructions, they are increased/deducted by a same amount.

```
/* hedge-vault/programs/hedge-vault/src/processors/redeem_vault.rs */
208 | // Calculate the USH that's going to be used to pay down the vault
209 | let ush_to_pay_down_debt = normalized_redeem_amount - ush_redeem_fee;
259 | vault_account.denormalized_debt -= normalized_redeem_amount.to_u64().unwrap();
263 | // Calculate the denormalized value of the debt.
264 | let denormalized_debt_paid_off =
265 |     vault_type_account.denormalize(ush_to_pay_down_debt.to_u64().unwrap())?;
267 | // Update the total debt held for this collateral type
268 | vault_type_account.denormalized_debt_extended -= denormalized_debt_paid_off.to_u64().unwrap();
```

Resolution

The issue has been resolved.

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and Hedge 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 that currently focuses on Solana programs. 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™