Security Assessment Report

Pye Fi Bonds

March 19, 2025

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Pye Fi Bonds 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 23 issues or questions.

program	type	commit
bonds	Solana	`ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`

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

Table of Contents

Result Overview

Issue	Impact	Status
BONDS		
[H-01] Validate the maturity timestamp in `redeem_pt` and `redeem_yt`	High	Resolved
[H-02] Accounting issues in the counterparty and redemption cache design	High	Resolved
[H-03] Incorrect accounting in `counter_party_redeem_yt`	High	Resolved
[H-04] Hijack the bond stake account, re-stake and steal funds	High	Resolved
[M-01] Fee evasions	Medium	Resolved
[M-02] DoS in `counter_party_update_liquid_reserve`	Medium	Resolved
[M-03] Post-maturity `SoloValidatorDelegateTips` blocks `UpdateLiquidReserve`	Medium	Resolved
[L-01] Unchecked stake pool programs in `StakePoolBond` initialization	Low	Resolved
[L-02] Unhandled Token 2022 mints with the `TransferFee` extension	Low	Resolved
[L-03] Rounding up fees to prevent fee evasion	Low	Resolved
[L-04] Missing `issuance_ts < maturity_ts` checks	Low	Resolved
[L-05] `InsufficientFunds` error due to partial unstake amount	Low	Resolved
[L-06] The `counter_party_pt` and `counter_party_yt` should be ATAs	Low	Resolved
[L-07] Missing `can_redeem` check in `counter_party` instructions	Low	Resolved
[L-08] Inaccurate PT token mint amount in the `SoloValidator`	Low	Resolved
[L-09] Incorrect `SoloValidatorBond` account size	Low	Resolved
[L-10] Outdated `stake_pool.total_lamports"`	Low	Resolved
[L-11] Deactivated but not withdrawn stake	Low	Resolved
[I-01] Potential arithmetic overflow in `calc_pt_to_mint`	Info	Resolved
[I-02] Validate the `validator_vote_account`	Info	Resolved
[I-03] Validate `deposit_fee_bps` and `counter_party_fee_bps`	Info	Resolved
[I-04] Move the definition of `ephemeral_stake_account` to the activation scope	Info	Resolved
[I-05] Overdrawing the bond account	Info	Resolved

Findings in Detail

BONDS

[H-01] Validate the maturity timestamp in `redeem_pt` and `redeem_yt`

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

In `StakePoolBond`, redemptions should occur only after the bond has matured.

However, the `redeem_pt` and `redeem_yt` do not validate maturity time (even though there is already relevant check logic in the validate function, this function is not called), allowing users to redeem at any time.

```
/* programs/bonds/src/lib.rs */
164 | pub fn redeem_pt(ctx: Context<RedeemPt>, args: RedeemPtArgs) -> Result<()> {
165 |     instructions::redeem_pt::handler(ctx, args)
166 | }
168 | pub fn redeem_yt(ctx: Context<RedeemYt>, args: RedeemYtArgs) -> Result<()> {
169 |     instructions::redeem_yt::handler(ctx, args)
170 | }
```

During a normal redemption process, to correctly calculate principal and yield, the program invokes `set_redemption_cache` during the first redemption to cache some states:

- `pt_supply_at_first_redemption`: PT mint supply
- `yt_supply_at_first_redemption`: YT mint supply
- `bond_lst_bal_at_first_redemption`: Bond LST total balance
- `lst_for_all_pt_at_redemption`: LST for all PT

These values are cached during the first redemption and remain unchanged afterward. For subsequent redemptions, these cached values are used to calculate the LST amount:

1. `redeem_pt`: (input `pt_amount`)

```
lst_amount = pt_amount * lst_for_all_pt_at_redemption / pt_supply_at_first_redemption
```

2. `redeem_yt`: (input `yt_amount`)

```
lst_amount = yt_amount * (bond_lst_bal_at_first_redemption - lst_for_all_pt_at_redemption) /
↪  yt_supply_at_first_redemption
```

There are several ways to exploit this oversight:

1. Inaccurate deposit accounting

An attacker could call the redeem instruction, causing the LST amounts from all subsequent deposit transfers to be incorrectly recorded in the Vault's LST balance.

2. Withdraw more LST

Also, an attacker could deposit LST to acquire PT and YT at some point, while the recorded supplies remain unchanged, potentially enabling arbitrage opportunities. Example scenario with PT increase:

First, using a negligible amount of PT to call the redeem instruction, fixing the state as follows

```
stake pool SOL = 1_000_000_000
stake pool LST = 10_000_000

PT mint supply = 80_000_000
YT mint supply = 10_000_000
Vault LST balance = 1_000_000
Vault SOL balance
  = Vault LST balance * stake pool SOL / stake pool LST
  = 1_000_000 * 1_000_000_000 / 10_000_000
  = 100_000_000
LST for all PT
  = Vault LST balance * PT mint supply / vault SOL balance
  = 1_000_000 * 80_000_000 / 100_000_000
  = 800_000
-> 100 PT = 1 LST
```

Then in the same slot, executing a deposit and a redeem together

- `deposit` instruction:
  - deposit 100 LST

  - mint `x` PT to the user
- `redeem_pt` instruction:
  - transfer `x` PT
  - withdraw `lst_amount`, which is `x * 800_000 / 80_000_000`, or (`x` / 100) LST

If the computed `x` (based on `LstState::lamports_for_lst`) results in 1 LST > 100 PT, the attacker gains more LST upon redemption than was deposited.

This occurs because the PT/LST rate during deposit exceeds the PT/LST rate during redemption.

3. DoS TY redeem

Another exploit method starts by minting 1 PT and 1 YT, then burning 1 YT to set the redeem cache. This causes `self.pt_supply_at_first_redemption` to be set to 0.

During the YT redemption process, a division by zero is triggered in `.div(U192::from(self.pt_supply_at_first_redemption))`, resulting in a panic and effectively causing a DoS on the bond's YT redemption functionality.

It's recommended to add the check.

```
#[access_control(RedeemPt::validate(&ctx))]
#[access_control(RedeemYt::validate(&ctx))]
```

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[H-02] Accounting issues in the counterparty and redemption cache design

Identified in commit `ce43f680998713428007b8eb8d5c65e7ecd44221`.

For a solo validator bond, after its `maturity_handled` is set to `true`, YT/PT token holders can redeem their tokens using the following instructions:

- PT tokens
  - `solo_validator_redeem_pt_for_stake`
  - `solo_validator_redeem_pt_for_sol`
- YT tokens
  - `solo_validator_redeem_yt_for_stake`
  - `solo_validator_redeem_yt_for_sol`

Among them, if token holders choose to redeem PT/YT tokens for sol, the process follows a design involving the global singleton counterparty owned by the contract. During the redemption, instead of directly burning the PT/YT tokens, tokens are transferred from the redeemer to the global counterparty, while the lamports represented by the tokens are removed from `lamports_for_pts` or `lamports_for_yts` in the redemption cache.

Later, when counterparty side instruction `counter_party_redeem_pt` or `counter_party_redeem_yt` is called, the PT/YT tokens transferred to the counterparty are burnt. Only until this moment, the value of the transferred away PT/YT tokens in the previous redemption is deducted from the bond's total value.

To simplify the discussion, let's assume that neither slashing nor staking yields occur when redemption instructions are executed. Although the discussion is around PT tokens, YT tokens share the same issue.

The impacts of the redeemer side and counterparty side PT redemption instructions are summarized in the table below.

	`solo_validator_redeem_pt_for_sol`	`counter_party_redeem_pt`
PT backed by cache	**decreased** by `pt_amount`	no change
`pt_supply`	no change	**decreased** by `pt_amount`
`lamports_for_pts`	**decreased** by `lamports_for_pts * pt_amount / pt_supply`	**decreased** by `lamports_for_pts * pt_amount / pt_supply`
`bond_total_lamports`	no change	**decreased** by `lamports_for_pts * pt_amount/ pt_supply`

Because these instructions have side effects, depending on how they are invoked, they could introduce the following accounting issues:

1. Redemption cache lamports double counting

1.1 `solo_validator_redeem_pt_for_sol`

When redeeming `pt_amount` PT tokens, their value is deducted from the `lamports_for_pts` in function `calculate_lamports_for_pt()` in line 103.

```
/* src/instructions/solo_validator/redeem_pt_for_sol.rs */
091 | pub fn handler(
092 |     ctx: Context<SoloValidatorRedeemPtForSol>,
093 |     args: SoloValidatorRedeemPtForSolArgs,
094 | ) -> Result<()> {
095 |     let pt_amount = args.amount;
103 |     let mut lamports_for_user = SoloValidatorBond::calculate_lamports_for_pt(
104 |         pt_amount,
105 |         ctx.accounts.principal_token_mint.supply,
106 |         &mut ctx.accounts.bond,
109 |     )?;

/* src/state/solo_validator_bond.rs */
233 | pub fn calculate_lamports_for_pt(
234 |     pt_burned: u64,
235 |     pt_supply: u64,
236 |     bond_account: &mut Account<'_, Self>
239 | ) -> Result<u64> {
246 |     bond_account.redemption_cache.redeem_pts_for_lamports(
247 |         pt_burned,
248 |         pt_supply,
250 |     )
251 | }
```

```
/* src/state/solo_validator_bond.rs */
069 | pub fn redeem_pts_for_lamports(
070 |     &mut self,
071 |     pt_amount: u64,
072 |     pt_current_supply: u64,
074 | ) -> Result<u64> {
075 |     if self.was_slashed {
078 |     } else {
080 |         // Handle the pro-rated PT amount
081 |         let users_lamports_for_pt_redemption =
082 |             mul_div(self.lamports_for_pts, pt_amount, pt_current_supply)?;
084 |         // Debit the lamports for PT
085 |         self.lamports_for_pts = self
086 |             .lamports_for_pts
087 |             .checked_sub(users_lamports_for_pt_redemption)
088 |             .ok_or(ErrorCode::ArithmeticOverflow)?;
089 |         Ok(users_lamports_for_pt_redemption)
090 |     }
091 | }
```

Later, when the same amount of PT tokens are burnt in the `counter_party_redeem_pt` instruction, even though they do not belong to the bond, their value is deducted again from the `lamports_for_pts` in function `calculate_lamports_for_pt()` in line 102.

```
/* src/instructions/counter_party/redeem_pt.rs */
098 | pub fn handler(ctx: Context<CounterPartyRedeemPt>, args: CounterPartyRedeemPtArgs) -> Result<()> {
099 |     let pt_amount = args.amount;
102 |     let lamports_for_counter_party = SoloValidatorBond::calculate_lamports_for_pt(
103 |         pt_amount,
104 |         ctx.accounts.principal_token_mint.supply,
105 |         &mut ctx.accounts.bond,
106 |         &ctx.accounts.stake_account,
107 |         bond_rent,
108 |     )?;
```

Additionally, the `counter_party_update_liquid_reserve` instruction uses the counterparty's PT/YT amount to redeem lamports from the bond stake account, after which the counterparty's redeem instruction is used to withdraw the bond's lamports. The `calculate_lamports_for_pt` function is used here as well, which deducts the `lamports_for_pts`.

```
/* programs/bonds/src/instructions/counter_party/update_liquid_reserve.rs */
083 | pub fn handler<'info>(ctx: Context<'_, '_, '_, 'info, UpdateLiquidReserve<'info>>) -> Result<()> {
135 |     let lamports_for_pts = SoloValidatorBond::calculate_lamports_for_pt(
136 |         ctx.accounts.counter_party_pt.amount,
137 |         ctx.accounts.principal_token_mint.supply,
138 |         &mut ctx.accounts.bond,
139 |         &ctx.accounts.stake_account,
```

```
140 |         bond_rent,
141 |     )?;
142 |     let lmaports_for_yts = SoloValidatorBond::calculate_lamports_for_yt(
143 |         ctx.accounts.counter_party_yt.amount,
144 |         ctx.accounts.yield_token_mint.supply,
145 |         &mut ctx.accounts.bond,
146 |         &ctx.accounts.stake_account,
147 |         bond_rent,
148 |     )?;
```

1.2 `counter_party_update_liquid_reserve`

Similarly, the lamports in the redemption cache are deducted twice for the same PT/YT tokens.

```
/* src/instructions/counter_party/update_liquid_reserve.rs */
083 | pub fn handler<'info>(ctx: Context<'_, '_, '_, 'info, UpdateLiquidReserve<'info>>) -> Result<()> {
135 |     let lamports_for_pts = SoloValidatorBond::calculate_lamports_for_pt(
136 |         ctx.accounts.counter_party_pt.amount,
137 |         ctx.accounts.principal_token_mint.supply,
138 |         &mut ctx.accounts.bond,
139 |         &ctx.accounts.stake_account,
140 |         bond_rent,
141 |     )?;
142 |     let lmaports_for_yts = SoloValidatorBond::calculate_lamports_for_yt(
143 |         ctx.accounts.counter_party_yt.amount,
144 |         ctx.accounts.yield_token_mint.supply,
145 |         &mut ctx.accounts.bond,
146 |         &ctx.accounts.stake_account,
147 |         bond_rent,
148 |     )?;
```

2. Tokens/lamports discrepancy in redemption cache

The instruction `solo_validator_redeem_pt_for_sol` transfers `pt_amount` PT tokens from the redeemer to the global counterparty so they won't belong to the redemption cache. However, these PT tokens will not be burnt until the counterparty-side redemption instruction is executed. So the total YT token supply remains the same.

On the other hand, the instruction deducts the lamports represented by these tokens from `lamports_for_pts`, which makes the lamports and tokens tracked by the redemption cache consistent.

```
/* src/instructions/solo_validator/redeem_yt_for_sol.rs */
095 | pub fn handler(
096 |     ctx: Context<SoloValidatorRedeemYtForSol>,
097 |     args: SoloValidatorRedeemYtForSolArgs,
098 | ) -> Result<()> {
099 |     let yt_amount = args.amount;
100 |
101 |     // transfer YTs to the counterparty
102 |     ctx.accounts
103 |         .transfer_yt_from_user_to_counter_party(yt_amount)?;
104 |
105 |     // Calculate lamports for YT
106 |     let mut lamports_for_user = {
107 |         let bond_rent = Rent::get()?.minimum_balance(SoloValidatorBond::SIZE);
108 |         SoloValidatorBond::calculate_lamports_for_yt(
109 |             yt_amount,
110 |             ctx.accounts.yield_token_mint.supply,
111 |             &mut ctx.accounts.bond,
112 |             &ctx.accounts.stake_account,
113 |             bond_rent,
114 |         )?
115 |     };
```

However, the accounting issue occurs in other redemption cache based redemption operations. When calculating the value per PT token in `redeem_pts_for_lamports`, the assumption is that the `lamports_for_pts` should be lamports of the `pt_current_supply` PT tokens. In the implementation, `pt_current_supply` is the total supply of the PT tokens, while `redeem_pts_for_lamports` is the result after removing the value of the PT tokens transferred to the counterparty. As a result, the calculated value per PT token is smaller than the actual value.

Example:

Assuming the `lamports_for_pts` is `2000`, Alice holds 1 PT token while Bob holds another 1 PT token (so the total supply of PT tokens is 2), each user should be able to receive 1000 uint underlying assets. Consider the following happens:

Alice redeems 1 PT token and receives `1000` units of the underlying asset. The `lamports_for_pts` is updated to `1000` but the `yt_current_supply` does not change.

Bob redeems his 1 PT token, and the underlying asset amount is calculated as `1000 * 1 / 2 = 500` instead of `1000`.

3. Mismatches between `bond_total_lamports` and redemption cache lamports

As mentioned above, the instruction `solo_validator_redeem_pt_for_sol` transfers PT tokens from the redeemer to the counterparty, substracts their value from `redemption_cache.lamports_for_pts`, and transfers lamports from the counterparty to the redeemer.

Even though the total lamports in the cache change, the `bond_total_lamports` stays the same until the counterparty side redemption instruction is called. This inconsistency can introduce accounting errors in the subsequent redemption cache based redemption operations, which all call `calculate_lamports_for_pt` or `calculate_lamports_for_yt` to calculate lamports for the tokens.

```
/* src/state/solo_validator_bond.rs */
233 | pub fn calculate_lamports_for_pt(
234 |     pt_burned: u64,
235 |     pt_supply: u64,
236 |     bond_account: &mut Account<'_, Self>,
237 |     bond_stake: &AccountInfo,
238 |     rent_exempt_bond: u64,
239 | ) -> Result<u64> {
240 |     let bond_total_lamports = Self::total_lamports_owned_by_bond(
241 |         bond_account,
242 |         bond_stake,
243 |         rent_exempt_bond,
244 |         bond_account.transient_lamports,
245 |     );
246 |     bond_account.redemption_cache.redeem_pts_for_lamports(
247 |         pt_burned,
248 |         pt_supply,
249 |         bond_total_lamports,
250 |     )
251 | }

/* src/state/solo_validator_bond.rs */
069 | pub fn redeem_pts_for_lamports(
070 |     &mut self,
071 |     pt_amount: u64,
072 |     pt_current_supply: u64,
073 |     bond_total_lamports: u64,
074 | ) -> Result<u64> {
075 |     if self.was_slashed {
078 |     } else {
079 |         self.update_lamport_buckets(bond_total_lamports)?;
```

In particular, the accounting error is in the function `update_lamport_buckets`, which updates the

total lamports in the redemption cache to handle slashing or yields.

```
/* src/state/solo_validator_bond.rs */
026 | fn update_lamport_buckets(&mut self, bond_total_lamports: u64) -> Result<()> {
027 |     let old_total_lamports = self
028 |         .lamports_for_pts
029 |         .checked_add(self.lamports_for_yts)
030 |         .ok_or(ErrorCode::ArithmeticOverflow)?;
031 |     if old_total_lamports > bond_total_lamports {
048 |     } else if bond_total_lamports > old_total_lamports {
049 |         // More yield was generated, handle it!
050 |         let diff = bond_total_lamports
051 |             .checked_sub(old_total_lamports)
052 |             .ok_or(ErrorCode::ArithmeticOverflow)?;
053 |         let pt_increment = mul_div(diff, self.lamports_for_pts, old_total_lamports)?;
054 |         let yt_increment = diff
055 |             .checked_sub(pt_increment)
056 |             .ok_or(ErrorCode::ArithmeticOverflow)?;
057 |         self.lamports_for_pts = self
058 |             .lamports_for_pts
059 |             .checked_add(pt_increment)
060 |             .ok_or(ErrorCode::ArithmeticOverflow)?;
061 |         self.lamports_for_yts = self
062 |             .lamports_for_yts
063 |             .checked_add(yt_increment)
064 |             .ok_or(ErrorCode::ArithmeticOverflow)?;
065 |     }
066 |     Ok(())
067 | }
```

Since `bond_total_lamports` stays the same while `old_total_lamports` becomes smaller after deducting the value of the tokens transferred to the counterparty in the previous redemption, the `diff` at line 50 is essentially the lamports of the tokens to the counterparty. However, it's incorrectly treated as yields and distributed to the `lamports_for_pts` and `lamports_for_yts`.

Tests

The following test case illustrates the accounting errors introduced by the side effects of the redeemer side redemption for sol operation.

```
    #[test]
    fn test_pt_redemption_at_maturity() {
        let mut redemption_cache = SoloValidatorRedemptionCache {
            lamports_for_pts: 10 * LAMPORTS_PER_SOL,
            lamports_for_yts: 1 * LAMPORTS_PER_SOL,
            was_slashed: false,
            padding: Default::default(),
        };
```

```
        let pt_current_supply = 10 * LAMPORTS_PER_SOL;
        let bond_total_lamports = 11 * LAMPORTS_PER_SOL;

        let expected = 5 * LAMPORTS_PER_SOL;
        let expected_lamports_pt_cache = redemption_cache.lamports_for_pts - expected; // 5 *
        ↪  LAMPORTS_PER_SOL
        let pt_redemption_amount: u64 = 5 * LAMPORTS_PER_SOL;
        let res = redemption_cache
            .redeem_pts_for_lamports(pt_redemption_amount, pt_current_supply, bond_total_lamports)
            .unwrap();
        assert_eq!(res, expected);
        // Assert lamports_for_pts and lamports_for_yts was updated
        assert_eq!(
            redemption_cache.lamports_for_pts, // 5 * LAMPORTS_PER_SOL
            expected_lamports_pt_cache
        );
        assert_eq!(redemption_cache.lamports_for_yts, 1 * LAMPORTS_PER_SOL);

        let choose = 0;
        match choose {
            0 => {
                // @audit: may be one of correct implementations
                let pt_redemption_amount: u64 = 5 * LAMPORTS_PER_SOL;
                let pt_current_supply = 10 * LAMPORTS_PER_SOL - pt_redemption_amount;
                let bond_total_lamports = 11 * LAMPORTS_PER_SOL - pt_redemption_amount;
                let res = redemption_cache
                .redeem_pts_for_lamports(pt_redemption_amount, pt_current_supply, bond_total_lamports)
                .unwrap();
                println!("{}", res);
                assert_eq!(res, 5 * LAMPORTS_PER_SOL);
            },
            1 => {
                // current implementation
                let pt_redemption_amount: u64 = 5 * LAMPORTS_PER_SOL;
                let pt_current_supply = 10 * LAMPORTS_PER_SOL;
                let bond_total_lamports = 11 * LAMPORTS_PER_SOL;
                let res = redemption_cache
                    .redeem_pts_for_lamports(pt_redemption_amount, pt_current_supply,
                    ↪  bond_total_lamports)
                    .unwrap();
                println!("amount by current approach, {}", res);
            },
            _ => {}
        }
    }
```

The `choose` is used as a flag.

- `choose = 0`. Besides `lamports_for_pts`, also update the `pt_supply` and `bond_total_lamports`. The result is expected.
- `choose = 1`. The current `solo_validator_redeem_pt_for_sol`, where only `lamports_for_pts`

is updated. The result is incorrect.

Resolution

Fixed by commit `229814c62e776a68036919b66057dd4cfde37b26`.

BONDS

[H-03] Incorrect accounting in `counter_party_redeem_yt`

Identified in commit `9a2edb1395ee7e936ae3e355763d6e3ad7156a59`.

Instruction `counter_party_redeem_yt` is supposed to calculate `lamports_for_counter_party`, which is the lamports represented by the user-provided `yt_amount` YT tokens.

Then, it burns `yt_amount` YT tokens and transfers `lamports_for_counter_party` lamports from the bond to the counterparty.

However, in the implementation, when calculating `lamports_for_counter_party`, `counter_party_yt.amount` is used as the YT token amount instead of the burnt amount `yt_amount`.

```
/* src/instructions/counter_party/redeem_yt.rs */
098 | pub fn handler(ctx: Context<CounterPartyRedeemYt>, args: CounterPartyRedeemYtArgs) -> Result<()> {
099 |     let yt_amount = args.amount;
103 |     let lamports_for_counter_party = SoloValidatorBond::calculate_lamports_for_yt(
104 |         ctx.accounts.counter_party_yt.amount,
105 |         ctx.accounts.yield_token_mint.supply,
106 |         &mut ctx.accounts.bond,
107 |         &ctx.accounts.stake_account,
108 |         bond_rent,
109 |     )?;
110 |     // Burn the PTs
111 |     ctx.accounts.burn_yt(yt_amount)?;
```

Since `yt_amount <= counter_party_yt.amount`, the `counter_party` redeeming YT for SOL will receive more SOL than they should, resulting in a loss for other users.

Resolution

Fixed by commit `0949d1ae4ef4be47a1e2939e1c5247ad8e3347e3`.

BONDS

[H-04] Hijack the bond stake account, re-stake and steal funds

Identified in commit `996b908756ca027b4e1e92722f5f9a9a488b3da1`.

The `UpdateLiquidReserve` instruction attempts to withdraw all lamports from the bond stake account when `completely_unstaked`.

If successful, the bond stake account will transit to the `Uninitialized` state, making it vulnerable to hijacking, where anyone can set its `Staker` and `Withdrawer`.

```
/* bonds/src/instructions/counter_party/update_liquid_reserve.rs */
107 | pub fn handler<'info>(ctx: Context<'_, '_, '_, 'info, UpdateLiquidReserve<'info>>) -> Result<()> {
108 |     let bond_already_unstaked = ctx.accounts.handle_bond_already_unstaked()?;
109 |     if bond_already_unstaked {
110 |         return Ok(())
111 |     }

/* bonds/src/instructions/counter_party/update_liquid_reserve.rs */
088 | pub fn handle_bond_already_unstaked(&self) -> Result<bool> {
089 |     if self.bond.completely_unstaked {
090 |         // Attempt to withdraw.
091 |         stake_withdraw(
092 |             self.stake_account.to_account_info(),
093 |             self.bond.to_account_info(),
094 |             self.bond.to_account_info(),
095 |             self.clock.to_account_info(),
096 |             self.stake_history.to_account_info(),
097 |             Some(&[solo_validator_bond_signer_seeds!(self.bond)]),
098 |             self.stake_account.lamports(),
099 |         )?;
100 |         Ok(true)
101 |     } else {
102 |         Ok(false)
103 |     }
104 | }

/* src/stake_state.rs */
1135 | // Deinitialize state upon zero balance
1136 | if lamports == stake_account.get_lamports() {
1137 |     stake_account.set_state(&StakeStateV2::Uninitialized)?;
1138 | }
```

In the current implementation, the `SoloValidatorDelegateTips` instruction is unrestricted and can be called anytime, allowing execution even after the bond stake account has been completely unstaked.

This enables attackers to re-stake all lamports in the bond account (excluding rent).

PoC

1. After the bond stake account becomes completely unstaked, the attacker hijacks it and sets both `staker` and `withdrawer` to the attacker.
2. In a single transaction:
   - Use attacker's staker authority to delegate partial lamports to bond validator, setting account to `Activating` state.
   - Set staker authority back to bond account.
   - Execute `SoloValidatorDelegateTips` to redelegate bond account's lamports (now transfers lamports directly to bond stake account due to `Activating` state).
   - Reset staker authority to attacker.
3. Wait until stake account reaches `FullyActive` state, then deactivate using attacker's staker authority.
4. After full deactivation, withdraw lamports using the attacker's withdrawer authority.

In addition, the program may expect that the bond stake accounts will not be reused after they're completely unstaked, so it does not disable or close the stake account.

It is recommended to add post-processing logic for the bond stake account to ensure that the bond stake account will not be used again.

Resolution

Fixed by commit `f35f135b4ae51919c26bfa485069c149dc5b9a33`.

BONDS

[M-01] Fee evasions

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

In all fee related instructions, authorities of the `fee_wallet_pt` and `fee_wallet_yt` accounts are not validated, which allows users to evade the fees by passing in their own token accounts.

```
/* programs/bonds/src/instructions/deposit_shared.rs */
088 | #[account(
089 |     mut,
090 |     token::token_program = token_program,
091 | )]
092 | pub fee_wallet_pt: Box<InterfaceAccount<'info, TokenAccount>>,
093 | #[account(
094 |     mut,
095 |     token::token_program = token_program,
096 | )]
097 | pub fee_wallet_yt: Box<InterfaceAccount<'info, TokenAccount>>,

/* programs/bonds/src/instructions/solo_validator/deposit_sol.rs */
103 | #[account(
104 |     mut,
105 |     token::token_program = token_program,
106 | )]
107 | pub fee_wallet_pt: Box<InterfaceAccount<'info, TokenAccount>>,
108 | #[account(
109 |     mut,
110 |     token::token_program = token_program,
111 | )]
112 | pub fee_wallet_yt: Box<InterfaceAccount<'info, TokenAccount>>,

/* programs/bonds/src/instructions/solo_validator/deposit_stake.rs */
084 | #[account(
085 |     mut,
086 |     token::token_program = token_program,
087 | )]
088 | pub fee_wallet_pt: Box<InterfaceAccount<'info, TokenAccount>>,
089 | #[account(
090 |     mut,
091 |     token::token_program = token_program,
092 | )]
093 | pub fee_wallet_yt: Box<InterfaceAccount<'info, TokenAccount>>,
```

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[M-02] DoS in `counter_party_update_liquid_reserve`

Identified in commit `0e8730083dc090f389e29dc84c6b940e7b9427cd`.

When the `bond.transient_stake_account` is in the `Inactive` state, the `counter_party_update_liquid_reserve` instruction will withdraw its entire balance.

```
/* programs/bonds/src/instructions/counter_party/update_liquid_reserve.rs */
116 | StakeStatus::Inactive => {
117 |     ctx.accounts
118 |         .bond
119 |         .check_transient_stake_address(&transient_stake_account)?;
120 |     stake_withdraw(
121 |         transient_stake_account.to_account_info(),
122 |         ctx.accounts.bond.to_account_info(),
123 |         ctx.accounts.bond.to_account_info(),
124 |         ctx.accounts.clock.to_account_info(),
125 |         ctx.accounts.stake_history.to_account_info(),
126 |         Some(&[solo_validator_bond_signer_seeds!(ctx.accounts.bond)]),
127 |         transient_stake_account.lamports(),
128 |     )?;

/* programs/bonds/src/state/solo_validator_bond.rs */
146 | pub fn check_transient_stake_address(
147 |     &self,
148 |     transient_stake_account: &AccountInfo<'_>,
149 | ) -> Result<()> {
150 |     require!(
151 |         transient_stake_account.key() == self.transient_stake_account,
152 |         ErrorCode::InvalidTransientStakeAccount
153 |     );
154 |     Ok(())
155 | }
```

After the `stake_withdraw()`, the transient stake account will transit to the `Uninitialized` state.

```
/* the current mainnet cluster version is 2.0.19  */
/* https://github.com/anza-xyz/agave/blob/v2.0.19/programs/stake/src/stake_state.rs#L1067-L1070 */
0974 | pub fn withdraw(
0985 | ) -> Result<(), InstructionError> {
1067 |   // Deinitialize state upon zero balance
1068 |   if lamports == stake_account.get_lamports() {
1069 |     stake_account.set_state(&StakeStateV2::Uninitialized)?;
1070 |   }
```

In addition, if `lamports_to_unstake = 0`, the `update_liquid_reserve` handler exits early without

any other operations:

```
/* programs/bonds/src/instructions/counter_party/update_liquid_reserve.rs */
162 | if lamports_to_unstake == 0 {
163 |     return Ok(());
164 | }
```

As a result, the transient stake account remains in the `Uninitialized` state, even though it is still referenced as the `transient_stake_account` in the Bond account. This prevents future `update_liquid_reserve` instructions on the bond from executing successfully:

1. Reusing the transient stake account. The `update_liquid_reserve` instruction only accepts transient stake accounts in the `Empty` or `Inactive` state. If the account is in the `Uninitialized` state, the instruction will fail.

```
/* programs/bonds/src/instructions/counter_party/update_liquid_reserve.rs */
133 | StakeStatus::Unitialized => return Err(ErrorCode::InvalidTransientStakeAccount.into()),
```

2. Using a new keypair for the transient stake account. Since the bond account still references the previous transient stake account's public key, the `check_transient_stake_unset` validation will fail, causing the instruction to fail.

```
/* programs/bonds/src/instructions/counter_party/update_liquid_reserve.rs */
101 | StakeStatus::Empty => {
102 |     // Validate that there is no transient account on the bond
103 |     ctx.accounts.bond.check_transient_stake_unset()?;
```

If the intended behavior is to use a new keypair as the transient stake account, the handler should immediately unset the `bond.transient_stake_account`.

Resolution

Fixed by commit `a5af113efe84b6f2acca0e96d9ee8f43cf3ce3de`.

BONDS

[M-03] Post-maturity `SoloValidatorDelegateTips` blocks `UpdateLiquidReserve`

Identified in commit `ce43f680998713428007b8eb8d5c65e7ecd44221`.

The `SoloValidatorDelegateTips` instruction allows compounding MEV tips by re-staking undelegated lamports from a bond stake account. The flow is:

```
bond stake account's mev tip (undelegated amount)
  (withdraw to) -> bond account (minus rent)
  (re-stake to) -> bond stake account
```

It can be called by anyone.

Also, `SoloValidatorDelegateTips` can be called at any time (even after bond maturity via `SoloValidatorHandleMaturity`). And if invoked post-maturity, it creates a new transient_stake_account with a stake state of `Activating` or `FullyActive` via `stake_sol!`.

However, `SoloValidatorHandleMaturity` is a one-time operation (due to the maturity_handled flag):

```
/* programs/bonds/src/instructions/solo_validator/handle_maturity.rs */
056 | if ctx.accounts.bond.maturity_handled {
057 |     return Ok(());
058 | }
```

But the `UpdateLiquidReserve` instruction requires the transient_stake_account to be in `Empty` or `Inactive` state:

```
/* programs/bonds/src/instructions/counter_party/update_liquid_reserve.rs */
095 | match transient_status {
126 |   StakeStatus::Unitialized => return Err(ErrorCode::InvalidTransientStakeAccount.into()),
127 |   StakeStatus::Activating => return Err(ErrorCode::InvalidTransientStakeAccount.into()),
128 |   StakeStatus::FullyActive => return Err(ErrorCode::InvalidTransientStakeAccount.into()),
```

An attacker can intentionally trigger `SoloValidatorDelegateTips` post-maturity, leaving the transient_stake_account in an `Activating` / `FullyActive` state. This permanently blocks `UpdateLiquidReserve` from withdrawing lamports from the bond stake account, causing a DoS for reserve

updates.

It is recommended to add a time-bound check in `SoloValidatorDelegateTips` according to design requirements.

Resolution

Fixed by commit `f35f135b4ae51919c26bfa485069c149dc5b9a33`.

BONDS

[L-01] Unchecked stake pool programs in `StakePoolBond` initialization

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

The program is intended to support various backends (e.g., SPL Stake Pool, Marinade).

However, the `from_account_info()` does not validate the SPL Stake Pool program ID, allowing any non-Marinade program to be treated as the SPL Stake Pool program and initialized as a StakePoolBond:

```
/* programs/bonds/src/state/stake_pool_bond.rs */
081 | /// Given an account, deserialized the data into the appropriate LstState
082 | pub fn from_account_info(account_info: &AccountInfo) -> Result<Self> {
083 |     if account_info.owner.eq(&marinade::ID) {
084 |         let mut data = &account_info.try_borrow_data().unwrap()[8..];
085 |         let marinade_state = marinade::State::deserialize(&mut data)?;
086 |         Ok(LstState::MarinadeState(marinade_state))
087 |     } else {
088 |         let stake_pool = try_from_slice_unchecked::<StakePool>(&account_info.data.borrow())?;
089 |         Ok(LstState::StakePool(stake_pool))
090 |     }
091 | }
```

Additionally, in `deposit_sol`, the program directly issues a CPI call with the user's signature to `bond.lst_program`:

```
/* programs/bonds/src/instructions/deposit_sol.rs */
011 | /// CHECK: Handled
012 | #[account(
013 |   address = deposit_shared.bond.lst_program
014 | )]
015 | pub stake_pool_program: UncheckedAccount<'info>,

030 | pub fn stake_pool_deposit_sol(&self, amount: u64) -> Result<()> {
031 |   // CPI tp Deposit SOL into LST, LST should be transferred to the vault
032 |   let deposit_sol_ix = deposit_sol(
033 |     &self.stake_pool_program.key,
034 |     &self.deposit_shared.stake_pool.key,
035 |     &self.stake_pool_withdraw_authority.key,
036 |     &self.reserve_stake_account.key,
037 |     &self.deposit_shared.owner.key,
038 |     &self.deposit_shared.lst_vault.key(),
039 |     &self.manager_fee_account.key,
040 |     &self.referrer_pool_tokens_account.key,
```

```
041 |     &self.deposit_shared.lst_mint.key(),
042 |     &self.deposit_shared.lst_token_program.key(),
043 |     amount,
044 |   );
045 |   invoke(&deposit_sol_ix, &vec![
046 |     self.stake_pool_program.to_account_info(),
047 |     self.deposit_shared.stake_pool.to_account_info(),
048 |     self.stake_pool_withdraw_authority.to_account_info(),
049 |     self.reserve_stake_account.to_account_info(),
050 |     self.deposit_shared.owner.to_account_info(),
051 |     self.deposit_shared.lst_vault.to_account_info(),
052 |     self.manager_fee_account.to_account_info(),
053 |     self.referrer_pool_tokens_account.to_account_info(),
054 |     self.deposit_shared.lst_mint.to_account_info(),
055 |     self.deposit_shared.lst_token_program.to_account_info()
056 |   ])?;
057 |   Ok(())
058 | }

/* spl-stake-pool-2.0.0/src/instruction.rs */
1943 | /// Creates instructions required to deposit SOL directly into a stake pool.
1944 | fn deposit_sol_internal(
1958 | ) -> Instruction {
1959 |     let mut accounts = vec![
1963 |         AccountMeta::new(*lamports_from, true),
1970 |     ];
```

As a result, an attacker could craft a malicious stake pool program to gain control over the user's signature and perform arbitrary instructions.

```
/* programs/bonds/src/instructions/initialize_stake_pool_bond.rs */
105 | // REVIEW: Should there be a whitelist of StakePool programs?
```

As suggested in the comment, a whitelist for program IDs could be implemented to prevent this.

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[L-02] Unhandled Token 2022 mints with the `TransferFee` extension

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

In `deposit`, users can deposit LST into the program-managed `lst_vault` to receive PT and YT.

```
/* programs/bonds/src/instructions/deposit.rs */
019 | pub fn transfer_lst_from_owner_to_vault(&self, amount: u64) -> Result<()> {
030 |     token_interface::transfer_checked(cpi_ctx, amount, self.deposit_shared.lst_mint.decimals)
031 | }

/* programs/bonds/src/instructions/deposit.rs */
039 | pub fn handler(ctx: Context<Deposit>, args: DepositArgs) -> Result<()> {
040 |     // Transfer amount of LST from owner to the Bond's lst vault
041 |     ctx.accounts.transfer_lst_from_owner_to_vault(args.amount)?;
043 |     deposit_shared::handler(&ctx.accounts.deposit_shared, args.amount)
044 | }
```

In the `deposit_shared` handler, `args.amount` is directly used to calculate the amount of PT and YT. However, the stake pool's pool token supports Token 2022 with the `TransferFeeConfig` extension:

```
/* stake-pool/program/src/state.rs */
501 | /// Checks if the given extension is supported for the stake pool mint
502 | pub fn is_extension_supported_for_mint(extension_type: &ExtensionType) -> bool {
503 |     const SUPPORTED_EXTENSIONS: [ExtensionType; 8] = [
504 |         ExtensionType::Uninitialized,
505 |         ExtensionType::TransferFeeConfig,
506 |         ExtensionType::ConfidentialTransferMint,
507 |         ExtensionType::ConfidentialTransferFeeConfig,
508 |         ExtensionType::DefaultAccountState, // ok, but a freeze authority is not
509 |         ExtensionType::InterestBearingConfig,
510 |         ExtensionType::MetadataPointer,
511 |         ExtensionType::TokenMetadata,
512 |     ];
513 |     if !SUPPORTED_EXTENSIONS.contains(extension_type) {
514 |         msg!(
515 |             "Stake pool mint account cannot have the {:?} extension",
516 |             extension_type
517 |         );
518 |         false
519 |     } else {
520 |         true
521 |     }
522 | }
```

For the LST mint with the `TransferFeeConfig` extension enabled, the amount that the `lst_vault` receives will be less than `args.amount`. Consequently, the calculated PT and YT amounts will exceed the actual distributable amount.

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[L-03] Rounding up fees to prevent fee evasion

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

In all areas where fees are calculated, the division in the calculation is rounded down, which may result in the received fee being lower than expected.

```
/* programs/bonds/src/instructions/deposit_shared.rs */
150 | pub fn handler(accounts: &DepositShared, lst_amount: u64) -> Result<()> {
161 |     let fees_pt = U128::from(amount_pt)
162 |         .checked_mul(deposit_fee_bps)
163 |         .expect("overflow")
164 |         .checked_div(bps_denominator)
165 |         .expect("overflow")
166 |         .as_u64();
167 |     let fees_yt = U128::from(amount_yt)
168 |         .checked_mul(deposit_fee_bps)
169 |         .expect("overflow")
170 |         .checked_div(bps_denominator)
171 |         .expect("overflow")
172 |         .as_u64();
173 |     // mint PT to fee account
174 |     accounts.mint_pt(accounts.fee_wallet_pt.to_account_info(), fees_pt)?;
175 |     accounts.mint_yt(accounts.fee_wallet_yt.to_account_info(), fees_yt)?;
183 | }

/* programs/bonds/src/macros.rs */
133 | // Calculate fees from the total PT and YT
134 | let deposit_fee_bps = U128::from($accounts.global_settings.deposit_fee_bps);
135 | let bps_denominator = U128::from(10_000);
136 | let fees_pt = U128::from(amount_pt)
137 |     .checked_mul(deposit_fee_bps)
138 |     .expect("overflow")
139 |     .checked_div(bps_denominator)
140 |     .expect("overflow")
141 |     .as_u64();
142 | let fees_yt = U128::from(amount_yt)
143 |     .checked_mul(deposit_fee_bps)
144 |     .expect("overflow")
145 |     .checked_div(bps_denominator)
146 |     .expect("overflow")
147 |     .as_u64();

/* programs/bonds/src/instructions/solo_validator/redeem_pt_for_sol.rs */
109 | let fee = U128::from(lamports_for_user)
110 |     .checked_mul(U128::from(
111 |         ctx.accounts.global_settings.counter_party_fee_bps,
112 |     ))
113 |     .expect("overflow")
```

```
114 |     .checked_div(U128::from(10_000))
115 |     .expect("overflow")
116 |     .as_u64();
```

In fact, due to the absence of a minimum deposit check for LST tokens in deposit_shared.rs, malicious users can repeatedly deposit dust amounts to bypass fee charges mechanism.

Consider rounding up the fees.

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[L-04] Missing `issuance_ts < maturity_ts` checks

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

When initializing bonds, currently only `maturity_ts > now` is validated. It is recommended to add a constraint ensuring `issuance_ts < maturity_ts - seconds_per_epoch`.

```
/* programs/bonds/src/instructions/initialize_stake_pool_bond.rs */
117 | // Validate maturity is in the future
118 | require!(args.maturity_ts > now, ErrorCode::PastMaturity);

/* programs/bonds/src/instructions/initialize_stake_pool_bond.rs */
179 | bond.issuance_ts = args.issuance_ts;
180 | bond.maturity_ts = args.maturity_ts;
```

If `issuance_ts > maturity_ts`, it will fail when calculating YT due to an overflow during the calculation.

```
/* programs/bonds/src/state/stake_pool_bond.rs */
170 | let denominator = U192::try_from(
171 |     self.maturity_ts
172 |         .checked_sub(self.issuance_ts)
173 |         .expect("underflow"),
174 | )
175 | .expect("u64");
```

Additionally, in the `validate` function of `deposit_sol.rs`, it is required that the user must deposit at least one epoch before `deposit_ends`.

```
/* programs/bonds/src/instructions/solo_validator/deposit_sol.rs */
133 | pub fn validate(
134 |         ctx: &Context<SoloValidatorDepositSol>,
135 |         args: &SoloValidatorDepositSolArgs,
136 |     ) -> Result<()> {
137 |         let now = Clock::get()?.unix_timestamp;
138 |         // Validate that Bond maturity is in the future by about 1 epoch
139 |         // Note: This could be improved by checking if next epoch boundary is beyond the bond's
140 |         //  maturity date.
141 |         require!(
142 |             now.lt(&ctx.accounts.bond.deposit_ends_ts()?),
143 |             ErrorCode::BondAlreadyMatured
144 |         );
146 |         // Enforce the 1 SOL minimum deposit. This is also enforced at the Stake program level
```

```
147 |         require!(args.amount > LAMPORTS_PER_SOL, ErrorCode::BelowMinimum);
151 |         Ok(())
152 |     }
```

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[L-05] `InsufficientFunds` error due to partial unstake amount

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

In the stake program's split instruction, the balance of the source stake account is checked.

If the balance is not zero and is less than the `source_minimum_balance`, it will return an `InsufficientFunds` error.

```
/* src/stake_state.rs */
946 | } else if source_remaining_balance < source_minimum_balance {
947 |     // the remaining balance is too low to do the split
948 |     return Err(InstructionError::InsufficientFunds);
```

Here, the `source_minimum_balance` is the sum of the stake account's rent-exempt reserve and the minimum required balance for delegation (`stake::get_minimum_delegation()`)

```
/* src/stake_state.rs */
939 | let source_minimum_balance = source_meta
940 |     .rent_exempt_reserve
941 |     .saturating_add(additional_required_lamports);
```

However, in the program:

```
/* programs/bonds/src/instructions/counter_party/update_liquid_reserve.rs */
163 | let source_stake_after_split = ctx
164 |     .accounts
165 |     .stake_account
166 |     .lamports()
167 |     .saturating_sub(lamports_to_unstake)
168 |     .saturating_sub(stake_rent);
169 |
170 | lamports_to_unstake = if source_stake_after_split < minimum_stake_account_amt {
171 |     source_stake_after_split
172 | } else {
173 |     lamports_to_unstake
174 | };
```

When `source_stake_after_split < minimum_stake_account_amt`, the program returns `source_stake_after_split` as the unstake amount, which will result in an `InsufficientFunds` error in stake split instruction.

Consider returning the entire balance of the stake account as the unstake amount, when `source_stake_after_split < minimum_stake_account_amt`.

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[L-06] The `counter_party_pt` and `counter_party_yt` should be ATAs

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

In the `solo_validator::redeem_for_sol` instruction, a user's PT and YT are transferred to the token account of the `counter_party`, and SOL is then transferred out from the `counter_party`.

However, the `counter_party_pt` and `counter_party_yt` accounts do not have to be ATAs.

As a result, any token account owned by the `counter_party` to be passed in, which can scatter the PT and YT collected by the `counter_party` in multiple token accounts, complicating their subsequent operations. For example, the counterparty needs to call `counter_party::redeem_pt` and `counter_party::redeem_yt` to get the lamports from the bond.

```
/* programs/bonds/src/instructions/solo_validator/redeem_pt_for_sol.rs */
043 | #[account(
044 |     mut,
045 |     token::mint = principal_token_mint,
046 |     token::token_program = token_program,
047 | )]
048 | pub counter_party_pt: InterfaceAccount<'info, TokenAccount>,

/* programs/bonds/src/instructions/solo_validator/redeem_yt_for_sol.rs */
056 | #[account(
057 |     mut,
058 |     token::token_program = token_program,
059 | )]
060 | pub counter_party_yt: InterfaceAccount<'info, TokenAccount>,
```

Resolution

Fixed by commit `ac441d017e1ea8ec022afc082b2378152f6c8563`.

BONDS

[L-07] Missing `can_redeem` check in `counter_party` instructions

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

Several counterparty-related instructions (`CounterPartyRedeemPt`, `CounterPartyRedeemYt`, `UpdateLiquidReserve`, `CounterPartyWithdrawSol`) are intended to be executed only after the `handle_maturity` instruction, but they do not enforce the `can_redeem()` check.

Additionally, the `update_liquid_reserve` instruction lacks validation for the `counter_party_pt` and the `counter_party_yt`, potentially allowing attackers to supply arbitrary PT/YT accounts, and then deactivate some lamports from the bond stake account and update the bond account prematurely.

```
/* programs/bonds/src/instructions/counter_party/update_liquid_reserve.rs */
031 | #[account(
032 |     token::token_program = token_program,
033 | )]
034 | pub counter_party_pt: InterfaceAccount<'info, TokenAccount>,
035 | #[account(
036 |     token::token_program = token_program,
037 | )]
038 | pub counter_party_yt: InterfaceAccount<'info, TokenAccount>,

130 | let lmaports_for_pts =
131 |     SoloValidatorBond::calculate_lamports_for_pt(ctx.accounts.counter_party_pt.amount);

186 | // Fire off instructions to split stake to transient
187 | stake_split(
188 |     &ctx.accounts.bond,
189 |     ctx.accounts.stake_account.to_account_info(),
190 |     ctx.accounts.bond.to_account_info(),
191 |     lamports_to_unstake,
192 |     ctx.accounts.transient_stake_account.to_account_info(),
193 | )?;
```

In this scenario, an attacker could deactivate the bond stake account's stake amount before bond maturity. While the attacker cannot directly claim the deactivated stake amount, it reduces the staking yield of the bond.

Furthermore, because the `transient_stake_account` is reused by the program, prematurely call-

ing `update_liquid_reserve` might interfere with the execution of the `solo_validator::deposit_sol` instruction.

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[L-08] Inaccurate PT token mint amount in the `SoloValidator`

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

When a user deposits SOL to the solo validator, they may need to pay a rent fee to create a transient account before staking a certain amount of SOL.

However, the protocol currently mints PT tokens to the user based solely on the staked amount. Since PT tokens represent the principal amount deposited by the user, the rent fee paid by the user should also be included in the minted PT tokens.

```
/* programs/bonds/src/instructions/solo_validator/deposit_sol.rs */
427 | StakeStatus::FullyActive => {
428 |     // merge the transient stake into the bond
429 |     ctx.accounts.stake_merge(
430 |         ctx.accounts.stake_account.to_account_info(),
431 |         ctx.accounts.transient_stake_account.to_account_info(),
432 |         ctx.accounts.bond.to_account_info(),
433 |         &ctx.accounts.bond,
434 |     )?;
435 |     // transfer stake amount + rent to the transient account
436 |     let rent_exempt = ctx.accounts.rent.minimum_balance(StakeStateV2::size_of());
437 |     transfer_lamports(
438 |         ctx.accounts.owner.to_account_info(),
439 |         ctx.accounts.transient_stake_account.to_account_info(),
440 |         args.amount + rent_exempt,
441 |         None,
442 |     )?;
443 |     // Then re-initialize the transient stake account
444 |     ctx.accounts
445 |         .stake_initialize(ctx.accounts.transient_stake_account.to_account_info())?;
446 |     // Delegate it to the Bond's validator
447 |     ctx.accounts.delegate_stake_account(
448 |         ctx.accounts.transient_stake_account.to_account_info(),
449 |     )?;
450 | }

/* programs/bonds/src/instructions/solo_validator/deposit_sol.rs */
476 | // Handles calculating and minting the PTs and YTs
477 |     calc_and_mint_pt_yt!(ctx.accounts, args.amount, est_time_to_stake);
```

Consider accounting for the rent fee paid by the user when minting PT tokens.

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[L-09] Incorrect `SoloValidatorBond` account size

Identified in commit `0e8730083dc090f389e29dc84c6b940e7b9427cd`.

In the `redeem_pt` function of the `counter_party` section, it uses `std::mem::size_of::<SoloValidatorBond>()` to calculate the size of the SoloValidatorBond account.

It fails to include the space required for the `SoloValidatorBond` account discriminator.

```
/* programs/bonds/src/instructions/counter_party/redeem_pt.rs */
105 | let bond_rent = Rent::get()?.minimum_balance(std::mem::size_of::<SoloValidatorBond>());
```

Consider replacing `std::mem::size_of::<SoloValidatorBond>()` with `SoloValidatorBond::SIZE`.

Resolution

Fixed by commit `a5af113efe84b6f2acca0e96d9ee8f43cf3ce3de`.

BONDS

[L-10] Outdated `stake_pool.total_lamports"`

Identified in commit `ce43f680998713428007b8eb8d5c65e7ecd44221`.

When using the StakePool Program's `stake_pool.total_lamports` to calculate `lamports_for_lst` for an LST Bond, the `stake_pool.total_lamports` should be updated in real time. Otherwise, using outdated data may result in a calculated `lamports_for_lst` that is too low.

This could happen because, after some epochs, rewards earned from validator stakes will increase the actual total lamports. As a result, users could receive fewer PT/YT than expected, leading to a loss on the user side.

```
/* programs/bonds/src/state/stake_pool_bond.rs */
116 | pub fn lamports_for_lst(&self, deposit_amount: u64) -> u64 {
117 |     match &self {
118 |         LstState::StakePool(stake_pool) => stake_pool
119 |             .calc_lamports_withdraw_amount(deposit_amount)
120 |             .expect("calc pool lamports"),

/* programs/spl-stake-pool-2.0.0/src/state.rs */
178 | pub fn calc_lamports_withdraw_amount(&self, pool_tokens: u64) -> Option<u64> {
182 |     let numerator = (pool_tokens as u128).checked_mul(self.total_lamports as u128)?;
183 |     let denominator = self.pool_token_supply as u128;
```

If the `deposit_sol` CPI is called from the pyefi program, the `stake_pool.last_update_epoch` is checked by the stake pool program.

However, pyefi also provides a deposit lst instruction, which directly transfers the user's LST to the pyefi bond vault without any CPI call to the stake pool. In this case, the `stake_pool.total_lamports` obtained might be outdated.

This issue also occurs in the `LstBondHandleMaturity` instruction, as it similarly uses `stake_pool.total_lamports` without any CPI call or constraint to ensure that the `stake_pool` state is up to date.

```
/* programs/bonds/src/state/stake_pool_bond.rs */
222 | if !self.is_redemption_conversion_set() {
223 |     let lst_for_all_pt = RedemptionCache::calc_lst_for_all_pt(
224 |         pt_mint_supply,
225 |         lst_vault_balance,
226 |         lst_state.total_lamports(),
227 |         stake_pool_tokens_supply,
228 |     );

/* programs/bonds/src/state/stake_pool_bond.rs */
128 | pub fn total_lamports(&self) -> u64 {
129 |     match &self {
130 |         LstState::StakePool(stake_pool) => stake_pool.total_lamports,
```

It is recommended to add a constraint: `stake_pool.last_update_epoch == clock.epoch` or just call `update_stake_pool_balance` CPI before using `stake_pool.total_lamports`.

Resolution

Fixed by commit `36fddadb5b9d20a1bf3026b27b83ea670162894b`.

BONDS

[L-11] Deactivated but not withdrawn stake

Identified in commit `cc4ae0e385d4c5c4cce1780fc88cee16d9f59d5e`.

When the `bond.completely_unstaked` is set to `true`, the program de-activates the stake in the `stake_account` by calling `stake_deactivate()`.

```
/* bonds/src/instructions/counter_party/update_liquid_reserve.rs */
107 | pub fn handler<'info>(ctx: Context<'_, '_, '_, 'info, UpdateLiquidReserve<'info>>) -> Result<()> {
215 |     // If we need to unstake the rest of the Bond, deactivate the Bond's stake account and short
216 |     //  circuit
217 |     if source_stake_after_split < minimum_stake_account_amt {
218 |         let bond = &mut ctx.accounts.bond;
219 |         bond.completely_unstaked = true;
220 |         return stake_deactivate(
221 |             ctx.accounts.stake_account.to_account_info(),
222 |             ctx.accounts.bond.to_account_info(),
223 |             ctx.accounts.clock.to_account_info(),
224 |             Some(&[solo_validator_bond_signer_seeds!(ctx.accounts.bond)]),
225 |         )
226 |     };
```

In `total_lamports_owned_by_bond()`, when `bond_account.completely_unstaked` is `true`, the `staked_lamports` is calculated as `0`.

```
/* bonds/src/state/solo_validator_bond.rs */
167 | pub fn total_lamports_owned_by_bond(
168 |     bond_account: &Account<'_, Self>,
169 |     bond_stake: &AccountInfo,
170 |     bond_rent_exemption: u64,
171 |     transient_stake: u64,
172 | ) -> u64 {
173 |     // Conditional check mitigates edge case where adversary bricks SOL redemptions
174 |     let staked_lamports = if bond_account.completely_unstaked {
175 |         0
176 |     } else {
188 |     };
190 |     // The total lamports owned by the bond is
191 |     //   1. Liquid reserve (bond account's lamports sans rent)
192 |     //   2. Transient lamports (sol possibly moving from the bond stake account to the liquid
193 |     //      reserve)
194 |     //   3. The bond's core stake account lamports
195 |     bond_account
196 |         .get_lamports()
197 |         .saturating_sub(bond_rent_exemption)
198 |         .saturating_add(transient_stake)
```

```
199 |             .saturating_add(staked_lamports)
200 | }
```

However, after the stake de-activation and before withdrawals, the lamports are still in the `stake_account`, which is still owned by the bond.

During this period, if the `counter_party` performs PT/YT redemption operations, the `total_lamports_owned_by_bond` does not consider the lamports that are unstaked but not withdrawn.

Resolution

Fixed by commit `ce4005268f779fe64130b0fb023597c08dc2c2d6`.

BONDS

[I-01] Potential arithmetic overflow in `calc_pt_to_mint`

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

The `principal_lamports`, `PT_YT_DECIMAL_FACTOR`, and `LAMPORTS_PER_SOL` are all `u64` integers.

```
/* programs/bonds/src/state/stake_pool_bond.rs */
158 | principal_lamports
159 |     .checked_mul(PT_YT_DECIMAL_FACTOR)
160 |     .expect("overflow")
161 |     .checked_div(LAMPORTS_PER_SOL)
162 |     .expect("overflow")
```

It is recommended to promote the calculations in `LstBond::calc_pt_to_mint` to `u128` or `u192` to prevent overflow, which could otherwise result in execution failure.

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[I-02] Validate the `validator_vote_account`

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

Consider adding a constraint to validate that the owner of the `validator_vote_account` is the vote program.

```
/* programs/bonds/src/instructions/initialize_solo_validator_bond.rs */
068 | impl<'info> InitializeSoloValidatorBond<'info> {
069 |     pub fn validate(
070 |         ctx: &Context<InitializeSoloValidatorBond>,
071 |         args: &InitializeSoloValidatorBondArgs,
072 |     ) -> Result<()> {
073 |       let now = Clock::get()?.unix_timestamp;
074 |         // Validate the validator vote account
075 |         let _ = VoteState::deserialize(
076 |             &ctx.accounts
077 |                 .validator_vote_account
078 |                 .try_borrow_data()
079 |                 .unwrap(),
080 |         )
081 |         .map_err(|_| ErrorCode::NoteActiveVoteAccount)?;
082 |
083 |         // Validate maturity is in the future
084 |         require!(args.maturity_ts > now, ErrorCode::PastMaturity);
085 |         Ok(())
086 |     }
```

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[I-03] Validate `deposit_fee_bps` and `counter_party_fee_bps`

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

It should ensure that `args.deposit_fee_bps` and `args.counter_party_fee_bps` are less than `10_000` during the initialization or fee update process.

```
/* programs/bonds/src/instructions/update_fee_settings.rs */
029 | pub fn handler(ctx: Context<UpdateFeeSettings>, args: UpdateFeeSettingsArgs) -> Result<()> {
030 |     let global = &mut ctx.accounts.global_settings;
032 |     global.fee_admin = args.fee_admin;
033 |     global.protocol_fee_wallet = args.fee_wallet;
034 |     global.deposit_fee_bps = args.deposit_fee_bps;
035 |     global.counter_party_fee_bps = args.counter_party_fee_bps;
037 |     Ok(())
038 | }
```

Consider adding relevant checks in the `update_fee_settings.rs` and `Initialize_global_settings.rs`.

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[I-04] Move the definition of `ephemeral_stake_account` to the activation scope

Identified in commit `ce5f5e82da10ba77a42ca892e1b5aa945a8a45c3`.

Since the `ephemeral_stake_account` is only used when the `bond_stake_status` is `StakeStatus::FullyActive` and the `transient_stake_status` is `StakeStatus::Activating`, its definition can be moved into the activating scope.

```
/* programs/bonds/src/instructions/solo_validator/deposit_sol.rs */
383 | StakeStatus::FullyActive => {
384 |     let ephemeral_stake_account = ctx.accounts.check_ephemeral_stake()?;

411 | StakeStatus::Activating => { // ActivationEpoch in fact
412 |     SoloValidatorDepositSol::create_stake_account(
413 |         &ctx,
414 |         &ephemeral_stake_account,
415 |         args.amount,
416 |         false,
417 |         &[],
418 |     )?;
419 |     // merge the stake
420 |     ctx.accounts.stake_merge( // Inactive -> ActivationEpoch
421 |         ctx.accounts.transient_stake_account.to_account_info(),
422 |         ephemeral_stake_account.to_account_info(),
423 |         ctx.accounts.bond.to_account_info(),
424 |         &ctx.accounts.bond,
425 |     )?;
426 | }
```

Resolution

Fixed by commit `3edf2d78e7251fb98486f4918df648b7e9b7d73c`.

BONDS

[I-05] Overdrawing the bond account

Identified in commit `996b908756ca027b4e1e92722f5f9a9a488b3da1`.

Currently, the `delegate_tips` instruction can be called at any time by anyone. It stakes the excess lamports in `bond` (total lamports minus rent) and the unstaked lamports in `stake_account`.

First, it checks the state of the `stake_account`. If the stake account is in `StakeStateV2::Stake` state, it withdraws its unstaked lamports minus `rent_exempt_reserve` to the `bond` account.

```
/* bonds/src/instructions/solo_validator/delegate_tips.rs */
061 | pub fn handler<'info>(
062 |     ctx: Context<'_, '_, '_, 'info, SoloValidatorDelegateTips<'info>>,
063 | ) -> Result<()> {
064 |     // Withdraw excess lamports from stake account to the bond's account
065 |     withdraw_bond_stake_exccess(
066 |         ctx.accounts.stake_account.to_account_info(),
072 |     )?;
088 |     match transient_stake_status {
089 |         StakeStatus::Empty | StakeStatus::FullyActive => {
090 |             excess_lamports = excess_lamports.saturating_sub(stake_account_rent)
091 |         }
092 |         _ => {}
093 |     }
094 |
095 |     stake_sol!(
096 |         ctx,
097 |         excess_lamports,
098 |         transient_stake_account,
099 |         ctx.accounts.bond
100 |     );
```

Please note that `stake_account` may not be created yet (or `StakeStatus::Empty` status), depending on when the `delegate_tips` instruction is called.

Then, in `stake_sol!()` at line 95, depending on the status of the `stake_account` and `transient_stake_account`, `excess_lamports` at line 97 together with **additional** rent will be transferred away from the `bond` account.

To compensate the rent overhead, at line 90, the `stake_account_rent` is subtracted from the `excess_lamports` if the `transient_stake_account` is in the state of StakeStatus::Empty or `StakeStat`

`us::FullyActive`. However, it misses two more scenarios.

In particular, there are four scenarios `stake_sol!()` transfers `excess_lamports` (the updated instance at line 97) + `stake_account_rent` from the `bond` account.

1. `stake_account` - `Empty`

At line 201, since the parameter `add_rent` is set to `true`, it will transfer `$staked_amount + rent_exempt` from the `bond` account to the `stake_account` account.

```
/* bonds/src/macros.rs */
187 | macro_rules! stake_sol {
188 |     ($ctx:expr, $staked_amount:expr, $transient_stake_account:expr, $owner:expr) => {
197 |         match bond_stake_status {
198 |             StakeStatus::Empty => {
201 |                 SoloValidatorDepositSol::create_stake_account(
202 |                     &$ctx.accounts.rent,
203 |                     &$owner.to_account_info(),  // @audit: from
204 |                     &$ctx.accounts.bond.to_account_info(),
205 |                     &$ctx.accounts.stake_account,
206 |                     &$ctx.accounts.system_program.to_account_info(),
207 |                     $staked_amount,
208 |                     true,  // @audit: add_rent
214 |                 )?;
```

2. `stake_account` - `FullyActive` and `transient_stake_account` - `Empty`

```
/* bonds/src/macros.rs */
187 | macro_rules! stake_sol {
188 |     ($ctx:expr, $staked_amount:expr, $transient_stake_account:expr, $owner:expr) => {
197 |         match bond_stake_status {
244 |             StakeStatus::FullyActive => {
253 |                 match transient_stake_status {
254 |                     StakeStatus::Empty => {
261 |                         SoloValidatorDepositSol::create_stake_account(
262 |                             &$ctx.accounts.rent,
263 |                             &$owner.to_account_info(),  // @audit: add_rent
264 |                             &$ctx.accounts.bond.to_account_info(),
265 |                             &$transient_stake_account,
266 |                             &$ctx.accounts.system_program.to_account_info(),
267 |                             $staked_amount,
268 |                             true,  // @audit: add_rent
271 |                         )?;
```

Similarly, since the parameter `add_rent` is set to `true`, it will transfer `$staked_amount + rent_exempt` from the `bond` account to the `transient_stake_account` account.

3. `stake_account` - `FullyActive` and `transient_stake_account` - `FullyActive`

```
/* bonds/src/macros.rs */
187 | macro_rules! stake_sol {
188 |     ($ctx:expr, $staked_amount:expr, $transient_stake_account:expr, $owner:expr) => {
197 |         match bond_stake_status {
244 |             StakeStatus::FullyActive => {
253 |                 match transient_stake_status {
320 |                     StakeStatus::FullyActive => {
340 |                         // transfer stake amount + rent to the transient account
341 |                         let rent_exempt =
↪  $ctx.accounts.rent.minimum_balance(StakeStateV2::size_of());
342 |                         $staked_amount = $staked_amount + rent_exempt;
344 |                         if $owner.key() == $ctx.accounts.bond.key() {
345 |                             transfer_native($owner.to_account_info(),
↪  $transient_stake_account.to_account_info(), ..., $staked_amount, ...)?;
346 |                         } else {
353 |                         }
```

At line 345, `$staked_amount + rent_exempt` is transferred from the `bond` account to the `transient_stake_account` account.

4. `stake_account` - `FullyActive` and `transient_stake_account` - `Unitialized`

```
/* bonds/src/macros.rs */
187 | macro_rules! stake_sol {
188 |     ($ctx:expr, $staked_amount:expr, $transient_stake_account:expr, $owner:expr) => {
197 |         match bond_stake_status {
244 |             StakeStatus::FullyActive => {
253 |                 match transient_stake_status {
374 |                     StakeStatus::Unitialized => {
378 |                         let rent_exempt =
↪  $ctx.accounts.rent.minimum_balance(StakeStateV2::size_of());
379 |                         $staked_amount = $staked_amount + rent_exempt;
380 |                         transfer_lamports(
381 |                             $owner.to_account_info(),
382 |                             $transient_stake_account.to_account_info(),
383 |                             $staked_amount,
384 |                             None,
385 |                         )?;
```

Similarly, `$staked_amount + rent_exempt` is transferred from the `bond` account to the `transient_stake_account` account.

In summary, the current implementation considers scenarios 2 and 3. Scenarios 1 and 4 are not properly handled, where the `bond` account will be overdrawn.

Resolution

Fixed by commit `dbb76f827d9e16a9e4946f917860c17a9bac99b0`.

Appendix: Methodology and Scope of Work

Assisted by the Sec3 Scanner developed in-house, the manual audit 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 Pye in the sky labs dba Pye (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

The Sec3 audit team comprises a group of computer science professors, researchers, and industry veterans with extensive experience in smart contract security, program analysis, testing, and formal verification. We are also building automated 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.