Zellic

Prepared for
Alberto Cevallos
Pye in the Sky Labs Ltd.

Prepared by
Bryce Casaje
Junyi Wang
Zellic

January 23, 2025

Pye

Solana Application Security Assessment

Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Pye in the Sky Labs Ltd. from December 11 to January 3rd, 2025. During this engagement, Zellic reviewed Pye's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is the bond maturity mechanism working as intended?
- Can malicious users exploit vulnerabilities to access the yield or the staked SOL of other users?
- Are there any edge cases that could stop users from withdrawing after a bond reaches maturity?
- Does the yield reward calculation work as intended?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Pye programs, we discovered eight findings. Two critical issues were found. One was of high impact, one was of medium impact, and four were of low impact.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Pye in the Sky Labs Ltd. in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	2
🟧 High	1
🟨 Medium	1
🟩 Low	4
⬜ Informational	0

2. Introduction

2.1. About Pye

Pye in the Sky Labs Ltd. contributed the following description of Pye:

> Pye allows stakers to sell their staked Solana and future yield for USDC today.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the programs.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding’s impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and

Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped programs itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Pye Programs

Type	Rust
Platform	Solana
Target	pye-program-library
Repository	https://github.com/pyefi/pye-program-library ↗
Version	`3edf2d78e7251fb98486f4918df648b7e9b7d73c`
Programs	`programs/bonds/**.rs`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of three person-weeks. The assessment was conducted by two consultants over the course of four calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Bryce Casaje
Engineer
bryce@zellic.io ↗

Junyi Wang
Engineer
junyi@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

December 11, 2024	Start of primary review period
December 12, 2024	Kick-off call
January 3, 2025	End of primary review period
March 30, 2025	Start of secondary review period
March 30, 2025	Reviewed `1368c3de` ↗, `19aba18e` ↗ and `b05714fb` ↗
March 31, 2025	End of secondary review period

3. Detailed Findings

3.1. Uninitialized transient stake account can be stolen

Target	Pye		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	Medium	**Impact**	Critical

Description

When a solo validator bond has matured, a user can call the `handle_maturity` instruction to mark the bond as matured. The instruction also handles transient stake accounts, merging the transient stake account into the bond's stake account if it is fully active:

```
// ...

let transient_stake_status = get_stake_status(
    &ctx.accounts.transient_stake_account,
    ctx.accounts.clock.epoch,
    &ctx.accounts.stake_history,
)?;

// Handle existing transient account
match transient_stake_status {
    StakeStatus::FullyActive => {
        // merge it with the main stake account
        ctx.accounts.stake_merge(
            ctx.accounts.stake_account.to_account_info(),
            ctx.accounts.transient_stake_account.to_account_info(),
            ctx.accounts.bond.to_account_info(),
            &ctx.accounts.bond,
        )?;
    }
    StakeStatus::Empty => return Ok(()),
    _ => return Err(ErrorCode::UnsupportedStakeState.into()),
}

// ...
```

Merging the transient stake account sets its stake state to uninitialized and relocates all of its lamports to the bond's stake account.

Once the transaction is finished, since the transient stake account has zero lamports, the account will be closed as it does not have enough to cover rent.

However, an attacker can stop the transient stake account from being closed by sending enough lamports for rent in the same transaction. This leaves the transient stake account in the uninitialized state. An attacker could then reinitialize the transient stake account, stealing it by setting their own account as the authorized staker and withdrawer.

Impact

When redeeming yield tokens (YTs) for lamports or stake, the `SoloValidator-Bond::calculate_lamports_for_yt` method is used to calculate the yield for YTs, which includes the lamports of the transient stake account in its calculations.

If an attacker owned the transient stake account, they could deposit lamports into the transient stake account, redeem their YT for lamports or stake, and then immediately withdraw their deposit from the transient stake account.

Since the deposited lamports are taken into account in the yield calculations, they get a larger return on their YT than intended at no cost.

This could be used to steal funds from the bond's stake account or drain the global counter party (GCP).

Recommendations

Rework the bond implementation to allow for multiple transient stake accounts and not depend on their balance for yield calculations.

Remediation

This issue has been acknowledged by Pye in the Sky Labs Ltd., and fixes were implemented in the following commits:

- `19aba18e` ↗
- `36fddadb` ↗

3.2. Uninitialized stake account can be stolen

Target	Pye		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	Medium	**Impact**	Critical

Description

After a solo validator bond has matured, a user can call the `counter_party_update_liquid_reserve` instruction to unstake lamports from the bond's stake account. These lamports are moved into the bond's liquid reserve and can be later used by the global counter party to redeem their staking tokens.

The amount unstaked is based on the amount of staking tokens held by the global counter party. If unstaking this amount would leave the bond's stake account below the minimum delegation amount, then the entire stake account is unstaked:

```
let stake_account_lamports = ctx.accounts.stake_account.lamports();
let source_stake_after_split = stake_account_lamports
    .saturating_sub(lamports_to_unstake)
    .saturating_sub(stake_rent);

lamports_to_unstake = if source_stake_after_split < minimum_stake_account_amt
    {
    stake_account_lamports
} else {
    lamports_to_unstake
};
```

Then, the lamport amount is split from the bond's stake account into the transient stake account to be later withdrawn into the bond and used by the global counter party.

The issue arises when the entire balance of the bond's stake account is unstaked, as the stake account is deinitialized ↗ if it has zero balance after the split. Just as in Finding 3.1. ↗, this leaves a stake account uninitialized where it will be closed once the transaction is finished.

An attacker could stop the stake account from being closed by sending enough lamports for rent in the same transaction, which leaves it uninitialized, allowing them to then reinitialize the stake account to steal it.

Impact

The lamport amount of the bond's stake account is also included in the yield calculations, so an attacker could exploit this issue to drain the global counter party.

Recommendations

Rework the yield calculations to only take into account the balance of the bond's stake account if it is the withdrawer authority.

Remediation

This issue has been acknowledged by Pye in the Sky Labs Ltd., and a fix was implemented in commit `d093b983` ↗.

3.3. Fee can be front-run on deposit/withdraw

Target	Pye		
Category	Business Logic	**Severity**	High
Likelihood	Low	**Impact**	High

Description

When a user deposits into a bond, principal tokens (PTs) and YTs are minted. Some fraction of these tokens are distributed to the fee account, and this percentage is controlled by the `deposit_fee_bps` setting. This setting is controlled by the fee admin, and they can change it at any moment via the `update_fee_settings` instruction.

A malicious fee admin could wait for a user to deposit into a bond and then front-run their transaction to set the fees to the maximum. While the `update_fee_settings` instruction validates that the fee must be under `10_000` BPS, the malicious fee admin could set the fee to `9_999`, which would steal most of the tokens from the user.

A similar issue exists when redeeming PTs/YTs for SOL in a solo validator bond, as some amount of SOL is taken as a fee upon withdrawing based on the `counter_party_fee_bps` setting.

Impact

When depositing, a malicious fee admin could steal pool tokens from an unsuspecting user, receiving almost the full value of their deposit.

When redeeming PTs/YTs for SOL in a solo validator bond, a malicious fee admin could steal almost all of their yield.

Recommendations

For depositing, add instructions that allow users to specify slippage, where they can specify the minimum amount of tokens they expect to receive on deposit.

For withdrawing, require that fee changes only apply after some period of time, so that users cannot be front-run by any changes.

These recommendations are based on the Solana Program Library (SPL) stake-pool implementation.

Remediation

This issue has been acknowledged by Pye in the Sky Labs Ltd., and a fix was implemented in commit `26bd837c` ↗.

3.4. Linear-yield reduction for late staking

Target	Pye		
Category	Business Logic	Severity	Medium
Likelihood	Low	Impact	Medium

Description

For stake-pool bonds and solo validator bonds, the following code is used to calculate the amount of YTs that should be minted for a deposit.

```
// Calculations here round down intentionally to remove risk of rounding error
    attacks.
pub fn calc_yt_to_mint(&self, lst_state: &LstState, deposit_amount: u64, now:
    i64) -> u64 {
    let principal_lamports = lst_state.lamports_for_lst(deposit_amount);
    let numerator =
        U192::try_from(
            self.maturity_ts
                .checked_sub(now)
                .expect("underflow")
        ).expect("u64");
    let denominator = U192::try_from(
        self.maturity_ts
            .checked_sub(self.issuance_ts)
            .expect("underflow"),
    )
    .expect("u64");
    U192::from(principal_lamports)
        .checked_mul(numerator)
        .expect("overflow")
        .checked_mul(U192::from(PT_YT_DECIMAL_FACTOR))
        .expect("overflow")
        .checked_div(denominator)
        .expect("overflow")
        .checked_div(U192::from(LAMPORTS_PER_SOL))
        .expect("overflow")
        .as_u64()
}
```

Note that the formula for the amount of yield tokens that are minted is linear over time.

Impact

This allows an attacker to deposit a large amount of tokens just before the bond matures, capturing a large portion of the yield while only depositing into the bond for a very short time. Because of compound-interest effects, the earlier depositors should theoretically have a much larger share of the yields as they have contributed to a larger share of the yields. In practice, depositing just before a reward-distribution event has significant advantages. Potentially, an attacker could cycle through many bonds, depositing in all of them at the last second and being entitled to a significant portion of the yield despite not contributing to it.

Recommendations

Change the economic incentives so that earlier depositors are properly rewarded.

Remediation

This issue has been acknowledged by Pye in the Sky Labs Ltd..

3.5. Redeeming later is better for YT holders

Target	Pye		
Category	Business Logic	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

Upon the first redemption of a bond, the following function is called to populate the redemption cache.

```
pub fn set_redemption_cache_if_none(
    &mut self,
    lst_state: &LstState,
    stake_pool_tokens_supply: u64,
    pt_mint_supply: u64,
    yt_mint_supply: u64,
    lst_vault_balance: u64,
) {
    if !self.is_redemption_conversion_set() {
        let lst_for_all_pt = RedemptionCache::calc_lst_for_all_pt(
            pt_mint_supply,
            lst_vault_balance,
            lst_state.total_lamports(),
            stake_pool_tokens_supply,
        );
        self.redemption_cache = RedemptionCache {
            pt_supply_at_first_redemption: pt_mint_supply,
            bond_lst_bal_at_first_redemption: lst_vault_balance,
            lst_for_all_pt_at_redemption: lst_for_all_pt,
            yt_supply_at_first_redemption: yt_mint_supply,
        }
    }
}
```

The redemption cache notably fixes the value of a principal token to one SOL. Since the LST is still deposited in the pool and still earning yield, it is possible a yield-distribution event happens after the first redemption. In this case, the value of LST relative to SOL and the value of PTs would rise. If the conversion rate of PT to LST is fixed earlier, this rise in value would be captured in the PT. If fixing of the value of PT occurs later, this value would instead go to the YT.

Impact

It is in the best interests of the PT holders for the first redemption to happen as soon as possible after the bond matures, to fix the value of PT to LST in order to capture the rise in value of the LST. In practice, it is highly likely that the first redemption happens long before the next reward-distribution event, and therefore the impact of this is limited.

Recommendations

Consider adding a crank to set the redemption cache immediately after bond maturity, or fix the value through another mechanism.

Remediation

This issue has been acknowledged by Pye in the Sky Labs Ltd., and fixes were implemented in the following commits:

- `f921b4e8` ↗
- `95aed429` ↗
- `b05714fb` ↗

3.6. Redeeming can be blocked by empty GCP

Target	Pye		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

When a user redeems their PTs or YTs from a solo validator bond, they have the option to redeem directly as SOL or as stake.

If redeeming directly as SOL, they swap their tokens for SOL from the global counter party. If the global counter party has no funds, a user can permissionlessly redeem the global counter party's PTs/YTs to refill its funds to be later withdrawn.

However, the fee admin has unrestricted access to the `counter_party_withdraw_sol` instruction, which allows them to withdraw any amount of SOL from the global counter party's account.

This is normally fine, as user funds are backed by the solo validator bond itself, not the global counter party. However, an empty global counter party prevents users from redeeming their tokens as SOL.

This is an issue as the alternative is redeeming as stake, but this is only possible if a user has more than 1 SOL to redeem.

Impact

A fee admin could withdraw all the funds from the global counter party, preventing users from redeeming their solo validator bond tokens as SOL.

If those users have less than 1 SOL to redeem, their funds are stuck and they cannot withdraw.

Recommendations

Add a new redemption method for users with less than 1 SOL in a solo validator bond that does not rely on the global counter party.

Remediation

This issue has been acknowledged by Pye in the Sky Labs Ltd.. Pye in the Sky Labs Ltd. assumes `fee_admin` is operationally secure and acting in good faith and has opted not to make changes.

3.7. Fee-deposit accounts are missing associated-token-account constraint

Target	Pye		
Category	Coding Mistakes	Severity	Low
Likelihood	Low	Impact	Low

Description

When a user deposits into a bond, PTs and YTs are minted. Some of these tokens should be distributed to the fee wallet. Since each bond has its own mint of PTs and YTs, each bond needs two fee wallets specifically for those mints.

These fee wallets are passed in by the user when depositing:

```
#[account(
    mut,
    token::token_program = token_program,
    token::authority = protocol_fee_wallet,
)]
pub fee_wallet_pt: Box<InterfaceAccount<'info, TokenAccount>>,
#[account(
    mut,
    token::token_program = token_program,
    token::authority = protocol_fee_wallet,
)]
pub fee_wallet_yt: Box<InterfaceAccount<'info, TokenAccount>>
```

However, there is no constraint that the two accounts are associated token accounts for the `protocol_fee_wallet` authority. This means that a user could create new fee wallets for the `protocol_fee_wallet` account on each deposit.

Impact

The collected fees for one bond could be distributed over a large number of fee wallets, making it hard to collect and redeem fees.

Recommendations

Change the token constraint to require that they are associated token accounts.

Remediation

This issue has been acknowledged by Pye in the Sky Labs Ltd., and a fix was implemented in commit `ac441d01` ↗.

3.8. Missing referrer-account constraint

Target	Pye		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

Some stake pools may have referral fees set, which give a bonus to a specific referrer account upon deposit.

The test case for depositing into a stake pool on Pye sets this referrer account to the fee wallet, but the program has no constraints for this field.

Impact

A user could deposit into a stake pool without setting the referrer account to the fee wallet, causing the protocol to miss out on fees.

Recommendations

If the program is intended to collect referral fees, add a constraint to the account to ensure it is the bond's LST-fee wallet.

Remediation

This issue has been acknowledged by Pye in the Sky Labs Ltd., and a fix was implemented in commit `5bb0bc75` ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Untrusted bond creation

It is currently unclear whether untrusted users are supposed to be able to make bonds or whether this behavior is intended to be restricted. Malicious users could create bonds with options like issuance dates far in the past / maturity dates far in the future, bonds on stake pools with high fees, or with other issues.

An unsuspecting user depositing SOL into one of these malicious bonds could potentially lose funds.

However, if the Pye front-end application only displays bonds created by Pye admins, this should not be an issue.

4.2. Missing constraints

Some of the instructions have accounts that are missing some constraints. Besides Finding 3.1. ↗, most of the missing constraints do not lead to issues, as some part of the program will eventually error and cause a revert, but these should still be addressed as future code changes could potentially cause issues.

Here are two examples.

1. There are missing token mints on `fee_wallet_pt`, `fee_wallet_yt`, `owner_pt_account`, `owner_yt_account`, and `owner_lst_account`.
2. Marinade SOL stake pools can be passed into the wrong deposit instructions and vice versa.

4.3. Glob re-export clobbering

In the `mod.rs` file for various folders in the program, some values re-exported by the wildcard glob have duplicate names. This causes these values to be clobbered by other modules.

For example, a struct with the name `SoloValidatorRedeemPtForStakeCpiArgs` exists in both `instructions/counter_party/redeem_pt.rs` and `instructions/counter_party/redeem_yt.rs`. This could potentially cause confusion if code attempted to import the struct from the module `bonds::instructions::counter_party`. This same issue also happens for the `handler` export, as each instruction defines and exports a function with this name.

These values should each have a distinct name to avoid possible confusion when importing from these modules. Alternatively, the glob re-export could be removed if not used.

4.4. Bond timestamps

When creating a bond, the creator provides the issuance timestamp as an argument to the instruction. This issuance argument could be any timestamp before the maturity timestamp, and not the true issuance time of the bond.

A user could create a bond with the issuance timestamp far in the past or far in the future, which changes the total timespan of the bond. This scales the amount of yield tokens minted for each deposit. However, this issue does not appear to be exploitable since this affects all users proportionally. But, this could still lead to potential confusion.

The issuance timestamp could instead be set from the Unix timestamp provided by the Clock sysvar rather than provided by the user, which would make it the true issuance time of the bond.

4.5. Potential deadlocks when slashed

In many parts of the code, it is assumed that LSTs would never decrease in value.

These assumptions, when violated, have effects ranging from deadlocks to incorrect economic incentives. LSTs, however, can decrease in value if the backing validator(s) are slashed.

While there is no automatic mechanism to slash validators on Solana, attackers causing the network to halt can be manually slashed upon network restart. While the risk of this happening is low, the Pye team should still consider updating the protocol to correctly handle slashing in order to future-proof the protocol.

This issue has been acknowledged by Pye in the Sky Labs Ltd., and a fix was implemented in commit `59c31eef` ↗.

4.6. Test suite

The test suite provided covered most of the instructions implemented by the program.

However, some improvements could still be made, especially in regards to edge cases and negative scenarios, as the test suite mostly focuses on verifying that each instruction works as expected. In addition, most of the test suite runs with the program signing each instruction and not from unprivileged accounts, which does not simulate real-world usage.

5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function’s externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. Solo validator bonds

Description

A user can deposit SOL or staked SOL into a solo validator bond, and the Pye program delegates these funds to a validator using the SPL stake program. The Pye program has one main stake account and one transient stake account for each solo validator bond, both of which may be active with staked funds.

If a user deposits staked SOL, their stake is directly merged into the bond's stake account.

Since unstaked SOL cannot be directly deposited into a stake account, if a user deposits unstaked SOL, a more complicated process occurs:

1. If the bond's stake account does not exist, it is created with the user's deposit.
2. If the bond's stake account does exist but is activating, an uninitialized stake account owned by the user is merged into the stake account instead.
3. If the bond's stake account is fully active, the transient stake account is used.

When using the transient stake account,

1. If the transient stake account is empty, it is created with the user's deposit.
2. If the transient stake account does exist but is uninitialized, it is initialized with the user's deposit.
3. If the transient stake account does exist but is activating, an uninitialized stake account owned by the user is merged into the transient account instead.
4. If the transient stake account is fully activated, it is merged into the main stake account, and then the transient stake account is reinitialized with the user's deposit.

PTs and YTs are then minted and distributed to the user for their deposit, to be later burned when the bond has matured to be redeemed for SOL or staked SOL.

Invariants

- **Funds cannot be redeemed until the bond has matured.** This is enforced by using the `maturity_ts` timestamp, set when the bond is created. The `handle_maturity` instruction that marks a bond as mature can only be called when the current timestamp is greater than `maturity_ts`.

- **Only authorized entities should be able to redeem from the bond.** When depositing, PTs and YTs are minted and distributed to the user and are required to redeem from the bond.
- **Users cannot withdraw more than they are entitled to.** Redemption amount is directly related to the amount of PTs and YTs redeemed. Users entitled to more rewards are given more tokens. Once the tokens are redeemed, they are burned so they cannot be used twice.
- **Users depositing early should have higher yields than users depositing closer to the bond's maturity.** The amount of YTs minted for a deposit linearly decays with the time until the bond's maturity.
- **The amount per YT should be correlated with the rewards from staking.** Staking rewards are automatically added to the stake account's staked balance, which is used in the yield calculations.
- **Total rewards from the bond must be less than or equal to the lamport balance of the bond.** When redeeming, the lamports per YT is calculated as a proportion of the total lamport amount of the bond (including the transient stake account).

Test coverage

Cases covered

- Initializing of solo validator bonds
- Depositing SOL / staked SOL into a solo validator bond
- Calling the `handle_maturity` instruction after the bond has matured
- Redeeming PT/YT for SOL / staked SOL

Cases not covered

- Calling the `handle_maturity` instruction before the bond has matured
- Attempting to redeem more PTs/YTs than the user has access to
- Checking whether different users can deposit / redeem from a solo validator bond
- Redeeming multiple times / at different times

Attack surface

- **Solo validator bond.** A `SoloValidatorBond` account can only be created via the `initialize_solo_validator_bond` instruction, which validates the bond's settings (for example, maturity timestamp, token mint, and so on).
- **PTs/YTs.** These tokens can only be minted by the program, and they have to match the bond's token mint.

- **Stake accounts.** The main stake account and transient stake account must match the accounts stored in the `SoloValidatorBond` account, which are PDAs derived from the bond's address.
- **Fee wallet / global counter party.** These are checked by matching with global settings.
- **Fee-wallet token accounts.** Fee-wallet accounts must have the fee wallet as their authority. The token mint is validated when fee tokens are distributed to the token account.

5.2. Stake pools

Description

The stake-pool bond system turns SOL staked into a stake pool into a bond. The deposited funds are segregated into principal and yield. The exact amount each PT and YT will redeem for is determined at the first redemption of a bond. The PTs and YTs can be used to redeem for LSTs, which in turn can be redeemed for SOL.

Invariants

- The amount of tokens minted should be correctly related to the amount deposited.
- The amount of PTs in circulation should not exceed the value in SOL deposited.

Test coverage

Cases covered

- Initializing a bond from a stake pool
- Depositing SOL into a bond generated from a stake pool
- Depositing LST into a bond generated from a stake pool
- Depositing mSOL into an mSOL bond

Cases not covered

- Redeeming PTs from a stake-pool bond
- Redeeming YTs from a stake-pool bond
- Redeeming PTs from an mSOL bond
- Redeeming YTs from an mSOL bond
- Changing in the value of LSTs before the first redemption and after bond maturity
- Trying to redeem before bond maturity
- Depositing multiple times from different accounts
- Depositing after bond maturity

Attack surface

- **Depositing a custom LST coin type.** A transfer will be attempted to an account that is enforced to be the correct coin type, which will fail in the case the incorrect type of coin account is given.
- **Passing in a fake fee wallet to capture the fee.** The fee-wallet authority is checked, which in turn is part of the global settings, which is a PDA with a specific seed.
- **Using the PT/YT token mint for another, more valuable bond, causing the wrong PT-s/YTs to be minted.** The token mint is checked against the bond's data.
- **Redeeming early.** There is a check for the current time on Solana.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Solana Mainnet.

During our assessment on the scoped Pye programs, we discovered eight findings. Two critical issues were found. One was of high impact, one was of medium impact, and four were of low impact.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.