Zellic

Prepared for

Solana Foundation

Prepared by
Jasraj Bedi
Filippo Cremonese
Zellic

January 2, 2024

Single Pool
Program Security Assessment

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue ↗, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io ↗ or follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io ↗.

1. Executive Summary

Zellic conducted a security assessment for Solana Foundation from December 18th to December 22nd, 2023. During this engagement, Zellic reviewed Single Pool's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1. 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:

- Can an attacker take control of user-staked SOL?
- Could an attacker trigger a lockup of user stake?
- Are users always able to redeem their staked SOL?

1.2. 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.

1.3. Results

During our assessment on the scoped Single Pool programs, there were no security vulnerabilities discovered.

Zellic recorded its notes and observations from the assessment for Solana Foundation's benefit in the Discussion section (3. ↗).

Breakdown of Finding Impacts

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

2. Introduction

2.1. About Single Pool

Single Pool is a stripped-down stake pool program for the single-validator case that enables liquid staking with no fees and 100% capital efficiency.

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 (3. ↗) 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:

Single Pool Programs

Repository	https://github.com/solana-labs/solana-program-library ↗
Version	solana-program-library: `ef44df985e76a697ee9a8aabb3a223610e4cf1dc`
Programs	• single-pool/program/src/entrypoint.rs • single-pool/program/src/error.rs • single-pool/program/src/instruction.rs • single-pool/program/src/lib.rs • single-pool/program/src/processor.rs • single-pool/program/src/state.rs
Type	Rust
Platform	Solana

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of two person-weeks. The assessment was conducted over the course of one calendar week.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Jasraj Bedi
Engineer
jazzy@zellic.io

Filippo Cremonese
Engineer
fcremo@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

December 18, 2023	Start of primary review period
December 22, 2023	End of primary review period

3. 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.

3.1. Authorization pattern

We note that the program adopts some design patterns that could potentially be misused. For instance, deposits assume the user grants authorization over a stake account to a single authority PDA that is only tied to the pool. Similarly, withdrawals assume the user grants authorization over their pool token account to an authority PDA that is also only tied to the pool address.

This design assumes the user grants authorization to manage their assets to the single pool authority PDAs in the same transaction that performs the desired single pool action; if the user grants authorization to the pool PDA without performing the intended action in the same transaction, an attacker could invoke a pool action on their behalf, stealing their assets.

The current design can be used safely, and the repository contains code that helps developers to create a list of instructions that use the program as intended. However, we note that other designs could be safer and harder to misuse. For instance, tying the authority PDAs to the token account or stake account they control would improve the safety of the design.

4. Threat Model

This provides a full threat model description for the instructions supported by the program. As time permitted, we analyzed each instruction and created a written threat model that documents which input accounts and data can be supplied to each instruction handler.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

4.1. Instructions

Instruction: `InitializePool`

This instruction can be used to initialize a new pool and supporting accounts.

Specifically, the following accounts are initialized:

- An account representing the pool
- A stake account for the pool, owned by the stake program
- A mint account, used to mint tokens representing a portion of the pool

Input accounts

- `vote_account_info`: Vote account associated with the newly created pool.
  - **W/S requisites**: None.
  - **Owner**: Must be owned by native vote program.
  - **Rent**: Not a responsibility of the program under review.
  - **Initialized**: Must be initialized.
  - **Discriminant**: Checked.
  - **PDA**: No.
- `pool_info`: New pool account.
  - **W/S requisites**: Writable.
  - **Owner**: Single pool program (newly created).
  - **Rent**: Must be exempt.
  - **Initialized**: No (initialized by this instruction).
  - **Discriminant**: Checked (`account_type` field).
  - **PDA**: Yes, controlled by the single pool program — using seeds `“pool”`, `vote_account_address`.
- `pool_stake_info`: Stake account for the new pool.
  - **W/S requisites**: Writable.
  - **Owner**: Stake program (newly created).
  - **Rent**: Must be exempt.
  - **Initialized**: No (initialized by this instruction).
  - **Discriminant**: Checked.
  - **PDA**: Yes, controlled by the single pool program — using seeds `“stake”`,

`pool_address`.

- `pool_mint_info`: New mint account for the pool.
    - **W/S requisites**: Writable.
    - **Owner**: Token program (newly created).
    - **Rent**: Not a responsibility of the program under review, must be rent free due to checks done by spl-token.
    - **Initialized**: No (initialized by this instruction).
    - **Discriminant**: Not a responsibility of the program under review.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“mint”`, `pool_address`.
- `pool_stake_authority_info`: Authority controlling the pool stake.
    - **W/S requisites**: None.
    - **Owner**: N/A.
    - **Rent**: N/A.
    - **Initialized**: N/A.
    - **Discriminant**: N/A.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“stake authority”, pool_address`.
- `pool_mint_authority_info`: Authority controlling the pool mint.
    - **W/S requisites**: None.
    - **Owner**: N/A.
    - **Rent**: N/A.
    - **Initialized**: N/A.
    - **Discriminant**: N/A.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“mint authority”, pool_address`.
- `rent_info`: Rent sysvar account.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No, but the address must match the known rent sysvar.
- `clock_info`: Clock sysvar account — not used, only passed along to the stake program.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No — the address is checked to ensure it matches the actual clock sysvar.

- `stake_history_info`: Stake history sysvar account — not used, only passed along to the stake program.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No.
- `stake_config_info`: Stake config sysvar account — not used, only passed along to the stake program.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No.
- `system_program_info`: System program account.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No — the address is checked to ensure it matches the actual system program.
- `token_program_info`: Token program account.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No — the address is checked to ensure it matches the actual token program.
- `stake_program_info`: Stake program account.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No — the address is checked to ensure it matches the actual stake program.

Instruction: `ReactivatePoolStake`

This instruction can be used to restake the pool stake account. This instruction is needed if the stake is deactivated for any reason. Currently, a stake can be forcefully deactivated by inactivity (via the stake program `DeactivateDelinquent` instruction) or due to a cluster restart.

The instruction is unpermissioned; this is safe, since the stake is committed to the vote account associated with the pool. Stake accounts that have been deactivated due to natural expiry are forbidden from being restaked.

Input accounts

- `vote_account_info`: Vote account associated with the pool.
  - **W/S requisites**: None.
  - **Owner**: Must be owned by native vote program.
  - **Rent**: Not a responsibility of the program under review.
  - **Initialized**: Must be initialized.
  - **Discriminant**: Checked.
  - **PDA**: No.
- `pool_info`: Pool account.
  - **W/S requisites**: None.
  - **Owner**: Single pool program.
  - **Rent**: N/A.
  - **Initialized**: Yes.
  - **Discriminant**: Checked (`account_type` field).
  - **PDA**: Yes, controlled by the single pool program — using seeds `“pool”`, `vote_account_address`.
- `pool_stake_info`: Stake account associated with the pool.
  - **W/S requisites**: Writable.
  - **Owner**: Stake program (indirectly checked, since it is written to by the stake program).
  - **Rent**: N/A.
  - **Initialized**: Yes.
  - **Discriminant**: Checked.
  - **PDA**: Yes, controlled by the single pool program — using seeds `“stake”`, `pool_address`.
- `pool_stake_authority_info`: Authority controlling the pool stake.
  - **W/S requisites**: None.
  - **Owner**: N/A.
  - **Rent**: N/A.
  - **Initialized**: N/A.
  - **Discriminant**: N/A.
  - **PDA**: Yes, controlled by the single pool program — using seeds `“stake au-`

thority", pool_address.

- `clock_info`: Clock sysvar account.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No — the address is checked to ensure it matches the actual clock sysvar.
- `stake_history_info`: Stake history sysvar account — not used, only passed along to the stake program.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No.
- `stake_config_info`: Stake config sysvar account — not used, only passed along to the stake program.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No.
- `stake_program_info`: Stake program account.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No — the address is checked to ensure it matches the actual stake program.

Instruction: `DepositStake`

This instruction can be used to deposit some stake into the pool. The program assumes the user passes in a stake account that has been authorized to the authority controlled by the single pool program. The authorization instruction must happen in the same transaction in order for this mechanism to be safe, and the repository provides helper functions that compose a bundle of instructions that use the program safely.

The instruction merges the user stake account with the pool stake account. The user is minted an amount of pool tokens representing their fraction of the staked assets contained in the pool.

Input accounts

- `pool_info`: Pool account.
    - **W/S requisites**: None.
    - **Owner**: Single pool program.
    - **Rent**: N/A.
    - **Initialized**: Yes.
    - **Discriminant**: Checked (`account_type` field).
    - **PDA**: Yes, controlled by the single pool program; using seeds `“pool”, vote_account_address`; and checked by reading internal state.
- `pool_stake_info`: Stake account associated with the pool.
    - **W/S requisites**: Writable.
    - **Owner**: Stake program (indirectly checked, since it is written to by the stake program).
    - **Rent**: N/A.
    - **Initialized**: Yes.
    - **Discriminant**: Checked.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“stake”, pool_address`.
- `pool_mint_info`: Mint account associated with the pool.
    - **W/S requisites**: Writable.
    - **Owner**: Token program (indirectly checked, since it is written to by the token program).
    - **Rent**: N/A.
    - **Initialized**: Yes.
    - **Discriminant**: Not a responsibility of the program under review.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“mint”, pool_address`.
- `pool_stake_authority_info`: Authority controlling the pool stake.
    - **W/S requisites**: None.
    - **Owner**: N/A.
    - **Rent**: N/A.
    - **Initialized**: N/A.
    - **Discriminant**: N/A.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“stake authority”, pool_address`.
- `pool_mint_authority_info`: Authority controlling the pool mint.
    - **W/S requisites**: None.
    - **Owner**: N/A.

- **Rent**: N/A.
- **Initialized**: N/A.
- **Discriminant**: N/A.
- **PDA**: Yes, controlled by the single pool program — using seeds `“mint authority”, pool_address`.

- `user_stake_info`: User stake account that will be merged into the pool.
  - **W/S requisites**: Writable.
  - **Owner**: Stake program (indirectly checked, since it is written to by the stake program).
  - **Rent**: N/A.
  - **Initialized**: Yes.
  - **Discriminant**: Not a responsibility of the program under review.
  - **PDA**: No. Address must be different than `pool_stake_info`.
- `user_token_account_info`: User token account that will receive the minted pool tokens.
  - **W/S requisites**: Writable.
  - **Owner**: Token program (indirectly checked, since it is written to by the token program).
  - **Rent**: N/A.
  - **Initialized**: Yes.
  - **Discriminant**: Not a responsibility of the program under review.
  - **PDA**: No.
- `user_lamport_account_info`: Account that will receive any excess lamports.
  - **W/S requisites**: Writable.
  - **Owner**: Not checked (not an issue).
  - **Rent**: N/A.
  - **Initialized**: N/A.
  - **Discriminant**: N/A.
  - **PDA**: No.
- `clock_info`: Clock sysvar account.
  - **W/S requisites**: None.
  - **Owner**: N/A (sysvar).
  - **Rent**: N/A (sysvar).
  - **Initialized**: N/A (sysvar).
  - **Discriminant**: N/A (sysvar).
  - **PDA**: No — the address is checked to ensure it matches the actual clock sysvar.
- `stake_history_info`: Stake history sysvar account — not used, only passed along to the stake program.
  - **W/S requisites**: None.
  - **Owner**: N/A (sysvar).
  - **Rent**: N/A (sysvar).

    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No.
- `token_program_info`: Token program account.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No — the address is checked to ensure it matches the actual token program.
- `stake_program_info`: Stake program account.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No — the address is checked to ensure it matches the actual stake program.

Instruction: `WithdrawStake`

This instruction can be used to redeem pool tokens in exchange for their corresponding shares of staked SOL. The amount of staked assets to be withdrawn is proportional to the amount of pool tokens being redeemed with respect to the total minted amount.

The program burns the given amount of pool tokens and splits off the computed amount of SOL into a new stake account; finally, it transfers control of the new stake account to an address specified by the user.

Inputs

- `user_stake_authority`: User authority for the new stake account.
    - **Validation**: None.
- `token_amount`: Amount of tokens to redeem for stake.
    - **Validation**: None explicitly — must be less than the amount owned by the user token account.

Input accounts

- `pool_info`: Pool account.
    - **W/S requisites**: None.

    - **Owner**: Single pool program.
    - **Rent**: N/A.
    - **Initialized**: Yes.
    - **Discriminant**: Checked (`account_type` field).
    - **PDA**: Yes, controlled by the single pool program; using seeds `“pool”`, `vote_account_address`; and checked by reading internal state.
- `pool_stake_info`: Stake account associated with the pool.
    - **W/S requisites**: Writable.
    - **Owner**: Stake program (indirectly checked, since it is written to by the stake program).
    - **Rent**: N/A.
    - **Initialized**: Yes.
    - **Discriminant**: Checked.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“stake”`, `pool_address`.
- `pool_mint_info`: Mint account associated with the pool.
    - **W/S requisites**: Writable.
    - **Owner**: Token program (indirectly checked, since it is written to by the token program).
    - **Rent**: N/A.
    - **Initialized**: Yes.
    - **Discriminant**: Not a responsibility of the program under review.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“mint”`, `pool_address`.
- `pool_stake_authority_info`: Authority controlling the pool stake.
    - **W/S requisites**: None.
    - **Owner**: N/A.
    - **Rent**: N/A.
    - **Initialized**: N/A.
    - **Discriminant**: N/A.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“stake authority”, pool_address`.
- `pool_mint_authority_info`: Authority controlling the pool mint.
    - **W/S requisites**: None.
    - **Owner**: N/A.
    - **Rent**: N/A.
    - **Initialized**: N/A.
    - **Discriminant**: N/A.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“mint authority”, pool_address`.
- `user_stake_info`: Empty account that will be initialized as a stake account. It will receive the stake split off of the pool stake account.

    - **W/S requisites**: Writeable.
    - **Owner**: Stake program (indirectly checked, since it is written to by the stake program).
    - **Rent**: Not a responsibility of the program under review.
    - **Initialized**: No.
    - **Discriminant**: Not a responsibility of the program under review.
    - **PDA**: No. Address must be different than `pool_stake_info`.
- `user_token_account_info`: User token account from where pool tokens are taken and burned from.
    - **W/S requisites**: Writable.
    - **Owner**: Token program (indirectly checked, since it is written to by the token program).
    - **Rent**: N/A.
    - **Initialized**: Yes.
    - **Discriminant**: Not a responsibility of the program under review.
    - **PDA**: No.
- `clock_info`: Clock sysvar account — not used, only passed along to the stake program.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No.
- `token_program_info`: Token program account.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No — the address is checked to ensure it matches the actual token program.
- `stake_program_info`: Stake program account.
    - **W/S requisites**: None.
    - **Owner**: N/A (sysvar).
    - **Rent**: N/A (sysvar).
    - **Initialized**: N/A (sysvar).
    - **Discriminant**: N/A (sysvar).
    - **PDA**: No — the address is checked to ensure it matches the actual stake program.

Instruction: `CreateTokenMetadata`

This instruction is used to initialize the token metadata account for the pool mint.

Input accounts

- `pool_info`: Stake pool account.
  - **W/S requisites**: None.
  - **Owner**: Single pool program (newly created).
  - **Rent**: N/A.
  - **Initialized**: Yes.
  - **Discriminant**: Checked (`account_type` field).
  - **PDA**: Yes, controlled by the single pool program — using seeds `“pool”, vote_account_address`.
- `pool_mint_info`: Pool mint account.
  - **W/S requisites**: None.
  - **Owner**: Token program (newly created).
  - **Rent**: Not a responsibility of the program under review — must be rent free due to checks done by spl-token.
  - **Initialized**: Yes.
  - **Discriminant**: Not a responsibility of the program under review.
  - **PDA**: Yes, controlled by the single pool program — using seeds `“mint”, pool_address`.
- `pool_mint_authority_info`: Authority controlling the pool mint.
  - **W/S requisites**: None.
  - **Owner**: N/A.
  - **Rent**: N/A.
  - **Initialized**: N/A.
  - **Discriminant**: N/A.
  - **PDA**: Yes, controlled by the single pool program — using seeds `“mint authority”, pool_address`.
- `pool_mpl_authority_info`: The update authority controlling the metadata account.
  - **W/S requisites**: None.
  - **Owner**: N/A.
  - **Rent**: N/A.
  - **Initialized**: N/A.
  - **Discriminant**: N/A.
  - **PDA**: Yes, controlled by the single pool program — using seeds `“mpl_authority”, pool_address`.
- `payer_info`: The account set to pay for creating the new metadata account.
  - **W/S requisites**: Signed and writable.
  - **Owner**: N/A.

- **Rent**: N/A.
- **Initialized**: N/A.
- **Discriminant**: N/A.
- **PDA**: No.

- `metadata_info`: The MPL metadata account address.
  - **W/S requisites**: Writable.
  - **Owner**: MPL Metadata program.
  - **Rent**: N/A.
  - **Initialized**: No.
  - **Discriminant**: N/A.
  - **PDA**: Yes — using seeds
    - `“metadata”`
    - `metaqbxxUerdq28cj1RbAWkYQm3ybzjb6a8bt518x1s`
    - `mint`
    - `metaqbxxUerdq28cj1RbAWkYQm3ybzjb6a8bt518x1s`
- `mpl_token_metadata_program_info`: MPL Token Program account.
  - **W/S requisites**: None.
  - **Owner**: N/A (sysvar).
  - **Rent**: N/A (sysvar).
  - **Initialized**: N/A (sysvar).
  - **Discriminant**: N/A (sysvar).
  - **PDA**: No, but the address must match the MPL token metadata program.
- `system_program_info`: System program account.
  - **W/S requisites**: None.
  - **Owner**: N/A (sysvar).
  - **Rent**: N/A (sysvar).
  - **Initialized**: N/A (sysvar).
  - **Discriminant**: N/A (sysvar).
  - **PDA**: No — the address is checked to ensure it matches the actual system program.

Instruction: `UpdateTokenMetadata`

This instruction is used to update the pool’s token metadata account.

A key differentiation between the `UpdateTokenMetadata` instruction and `CreateTokenMetadata` instruction is that only the `UpdateTokenMetadata` instruction can specify name, symbol, and URI.

Input accounts

- `vote_account_info`: Vote account associated with the pool.
  - **W/S requisites**: None.

    - **Owner**: Must be owned by native vote program.
    - **Rent**: Not a responsibility of the program under review.
    - **Initialized**: Must be initialized.
    - **Discriminant**: Checked.
    - **PDA**: No.
- `pool_info`: The stake pool.
    - **W/S requisites**: None.
    - **Owner**: Single pool program (newly created).
    - **Rent**: Must be exempt.
    - **Initialized**: Yes.
    - **Discriminant**: Checked (`account_type` field).
    - **PDA**: Yes, controlled by the single pool program — using seeds `“pool”, vote_account_address`.
- `pool_mpl_authority_info`: The update authority controlling the metadata account.
    - **W/S requisites**: None.
    - **Owner**: N/A.
    - **Rent**: N/A.
    - **Initialized**: N/A.
    - **Discriminant**: N/A.
    - **PDA**: Yes, controlled by the single pool program — using seeds `“mpl_authority”, pool_address`.
- `authorized_withdrawer_info`: The authorized withdrawer account as set on the vote account.
    - **W/S requisites**: Signer.
    - **Owner**: N/A.
    - **Rent**: N/A.
    - **Initialized**: N/A.
    - **Discriminant**: N/A.
    - **PDA**: N/A.
- `metadata_info`: The MPL metadata account address.
    - **W/S requisites**: Writable.
    - **Owner**: MPL Metadata program.
    - **Rent**: N/A.
    - **Initialized**: No.
    - **Discriminant**: N/A.
    - **PDA**: Yes — using seeds
        - `“metadata”`
        - `metaqbxxUerdq28cj1RbAWkYQm3ybzjb6a8bt518x1s`
        - `mint`
        - `metaqbxxUerdq28cj1RbAWkYQm3ybzjb6a8bt518x1s`
- `mpl_token_metadata_program_info`: MPL Token Program account.
    - **W/S requisites**: None.

- **Owner**: N/A (sysvar).
- **Rent**: N/A (sysvar).
- **Initialized**: N/A (sysvar).
- **Discriminant**: N/A (sysvar).
- **PDA**: No, but the address must match the MPL token metadata program.

5. Assessment Results

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

During our assessment on the scoped Single Pool programs, there were no security vulnerabilities discovered.

5.1. 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.