Zellic

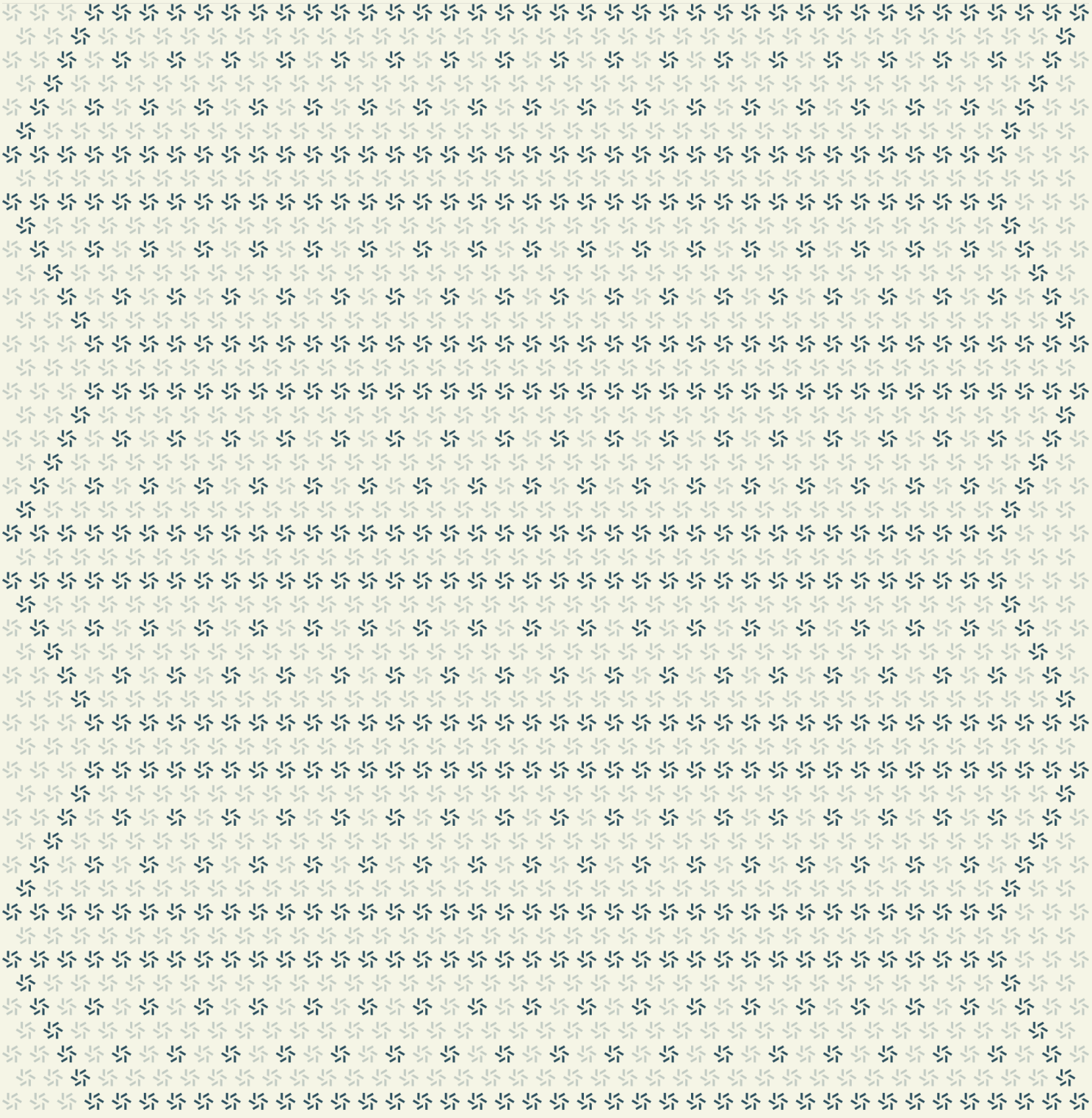
Prepared for
YO Foundation

Prepared by
Nathanial Lattimer
Maik Robert
Zellic

February 24, 2026

YO

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 YO Foundation from February 17th to February 23rd, 2026. During this engagement, Zellic reviewed YO'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:

- Can the instruction whitelist be bypassed, allowing unintended CPI calls?
- Are CPI calls exploitable in any edge cases (e.g., if calling with the wrong target program)?
- Are there certain instructions (e.g., token approval) that can cause loss of funds via `manage()`?
- Are there any scenarios involving stale oracle prices that let users mint/redeem shares at advantageous prices?
- Is the `max_pct_change` and anchor price model resistant to fluctuations in prices and potential market volatility?

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.

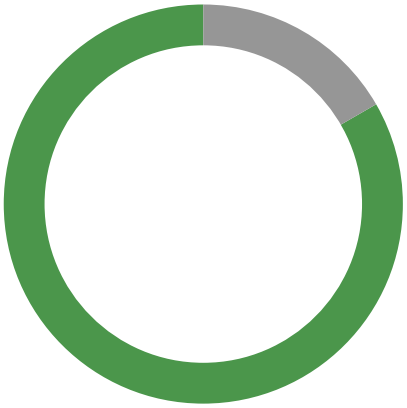
1.4. Results

During our assessment on the scoped YO programs, we discovered six findings. No critical issues were found. Five findings were of low impact and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of YO Foundation in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	5
Informational	1



2. Introduction

2.1. About YO

YO Foundation contributed the following description of YO:

> YO (short for Yield Optimizer) is a cross-chain yield optimization protocol launched in early 2025. It is live on mainnet across Base, Ethereum, Unichain, Arbitrum, Plasma, HyperEVM, and Monad and currently custodies $70M in TVL (with a peak of $90M).
>
> The protocol implements a ERC4626-style vault to accept user deposits, and invests them across underlying yield protocols to return the highest risk-adjusted yield. Funds can be bridged between YO vaults on different chains to be deployed to cross-chain opportunities.

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:

YO Programs

Type	Rust
Platform	SVM
Target	solana
Repository	https://github.com/yoprotocol/solana ↗
Version	`faffc9b501220681ca3c8a3a924f56c7217bf25e`

Programs	`programs/yo-oracle/*.rs`
	`programs/yo-vault/*.rs`
	`crates/yo-lib/src/*.rs`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.2 person-weeks. The assessment was conducted by two consultants over the course of five calendar days.

Contact Information

The following project manager was associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Nathanial Lattimer
Engineer
d0nut@zellic.io ↗

Maik Robert
Engineer
maik@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

February 17, 2026	Start of primary review period
February 18, 2026	Kick-off call
February 23, 2026	End of primary review period

3. Detailed Findings

3.1. Denial of service due to partial fulfillment

Target	crates/yo-vault/src/instructions/fulfill_redeem.rs		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The `fulfill_redeem` instruction is used when a previous attempt at redemption was unable to be fulfilled at that time, but assets then become available to service the redemption request. The operator invokes this instruction, specifying the amount of assets and shares to transfer and burn, respectively. It is required that the operator specify some figure greater than zero for both shares and assets; otherwise, the instruction will fail. This means that each attempt to fulfill a redemption request *must* specify both a nonzero shares and asset figure; otherwise, the pending redemption will be unserviceable.

We observed that there is no check that ensures that if either figure hits `0` that both `pending_shares` and `pending_assets` are fully depleted. Because of this missing check, it is possible for a pending redemption request to no longer be serviceable as the `shares > 0` and/or `assets > 0` checks will fail. Similarly, they cannot be canceled by the operator as well due to similar checks. The only path to fix the pending redemptions is either the user updating the request with more shares/assets to redeem or an upgrade to relax the constraints in these functions.

Impact

This can result in a situation where the operator depletes all of the shares but not all of the assets. This will permanently affix the deficit in redeemed assets against the vault total, meaning that other users will not be able to instantly redeem their assets unless the vault maintains *at least* the total remaining pending assets.

Recommendations

We recommend adding a check that ensures that if either `pending_shares` or `pending_assets` is about to be set to `0`, the other figure is as well to ensure the redemption request was fully serviced.

Remediation

This issue has been acknowledged by YO Foundation, and a fix was implemented in commit `23ed72f6` ↗.

3.2. Managed instructions provide excessive privileges

Target	crates/yo-vault/src/instructions/manage/*.rs		
Category	Protocol Risks	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The `manage_*` instructions are used to interact with other programs on chain as the operator and vault authority accounts. This allows the vault program to manage the assets held in the vault as needed for typical operation.

While some safety features were implemented by whitelisting specific programs and instruction pairs, there is still a risk that the program invoked has a vulnerability or a dangerous feature or an update is pushed that allows escalating the intended access in the CPI to one that was unintended. Because the programs are invoked using `invoke_signed` with the vault authority's seeds, this allows downstream programs to take actions on behalf of the vault authority. While recursive calls back to the vault program are not allowed due to Solana's design, the vault authority has minting authority over the shares mint as well as access to escrowed shares in a token account that are likely not meant to be involved in the `manage_*` operations.

Impact

Due to a vulnerability, dangerous feature, or malicious update, a downstream program could mint new shares or access escrowed shares via the `vault_authority`'s access.

Recommendations

We recommend migrating ownership of the assets to a distinct authority so that the `manage_*` instructions' use of `invoke_signed` with the vault authority's seeds does not conflate access to the managed assets with the right to mint shares or the escrowed shares.

Remediation

This issue has been acknowledged by YO Foundation, and a fix was implemented in commit `7b82fcc5` ↗.

3.3. Redemption fees rounded down

Target	yo-vault		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

Fees are collected both in deposits as well as redemptions. In the deposit flow, fees are calculated, rounding up, and then subtracted from the assets to determine what assets remain and are to be credited to the user in the form of shares. Rounding up ensures malicious users cannot specify amounts that would minimize or entirely eliminate fees by allowing fee calculations to round *down* to zero.

We noticed in both redemption flows, in the `request_redeem` instruction and `fulfill_redeem` instruction, that fees were rounded down. While fees would be collected correctly on the deposit step, which is required before redemption can occur, by rounding fees down, it is possible to reduce fees to `0` by redeeming small enough amounts to cause rounding to consistently err on the side of the user.

Impact

By rounding fees down, the protocol may experience a loss in revenue due to users redeeming small enough amounts of shares to cause fees to round down to `0`.

Recommendations

We recommend correcting the fee calculations everywhere to round up.

Remediation

This issue has been acknowledged by YO Foundation, and a fix was implemented in commit `e34136f4` ↗.

3.4. Double oracle update can lead to larger-than-anticipated price move

Target	yo-oracle/src/instructions/update_vault_price.rs		
Category	Business Logic	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The oracle stores two different prices, a `price` and an `anchor_price`. To prevent price updates that exceed a certain percent defined in `vault_data.max_pct_change`, the oracle uses a slot window — a certain amount of slots during which the increase in `price` should not deviate by more than `vault_data.max_pct_change` from the `anchor_price` in either direction.

After the slot window has passed, the `anchor_price` will be updated to reflect the current `price` and a new window is started. This design allows for the price to *exceed* the `vault_data.max_pct_change` if there are two oracle updates back-to-back that fall right at the end of a slot window.

If we assume that the current slot window is over and that the next oracle update would update the `anchor_price` and start a new window, we could see a `price` change of up to `vault_data.max_pct_change` in that transaction. If there is another oracle update in the same slot, that also increases the price by `vault_data.max_pct_change`. We have now increased the actual `price` by `2 * vault_data.max_pct_change` in the same slot, which is perfectly legal as we just started a new window in the previous transaction.

Impact

The design of the oracle-price update instruction allows increasing the `price` by up to `2 * vault_data.max_pct_change` if there are two consecutive oracle updates where the first update initiates a new slot window. This large price increase or decrease might be unexpected in the same slot.

Recommendations

We recommend changing the design of the oracle to use something similar to a TWAP oracle design, which would minimize the deviations possible while still providing reliable pricing. We believe this to be a more appropriate design choice given the anticipated lack of volatility in the asset being priced.

Remediation

This issue has been acknowledged by YO Foundation. YO Foundation also stated the following in response:

> We have made no changes here as effectively this finding is saying there is a maximum price change of 2 * max_pct_change every 2 * window_slots, which is intended behaviour. We have been using this formula in production and migrating to a more complicated TWAP system would increase scope for errors. We can always reduce the max_pct_change if needed.

3.5. Missing liveness check for oracle

Target	yo-vault		
Category	Protocol Risks	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The oracle program uses a `price` and `anchor_price` in its logic. While `price` represents the most up-to-date price, the `anchor_price` is a baseline price that gets updated every `window_slots` slots. This is to ensure that the `price` does not deviate by more than `max_pct_change` from the `anchor_price` during a window.

One check that is missing from this logic is a liveness check (a check if the oracle been updated recently). If we assume that the oracle updates fail for an extended period of time, the real price of the shares might still move by a substantial amount. Any user of the program can still interact with the program, but it will use the last set `price` from when the oracle was still active, which can lead to substantial losses for the operator or the user.

Impact

Stale oracle updates could be used for share and asset calculations as an upper bound is missing on the staleness or the last set `price`. This may lead to substantial losses for the operator or user in the case that the real price has deviated significantly since the last successful update.

Recommendations

Implement an upper bound on the time since the last successful oracle update to ensure that the program is halted if the price is too stale. In any logic that is accessing the price, the `last_slot_updated` from when that update happened should be checked and compared to the current slot. If the delta between the present and the last update is too big, the program should refuse operation until a fresh oracle update arrives.

Remediation

This issue has been acknowledged by YO Foundation. YO Foundation also stated the following in response:

We have made no changes here. Given the share price ticks up very gradually over time, we believe it is a more disruptive user experience to prevent deposits / redeems if our service goes down. Users should have unfettered access to their funds, even if it is at a worse rate. Also this is consistent with the system we have been running in production on EVM.

3.6. Missing usage of `checked_add` in `mul_div`

Target	crates/yo-lib/src/libraries/math.rs		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Informational

Description

The implementation of the `mul_div` function in yo-lib uses unchecked addition in case the `round_up` flag is set.

```
pub fn mul_div(n: u128, m: u128, d: u128, round_up: bool) -> Result<u128> {
  if d == 0 {
    return Err(YoError::DivisionByZero.into());
  }
  if round_up {
    let nm = n.checked_mul(m).ok_or(YoError::MulOverflow)?;
    Ok((nm + d - 1) / d)
  } else {
    Ok(n.checked_mul(m).ok_or(YoError::MulOverflow)? / d)
  }
}
```

While all multiplication is done using `checked_mul`, the addition of the denominator is done using unchecked addition, which can lead to overflows.

As of the time of writing, this library is only used as part of the workspace, which contains the other yo programs. Additionally, the workspace has `overflow-checks = true` set in cargo.toml, preventing overflows in binaries within the workspace. However, if this library is ever used outside of the current workspace, the overflow check will *not* be applied, which could lead to overflows in the `round_up` branch of the `mul_div` function.

Impact

The usage of unchecked addition in the `round_up` branch may lead to overflows if the library is used outside of the current workspace that sets the `overflow-checks = true` flag in cargo.toml.

Recommendations

Make use of the `checked_add` function to prevent overflows.

Remediation

This issue has been acknowledged by YO Foundation, and a fix was implemented in commit 963b9d4e ↗.

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. Guarding initialization functions

The vault as well as the oracle program make use of an initialization function to set the initial state. The initialization functions are annotated with a comment stating that these functions are "unguarded",

```
/// Unguarded - should be called immediately after deployment.
```

indicating the risk of being front-run if the `initialize` instruction is not called in the same transaction as the deployment of the program. One way to eliminate the front-running risk entirely could be the usage of the following pattern, which ensures that the authority of the deployed program is a signer of the `initialize` instruction, ensuring it can only be called by the deployer itself.

```
pub struct SomeContext<'info> {
    pub authority: Signer<'info>,
    #[account(address = crate::ID)]
    #[account(constraint = program.programdata_address()
    == Ok(Some(program_data.key()))]
    pub program: Program<'info, crate::program::ProgramName>,
    #[account(constraint = program_data.upgrade_authority_address
    = Some(authority.key()))]
    pub program_data: Account<'info, ProgramData>,
}
```

4.2. General suggestions and improvements

Throughout the audit, we observed and relayed a few minor improvements to the codebase. These suggestions are not motivated via perceived risks and are purely suggestions to improve compute unit cost, code quality, and maintainability.

Unnecessary cast in yo-lib

We noticed an unnecessary cast in the `percentage_change` function in the yo-lib library. It appears that in the past, this function may used to have returned `Result<u64, _>`, but it no longer does. Despite this, a conversion from `u128` to `u128` remains which can be removed.

```
let pct = pct_u128
  .try_into()
  .map_err(|_| YoError::DowncastU128ToU64Overflow)?;
```

This issue has been acknowledged by YO Foundation, and a fix was implemented in commit `36d8aff0`.

Suggested use of abs-diff

A common pattern in the codebase is to compare two figures and take the difference starting with the larger figure and subtracting the smaller figure. This is used to get the absolute difference between these two figures for determining the percent deviation from some current standard. This can be seen here.

```
let diff = if price > vault_data.anchor_price {
    price - vault_data.anchor_price
} else {
    vault_data.anchor_price - price
};
```

Instead, we suggest considering using `u128::abs_diff(...)`, which performs this operation as well. This suggestion is largely motivated by code cleanliness reasons, but it also reduces the changes of subtle deviations in the logic performed in each branch.

This issue has been acknowledged by YO Foundation, and a fix was implemented in commit `978de8a6`.

Unnecessary `mut` constraint

Some of the instructions use `mut` on accounts that certainly do not need it. For example, in the `initialize_vault` instruction in the yo-vault program, the `oracle_config` account is marked as mutable despite no mutations occurring. Another instance is the `oracle_vault_data` account in the `deposit` instruction in yo-vault. This account similarly does not need `mut`, especially because the account is owned by another program (yo-oracle) and no attempts at mutation are made. Given the circumstances, the `mut` constraints can be removed.

This issue has been acknowledged by YO Foundation, and a fix was implemented in commit `0d67ec41`.

5. Threat Model

As time permitted, we analyzed each instruction in the program and created a written threat model for the most critical instructions. A threat model documents the high-level functionality of a given instruction, the inputs it receives, and the accounts it operates on as well as the main checks performed on them; it gives an overview of the attack surface of the programs and of the level of control an attacker has over the inputs of critical instructions.

For brevity, system accounts and well-known program accounts have not been included in the list of accounts received by an instruction; the instructions that receive these accounts make use of Anchor types, which automatically ensure that the public key of the account is correct.

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

Instruction: `initialize`

This instruction is responsible for initializing the program and setting the initial state. The initialize function is missing front-run protections, which make it technically possible for a malicious attacker to invoke the instruction before the program deployer can, assuming the program deployment and initialization do not happen in the same transaction. A possible design pattern that prevents front-running has been suggested in section 4.1. ↗.

Input parameters

```
admin: Pubkey,
updater: Pubkey,
default_window_slots: u64,
default_max_pct_change: u64,
```

Accounts

Fixed accounts

- **`caller`** (index 0): The caller that will initialize the program.
  - Signer: Yes.
  - Init: No.
  - PDA: No.
  - Writable: Yes.
  - Constraints: None.
- **`oracle_config`** (index 1): Oracle config.
  - Signer: No.
  - Init: Yes.

- PDA: Yes.
- Writable: Yes.
- Constraints: Seeds must be `OracleConfig::SEED_PREFIX.as_bytes()`.

- **`system_program`** (index 2): System program.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: No.
  - Constraints: Must be the system program.

Additional checks and behavior

- `admin != Pubkey::default()`.
- `updater != Pubkey::default()`.
- `default_window_slots > 0`.
- `default_max_pct_change > 0`.
- `default_max_pct_change <= MAX_PCT_CHANGE`.

Instruction: `update_vault_price`

This instruction handles updating the price in the vault.

Two issues were identified that relate to the price-updating logic.

1. The first issue is that the static window approach for the `anchor_price` and `price` logic can lead to an acute price drop or increase that exceeds the `max_pct_change` if two oracle updates land in the same slot and that slot is also responsible for setting a new `anchor_price`. This has been reported in Finding 3.4. ↗.
2. The second issue is a missing liveness check that would prevent overly stale prices from being used, in the case the oracle has an outage for an extended period of time. This has been reported in Finding 3.5. ↗.

The instruction can only be called by the updater configured in the oracle config.

Input parameters

```
price: u128
```

Accounts

Fixed accounts

- **`updater`** (index 0): Updater account.
    - Signer: Yes.
    - Init: No.
    - PDA: No.
    - Writable: Yes.
    - Constraints: Must be `updater.key() == oracle_config.updater`.
- **`vault_share_mint`** (index 1): Address of the vault share token.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: Must be a valid mint account.
- **`vault_data`** (index 2): Vault data account.
    - Signer: No.
    - Init: No.
    - PDA: Yes.
    - Writable: Yes.
    - Constraints: Seeds must be `OracleVaultData::SEED_PREFIX.as_bytes()`, `vault_program.key().as_ref()` and `vault_share_mint.key() == vault_data.share_mint`.
- **`oracle_config`** (index 3): Oracle config account.
    - Signer: No.
    - Init: No.
    - PDA: Yes.
    - Writable: Yes.
    - Constraints: Seeds must be `OracleConfig::SEED_PREFIX.as_bytes()`.
- **`vault_program`** (index 4): Vault program.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: None.
- **`system_program`** (index 5): System program.
    - Signer: No.
    - Init: No.

- PDA: No.
- Writable: No.
- Constraints: Must be the system program.

Additional checks and behavior

- `price > 0`.
- `diff_pct <= max_pct_change`.

Instruction: `manage_unchecked`

This instruction is responsible for proxying a whitelisted CPI call, where the output and the input of the CPI call remain unchecked. The instruction can only be called by the operator configured in the vault data account. This and the other `manage_*` functions give the program being invocated by the CPI a lot of power due to the `authority` being the owner of the asset token as well as the share token wallet.

This has been described in more detail in Finding 3.2. ↗.

Input parameters

```
data: Vec<u8>,
discriminator_bytes: u8,
```

Accounts

Fixed accounts

- **`operator`** (index 0): Operator account.
  - Signer: Yes.
  - Init: No.
  - PDA: No.
  - Writable: Yes.
  - Constraints: Must be `operator.key() == vault_data.operator`.
- **`authority`** (index 1): Authority of the vault.
  - Signer: No.
  - Init: No.
  - PDA: Yes.
  - Writable: Yes.

    - Constraints: Seeds must be `YO_VAULT_AUTH_SEED.as_bytes()`.
- **`vault_data`** (index 2): Vault data account.
    - Signer: No.
    - Init: No.
    - PDA: Yes.
    - Writable: Yes.
    - Constraints: Seeds must be `OracleVaultData::SEED_PREFIX.as_bytes()`.
- **`ix_whitelist`** (index 3): Program instruction whitelist.
    - Signer: No.
    - Init: No.
    - PDA: Yes.
    - Writable: No.
    - Constraints: Seeds must be `IxWhitelist::SEED_PREFIX.as_bytes(), target_program.key().as_ref(), &data[0..discriminator_bytes as usize]`.
- **`target_program`** (index 4): Target program.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: None.
- **`token_program`** (index 5): Token program.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: Must be the Token or Token2022 program.
- **`associated_token_program`** (index 6): Associated token program.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: Must be the associated token program.
- **`system_program`** (index 7): System program.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.

- Constraints: Must be the system program.

Additional checks and behavior

- `ix_whitelist.is_whitelisted` is true.
- `discriminator_bytes > 0`.

Instruction: `cancel_redeem`

This instruction will cancel a pending redemption. The instruction can only be called by the operator configured in the vault data account. In certain circumstances, it might be possible to cause a denial of service to the redeem system if the `fulfill_redeem` instruction were called and set either the pending assets or shares to zero. This was reported in Finding 3.1. ↗.

Input parameters

None.

Accounts

Fixed accounts

- **`operator`** (index 0): Operator account.
  - Signer: Yes.
  - Init: No.
  - PDA: No.
  - Writable: Yes.
  - Constraints: Must be `operator.key() == vault_data.operator`.
- **`user`** (index 1): User account.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: Yes.
  - Constraints: Must be a system account.
- **`authority`** (index 2): Authority of the vault.
  - Signer: No.
  - Init: No.
  - PDA: Yes.
  - Writable: Yes.

    - Constraints: Seeds must be `YO_VAULT_AUTH_SEED.as_bytes()`.
- **`vault_data`** (index 3): Vault data account.
    - Signer: No.
    - Init: No.
    - PDA: Yes.
    - Writable: Yes.
    - Constraints: Seeds must be `OracleVaultData::SEED_PREFIX.as_bytes()`.
- **`pending_redeem_state`** (index 4): User pending redeem state account. This account tracks the state of pending redemptions for a given user.
    - Signer: No.
    - Init: No.
    - PDA: Yes.
    - Writable: Yes.
    - Constraints: Seeds must be `UserPendingRedeem::SEED_PREFIX.as_bytes(), user.key().as_ref()`.
- **`token_mint`** (index 5): Vault token mint.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: Must be a valid mint account and `token_mint.key() == vault_data.token_mint`.
- **`share_mint`** (index 6): Share token mint.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: Must be a valid mint account and `vault_data.share_mint == share_mint.key()`.
- **`auth_share_vault`** (index 7): Share token vault.
    - Signer: No.
    - Init: Yes.
    - PDA: No.
    - Writable: Yes.
    - Constraints: `associated_token::mint = share_mint`, `associated_token::authority = authority`, and `associated_token::token_program = token_program`.
- **`user_share_account`** (index 8): User-share token account.
    - Signer: No.

- Init: Yes.
- PDA: No.
- Writable: Yes.
- Constraints: `associated_token::mint = share_mint`, `associated_token::authority = user`, and `associated_token::token_program = token_program`.

- **`token_program`** (index 9): Token program.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: No.
  - Constraints: Must be the Token or Token2022 program.
- **`associated_token_program`** (index 10): Associated token program.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: No.
  - Constraints: Must be the associated token program.
- **`system_program`** (index 11): System program.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: No.
  - Constraints: Must be the system program.

Additional checks and behavior

- `shares > 0`.
- `assets > 0`.

Instruction: `fulfill_redeem`

This instruction handles the fulfillment of a user's redemption request. The instruction can only be called by the operator configured in the vault data account.

Input parameters

```
shares: u64,
assets: u64
```

Accounts

Fixed accounts

- **`operator`** (index 0): Operator account.
  - Signer: Yes.
  - Init: No.
  - PDA: No.
  - Writable: Yes.
  - Constraints: Must be `operator.key() == vault_data.operator`.
- **`user`** (index 1): User account.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: Yes.
  - Constraints: Must be a system account.
- **`authority`** (index 2): Authority of the vault.
  - Signer: No.
  - Init: No.
  - PDA: Yes.
  - Writable: Yes.
  - Constraints: Seeds must be `YO_VAULT_AUTH_SEED.as_bytes()`.
- **`fee_recipient`** (index 3): Fee recipient.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: Yes.
  - Constraints: Must be a system account and `fee_recipient.key() == vault_data.fee_recipient`.
- **`vault_data`** (index 4): Vault data account.
  - Signer: No.
  - Init: No.

    - PDA: Yes.
    - Writable: Yes.
    - Constraints: Seeds must be `OracleVaultData::SEED_PREFIX.as_bytes()`.
- **`pending_redeem_state`** (index 5): User pending redeem state account. This account tracks the state of pending redemptions for a given user.
    - Signer: No.
    - Init: No.
    - PDA: Yes.
    - Writable: Yes.
    - Constraints: Seeds must be `UserPendingRedeem::SEED_PREFIX.as_bytes(), user.key().as_ref()`.
- **`token_mint`** (index 6): Vault token mint.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: Must be a valid mint account and `token_mint.key() == vault_data.token_mint`.
- **`share_mint`** (index 7): Share token mint.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: Must be a valid mint account and `vault_data.share_mint == share_mint.key()`.
- **`auth_share_vault`** (index 8): Authority share token vault.
    - Signer: No.
    - Init: Yes.
    - PDA: No.
    - Writable: Yes.
    - Constraints: `associated_token::mint = share_mint`, `associated_token::authority = authority`, and `associated_token::token_program = token_program`.
- **`auth_token_vault`** (index 9): Authority token vault.
    - Signer: No.
    - Init: Yes.
    - PDA: No.
    - Writable: Yes.
    - Constraints: `associated_token::mint = token_mint`,

`associated_token::authority = authority`, and `associated_token::token_program = token_program`.

- **`user_token_account`** (index 10): User token account.
  - Signer: No.
  - Init: Yes.
  - PDA: No.
  - Writable: Yes.
  - Constraints: `associated_token::mint = token_mint`, `associated_token::authority = user`, and `associated_token::token_program = token_program`.
- **`fee_recipient_token_account`** (index 11): Fee-recipient token account.
  - Signer: No.
  - Init: Yes.
  - PDA: No.
  - Writable: Yes.
  - Constraints: `associated_token::mint = token_mint`, `associated_token::authority = fee_recipient`, and `associated_token::token_program = token_program`.
- **`token_program`** (index 12): Token program.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: No.
  - Constraints: Must be the Token or Token2022 program.
- **`associated_token_program`** (index 13): Associated token program.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: No.
  - Constraints: Must be the associated token program.
- **`system_program`** (index 14): System program.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: No.
  - Constraints: Must be the system program.

Additional checks and behavior

- `shares > 0`.
- `assets > 0`.
- `pct_deviation <= vault_data.max_pct_deviation`.
- `pending.pending_shares >= shares`.
- `pending.pending_assets >= assets`.

Instruction: `initialize`

This instruction handles the initialization of the program, including setting the initial state. The initialize function is missing front-run protections, which makes it technically possible for a malicious attacker to invoke the instruction before the program deployer can, assuming the program deployment and initialization do not happen in the same transaction. A possible design pattern that prevents front-running has been suggested in section 4.1. ↗.

Input parameters

```
admin: Pubkey,
oracle: Pubkey,
operator: Pubkey,
fee_recipient: Pubkey,
max_pct_deviation: u64,
```

Accounts

Fixed accounts

- **`caller`** (index 0): The caller that will initialize the program.
  - Signer: Yes.
  - Init: No.
  - PDA: No.
  - Writable: Yes.
  - Constraints: None.
- **`vault_data`** (index 1): Vault data account.
  - Signer: No.
  - Init: No.
  - PDA: Yes.
  - Writable: Yes.

    - Constraints: Seeds must be `OracleVaultData::SEED_PREFIX.as_bytes()`.
- **`authority`** (index 2): Authority of the vault.
    - Signer: No.
    - Init: No.
    - PDA: Yes.
    - Writable: Yes.
    - Constraints: Seeds must be `YO_VAULT_AUTH_SEED.as_bytes()`.
- **`token_mint`** (index 3): Vault token mint.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: Must be a valid mint account and `token_mint.key() == vault_data.token_mint`.
- **`share_mint`** (index 4): Share token mint.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: Must be a valid mint account and `vault_data.share_mint == share_mint.key()`.
- **`auth_share_vault`** (index 5): Authority share token vault.
    - Signer: No.
    - Init: Yes.
    - PDA: No.
    - Writable: Yes.
    - Constraints: `associated_token::mint = share_mint`, `associated_token::authority = authority`, and `associated_token::token_program = token_program`.
- **`system_program`** (index 6): System program.
    - Signer: No.
    - Init: No.
    - PDA: No.
    - Writable: No.
    - Constraints: Must be the system program.
- **`token_program`** (index 7): Token program.
    - Signer: No.
    - Init: No.

  - PDA: No.
  - Writable: No.
  - Constraints: Must be the Token or Token2022 program.
- **`associated_token_program`** (index 8): Associated token program.
  - Signer: No.
  - Init: No.
  - PDA: No.
  - Writable: No.
  - Constraints: Must be the associated token program.

Additional checks and behavior

- `admin != Pubkey::default()`.
- `oracle != Pubkey::default()`.
- `operator != Pubkey::default()`.
- `fee_recipient != Pubkey::default()`.
- `max_pct_deviation > 0`.
- `max_pct_deviation <= MAX_PCT_CHANGE`.

6. Assessment Results

During our assessment on the scoped YO programs, we discovered six findings. No critical issues were found. Five findings were of low impact and the remaining finding was informational in nature.

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.