Revenue Distribution

Security Assessment

October 29th, 2025 — Prepared by OtterSec

Ajay Shankar Kunapareddy	d1r3wolf@osec.io
Xiang Yin	soreatu@osec.io
Alpha Toure	shxdow@osec.io
Robert Chen	r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

DoubleZero engaged OtterSec to assess the `revenue-distribution` program. This assessment was conducted between September 15th and September 19th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 4 findings throughout this audit engagement.

In particular, we identified a vulnerability in the reward distribution logic, which increments the count of distributed rewards each time rewards are distributed, with no check to ensure the count does not exceed the total number of contributors (OS-DRD-ADV-00). Additionally, we highlighted the possibility where the reward distribution function may process empty ContributorRewards accounts (OS-DRD-ADV-01).

We also made suggestions for modifying the codebase to improve consistency and functionality (OS-DRD-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/doublezerofoundation/doublezero-solana. This audit was performed against commit 9521453.

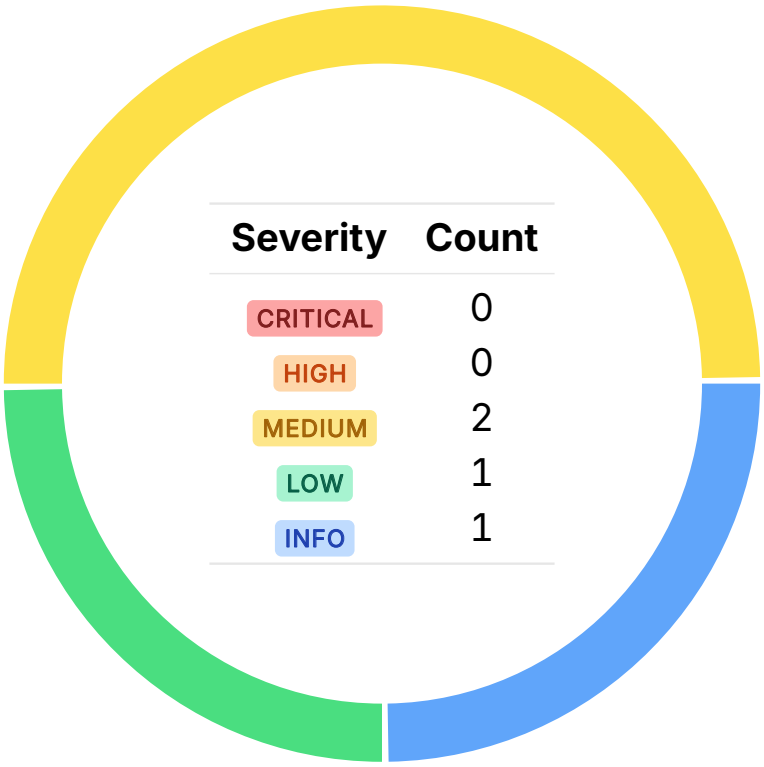
A brief description of the program is as follows:

Name	Description
revenue-distribution	A Solana on-chain protocol that manages the collection, accounting, and distribution of DoubleZero (2Z) token revenues to participants.

03 — Findings

Overall, we reported 4 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	0
HIGH	0
MEDIUM	2
LOW	1
INFO	1

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-DRD-ADV-00	MEDIUM	RESOLVED	Due to a lack of check on `distributed_rewards_count`, the distribution account may exceed its contributor limit, risking lamport depletion and accidental closure.
OS-DRD-ADV-01	MEDIUM	RESOLVED	Currently, when distributing rewards, the function may process empty `ContributorRewards` accounts, resulting in incorrect token burns and distribution state updates.
OS-DRD-ADV-02	LOW	RESOLVED	No minimum time check exists between initializing `Distribution` accounts, allowing rapid or duplicate executions that may create multiple unintended distributions.

Risk of Distribution Account Overflow MEDIUM OS-DRD-ADV-00

Description

In `try_distribute_rewards`, the `distributed_rewards_count` field is incremented each time rewards are distributed, but there is no check to ensure it does not exceed `total_contributors`. On Solana, accounts must maintain a minimum lamport balance to remain rent-exempt.

If `distributed_rewards_count` surpasses the number of contributors, the distribution account may drain its lamports. This may result in the account falling below the rent-exemption threshold, and as a result, it will be closed by the runtime.

Remediation

Verify `distribution.distributed_rewards_count <= distribution.total_contributors` to prevent account closure due to insufficient lamports for rent exemption.

Patch

Resolved in PR#79.

Failure to Verify Contributor Reward Recipent MEDIUM OS-DRD-ADV-01

Description

`try_distribute_rewards` currently does not check whether the `ContributorRewards` account has any active recipients. An unconfigured `contributor_rewards` account should not be able to receive rewards. If the account is empty, the function will skip the token transfers but still update the distribution state and burn the reward amount.

>_ *src/state/contributor_rewards/recipient_shares.rs* RUST

```rust
pub fn new(recipients: &[(Pubkey, u16)]) -> Option<Self> {
    [...]
    for (i, (recipient_key, share)) in recipients.iter().enumerate() {
        if recipient_key == &Pubkey::default() {
            return None;
        }
        let share = UnitShare16::new(*share)?;
        // Keep track of the running sum of shares to make sure it does not
        // exceed 100%.
        total_share = total_share.checked_add(share)?;
        out[i] = RecipientShare {
            recipient_key: *recipient_key,
            share,
        };
    }
    if total_share != UnitShare16::MAX {
        return None;
    }
    Some(Self(out))
}
```

Also, the current implementation of `RecipientShares::new` (shown above) allows recipients with zero `share`, wasting slots. Add a `share > 0` check to ensure only meaningful recipients are included.

Remediation

Check that the length of `contributor_rewards.recipient_shares.active_iter` is greater than zero.

Patch

Resolved in PR#77.

Lack of Time Interval for Distribution Initialization LOW OS-DRD-ADV-02

Description

`try_initialize_distribution` does not enforce a minimum time interval between consecutive distribution initializations. Since the instruction may be triggered via the frontend, duplicate or rapid re-executions may create multiple `Distribution` accounts in quick succession. This results in unexpected multiple distributions and increased on-chain storage.

Remediation

Implement a delay (such as for 24 hours) between initializations to prevent rapid creations.

Patch

Resolved in PR#78.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-DRD-SUG-00	Recommendation for modifying the codebase to improve consistency and functionality.

Code Refactoring OS-DRD-SUG-00

Description

1. The comments in `processor::try_sweep_distribution_tokens` incorrectly label the last three accounts, which may create confusion about their expected order. Update the comments for the last three accounts to state the accounts as Account 7, Account 8, and Account 9, respectively, to ensure clarity and consistency.
2. The `journal` account in the `sol_2z_swap_program::dequeue_fills` CPI is unnecessarily marked as writable (`mu`); it should be read-only since no state changes are made.

>_ *src/instruction/account.rs* RUST

```rust
impl From<DequeueFillsCpiAccounts> for Vec<AccountMeta> {
    fn from(accounts: DequeueFillsCpiAccounts) -> Self {
        let DequeueFillsCpiAccounts {
            configuration_registry_key,
            program_state_key,
            fills_registry_key,
            journal_key,
            sol_2z_swap_program_id: _,
        } = accounts;

        vec![
            AccountMeta::new_readonly(configuration_registry_key, false),
            AccountMeta::new_readonly(program_state_key, false),
            AccountMeta::new(fills_registry_key, false),
            AccountMeta::new(journal_key, true),
        ]
    }
}
```

Remediation

Update the codebase with the above refactors.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.