DoubleZero Passport

Security Assessment

October 31st, 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

01 — Executive Summary

Overview

DoubleZero engaged OtterSec to assess the `passport` program. This assessment was conducted between September 8th and September 12th, 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 where user refunds may be misdirected while granting access, allowing a malicious sentinel to steal deposits or result in transaction failures by not validating the rent beneficiary specified in `AccessRequest` (OS-DZP-ADV-00). Additionally, storing the deposit and fee only in the global configuration may result in overcharging old access requests or misprocessing them if the configuration changes (OS-DZP-ADV-01).

We also made suggestions to improve efficiency and to ensure adherence to coding best practices (OS-DZP-SUG-01), and advised zeroing out account data and setting the discriminator to ”closed” during closure of `AccessRequest` account in Passport to prevent account revival (OS-DZP-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at github.com/doublezerofoundation/doublezero-solana. This audit was performed against commit dcf672c.

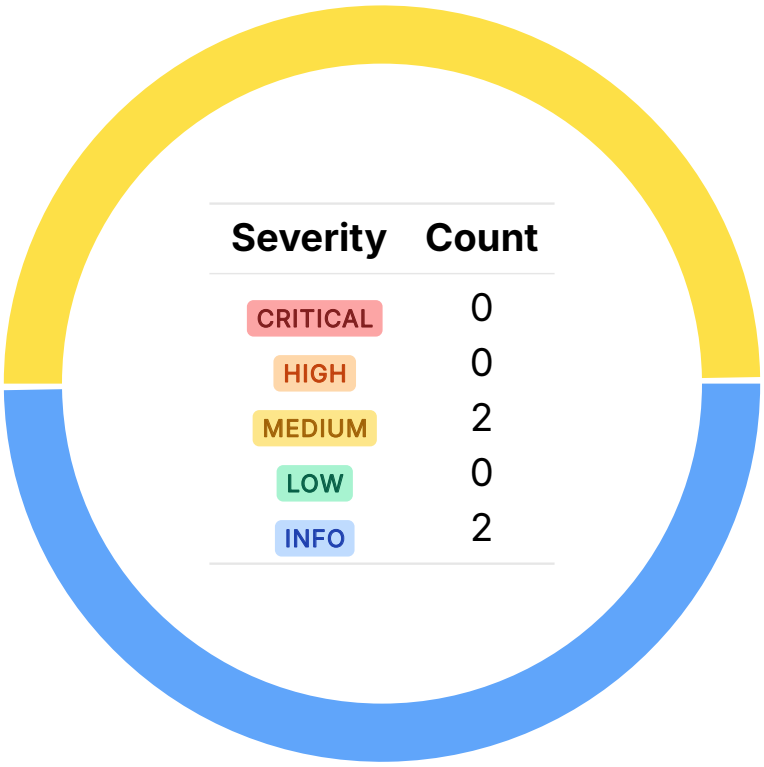
A brief description of the program is as follows:

Name	Description
passport	An access-control gatekeeper for the DoubleZero Ledger network. It manages program configuration (admin, sentinel, fees, pause state), allows users or services submit on-chain access requests, and enables the sentinel to grant or deny those requests, with deposits and refunds handled securely.

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	0
INFO	2

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-DZP-ADV-00	MEDIUM	RESOLVED	`try_grant_access` may misdirect user refunds, allowing a malicious sentinel to steal deposits or result in transaction failures by not validating the rent beneficiary specified in `AccessRequest`.
OS-DZP-ADV-01	MEDIUM	RESOLVED	Storing deposit and fee only in the global configuration may result in overcharging old access requests or mis-processing them if the configuration changes.

Absence of Refund Account Validation MEDIUM OS-DZP-ADV-00

Description

There is a lack of validation ensuring that the refund account matches the `AccessRequest`'s stored beneficiary. `try_grant_access` ignores the `rent_beneficiary_key` stored in the `AccessRequest` account and instead utilizes the next account supplied in the instruction as the refund recipient. This allows a malicious sentinel to redirect the refund to any writable account, effectively enabling theft of user funds. If a non-writable account or the access request account itself is utilized, the transaction will fail due to lamport conservation rules. This breaks the intended guarantee that the original payer receives their deposit back.

Remediation

Explicitly check that the refund recipient equals `access_request.rent_beneficiary_key`. This ensures refunds go only to the original depositor.

Patch

Resolved in PR#59.

Improper Configuration Updates MEDIUM OS-DZP-ADV-01

Description

Currently, the deposit and fee values are stored only in the global `ProgramConfig` and not per `AccessRequest`. If the configuration is updated after a request is created, `try_grant_access` or `try_deny_access` will apply the latest fee to an older request. This may result in incorrect refunds to the rent beneficiary, overcharging or underpaying the user, or even transaction failures due to lamport conservation violations if the updated fee exceeds the deposited lamports. Thus, changes to global parameters may retroactively affect pending requests.

>_ *programs/passport/src/state/program_config.rs* RUST

```rust
#[derive(Debug, Clone, Copy, Default, PartialEq, Eq, Pod, Zeroable)]
#[repr(C, align(8))]
pub struct ProgramConfig {
    pub flags: Flags,

    pub admin_key: Pubkey,

    /// Authority that grants or denies access to the DoubleZero Ledger network.
    pub sentinel_key: Pubkey,

    pub access_request_deposit_parameters: AccessRequestDepositParameters,

    /// 8 * 32 bytes of a storage gap in case more fields need to be added.
    _storage_gap: StorageGap<8>,
}

#[derive(Debug, Clone, Copy, Default, PartialEq, Eq, Pod, Zeroable)]
#[repr(C, align(8))]
pub struct AccessRequestDepositParameters {
    pub request_deposit_lamports: u64,
    pub request_fee_lamports: u64,
}
```

Remediation

Implement per-request fee storage in `AccessRequest`.

Patch

Resolved in PR#58.

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-DZP-SUG-00	Closing `AccessRequest` accounts by only draining lamports may leave valid data behind, allowing the account to be revived and resulting in denial-of-service or failed grant/deny operations.
OS-DZP-SUG-01	Suggestions regarding ensuring adherence to coding best practices.

Possibility of Reviving Access Request Account Post Closure OS-DZP-SUG-00

Description

There is a possibility of a revival attack occurring because the `AccessRequest` account is closed by draining lamports without clearing its data or resetting the discriminator. Although the rent exemption is lost, the account still retains valid fields and a discriminator, so if lamports are re-added before garbage collection, the account may be revived and mistakenly treated as active. This creates a denial-of-service risk, since the requester cannot submit a new request with the same PDA (Program Derived Address), and attempts to grant or deny access may fail due to lamport conservation unless additional funds are added. While the impact is minimal, it may block normal request flows and burden the sentinel.

Remediation

Zero out account data or set the discriminator to ”closed” during account closure.

Code Maturity OS-DZP-SUG-01

Description

1. Currently, sentinel instructions such as `try_grant_access` and `try_deny_access` bypass the program's pause check, allowing privileged actions even when the program is paused. This undermines the pause mechanism and may let a sentinel manipulate access or funds. Adding a pause check to these instructions ensures that all critical operations respect the paused state.
2. Prevent zero-value deposits, as they serve no purpose. Enforcing a non-zero deposit ensures meaningful access requests and avoids unnecessary operations.

Remediation

Implement the above mentioned suggestions.

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.