Adevar Labs

Prepared for:
DoubleZero

November 12, 2025

Audit Report

Version: Final Audit Report

Table of Contents

1. Executive Summary

About DoubleZero

The DoubleZero network is a global, high-performance routing layer that enables validators to connect with minimal latency by moving data directly from point A to point B. This audit focused on revenue distribution program, where the contributors receive the rewards paid by the validator debt.

Audit Summary

- Number of Findings: 0 total
- Critical: 0
- High: 0
- Medium: 0
- Low: 0
- Number of Enhancement Opportunities: 2

Risk Profile

The following table summarizes the distribution of identified vulnerabilities by risk level:

Risk Level	Count	Fixed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	0	0	0

Overall Security Posture

The DoubleZero revenue distribution program is generally secure and robust. Core functionality, including reward distribution, SOL withdrawal, sweeping and merkle proof inclusion, has been thoroughly reviewed.

Audit findings include minor efficiency improvements (redundant signer check) and low-likelihood edge cases, such as closed token accounts potentially blocking distribution.

No high-severity vulnerabilities remain. Remaining points focus on efficiency and resilience against unlikely scenarios.

Important Invariant

Debt Forgiveness Timing and System Integrity

The protocol includes a debt forgiveness mechanism that transfers unpaid validator debt ("bad debt") from one epoch to the next. During the audit, we analyzed whether the permissionless **try_sweep_distribution_tokens** function could be front-run to prevent bad debt from being properly transferred.

Upon thorough analysis, we determined this scenario is not practically exploitable due to the following system constraints:

1. **Sequential Dependency**: The **try_sweep_distribution_tokens** function requires sufficient swapped SOL to be available in the system. This swapped SOL only becomes available after validators pay their debt and the swap completes. Therefore, calling sweep immediately after debt finalization (before debt forgiveness) is not feasible.

1. **Accountant Priority**: The **forgive_debt** function can be called immediately after debt finalization since it doesn't depend on funds being available, giving the accountant a natural timing advantage over any potential front-runner.

1. **Persistent Accountability**: Even if sweep occurs before debt forgiveness in a particular epoch, the unpaid debt remains trackable and can be forwarded to any future distribution period, as long as the debt is finalized and hasn't been swept yet. The debt does not become permanently unaccounted.

Operational Invariant: The system's security relies on the accountant executing debt forgiveness operations in a timely manner after each epoch's debt finalization. Under normal operation, where the accountant is actively monitoring the system, the architectural design naturally prevents abuse. This timing-dependent invariant should be maintained through proper operational procedures and accountant reliability monitoring.

Audit Scope

- **Repository:** double-zero-revenue-distribution (private)
- **Commit Hash:** 7019a4a403be9fad13cd63af5d1fe5e8925870b8
- **Files/Modules in Scope:** programs/revenue-distribution/src/*.rs

- **Repository:** double-zero-svm-hash-rs-main (private)
- **Commit Hash:** 02ed28feec731b71493fb8ab933316b47a8a167d
- **Files/Modules in Scope:**
- svm-hash-rs-main/src/*.rs

2. Assumptions and Considerations

Scope Limitations

The following limitations and constraints should be considered when reviewing this security assessment report:

1. **Time-Limited Assessment**: The security assessment was conducted over a 5-day period. As such, not all potential security vulnerabilities may have been identified. Security assessment is a point-in-time exercise, and new vulnerabilities may emerge following the completion of this assessment.

1. **Defined Scope**: The assessment was limited to the systems, applications, and networks explicitly defined in the scope. Systems outside the defined scope, even if they interact with in-scope systems, were not assessed. A list of in-scope assets is provided in the Introduction section of this report.

3. Severity Definitions

Each issue identified in this report is assigned a severity level based on two dimensions: **Impact** and **Likelihood**. These dimensions help project our team's understanding of both the potential consequences of a vulnerability and how likely a vulnerability is to be discovered and exploited in the real world.

Impact

Impact reflects the potential consequences of the issue–particularly on **project funds**, **user funds**, and the **availability or integrity** of the protocol.

- **High Impact**: Successful exploitation could result in a complete loss of user or protocol funds, disruption of core protocol functionality, or permanent loss of control over critical components.
- **Medium Impact**: Exploitation could cause significant disruption or partial loss of funds, but not a total compromise. May impact some users or non-core functionality.
- **Low Impact**: The issue has minor or negligible consequences. It may affect edge cases, expose metadata, or degrade performance slightly without putting funds or core logic at serious risk.

Likelihood

Likelihood reflects how easy a vulnerability is to discover and exploit by an attacker, as well as how economically attractive the exploit is to an attacker.

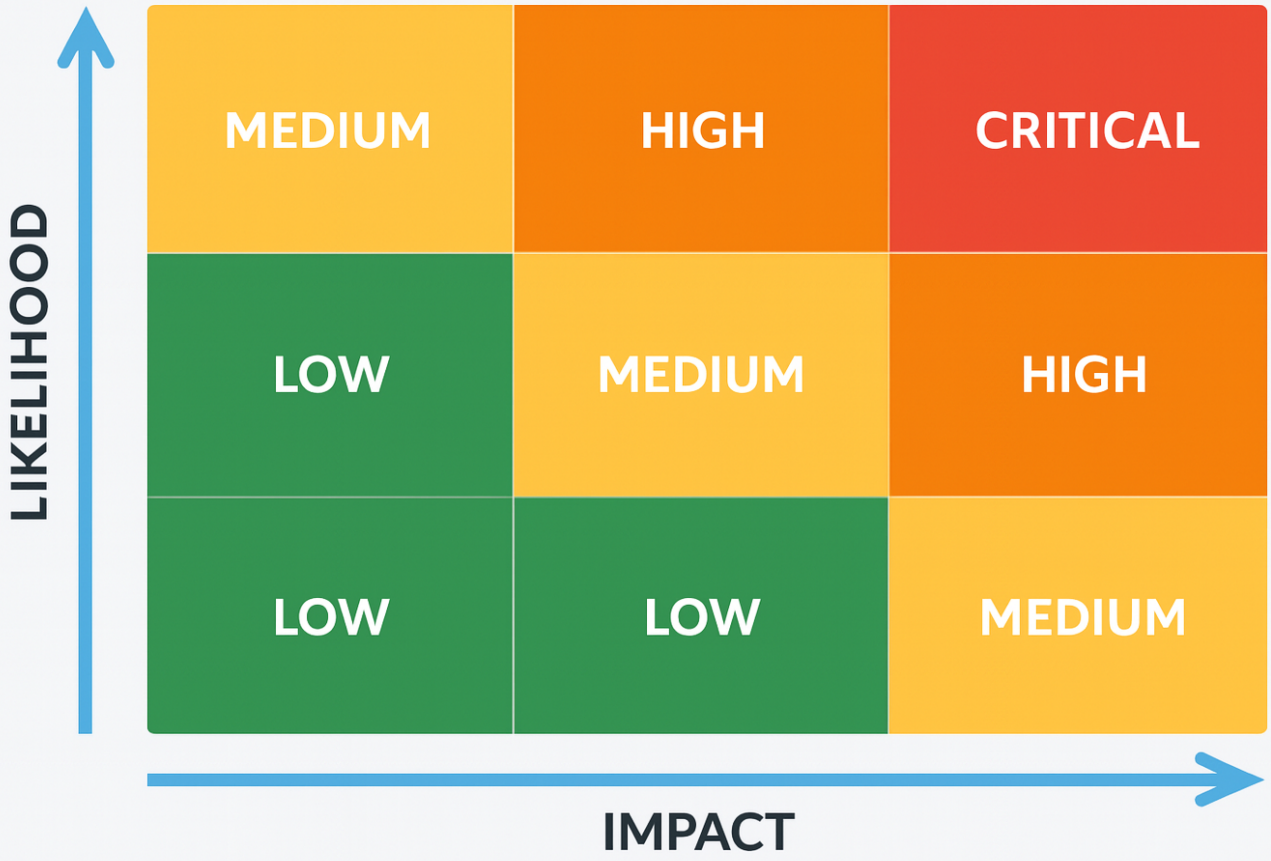
- **High Likelihood**: The vulnerability is trivially exploitable. This means it can be exploited by a wide range of actors without privileged access rights, with minimal capital requirements and low financial risks.
- **Medium Likelihood**: This type of vulnerability can be found and exploited with moderate effort. It might require a significant capital investment, but with manageable financial risk.
- **Low Likelihood**: Exploitation of these vulnerabilities is often technically unfeasible or requires highly specialized conditions. They may require extraordinary effort or a significant financial risk for an attacker, with a high chance of failure and minimal potential return.

Severity Classification Matrix

By combining **Impact** and **Likelihood**, we assign a severity level using the matrix below:

- **Critical**: High impact + high likelihood (e.g. a bug that could allow anyone to drain a substansial amount of protocol funds with minimal effort)
- **High**: High impact with medium likelihood, or medium impact with high likelihood
- **Medium**: Moderate impact and/or discoverability
- **Low**: Minimal impact or unlikely to be exploited

This structured approach helps teams prioritize fixes and mitigate the most dangerous threats first.



Severity Matrix

5. Enhancement Opportunities

E01: Duplicate Signer Check Increases Computational Overhead

Location: https://github.com/AdevarLabs/doublezero-rd-swap-merkle/blob/b68c1aaf876d74000a7ec67e6a6ebb99333df2fd/doublezero-solana-7019a4a403be9fad13cd63af5d1fe5e8925870b8/crates/program-tools/src/account_info/upgrade_authority.rs#L32-L43

>_ upgrade_authority.rs RUST

```
30:
31:         // Index == 1.
32:         let (index, owner_info) = try_next_enumerated_account(
33:             accounts_iter,
34:             NextAccountOptions {
35:                 must_be_signer: true,
36:                 ..Default::default()
37:             },
38:         )?;
39:
40:         if !owner_info.is_signer {
41:             msg!("Owner (account {}) must be signer", index);
42:             return Err(ProgramError::MissingRequiredSignature);
43:         }
44:
45:         let program_data_info_data = program_data_info.data.borrow();
```

Description:

The code performs an unnecessary signer check after calling try_next_enumerated_account with must_be_signer: true. Since the function already validates and enforces the signer requirement, the subsequent if !owner_info.is_signer check is redundant and can be removed.

RUST

```
let (index, owner_info) = try_next_enumerated_account(
            accounts_iter,
            NextAccountOptions {
                must_be_signer: true,
                ..Default::default()
            },
        )?;

        if !owner_info.is_signer {
            msg!("Owner (account {}) must be signer", index);
            return Err(ProgramError::MissingRequiredSignature);
        }
```

Potential Benefit:

Less CU consumption.

Recommendation:

The must_be_signer: true option already handles the validation and will return the appropriate error if the account is not a signer.

E02: Closed Token Account Causes Reward Distribution Failure

Location: https://github.com/AdevarLabs/doublezero-rd-swap-merkle/blob/b68c1aaf876d74000a7ec67e6a6ebb99333df2fd/doublezero-solana-7019a4a403be9fad13cd63af5d1fe5e8925870b8/programs/revenue-distribution/src/processor.rs#L1401

>_ processor.rs RUST

```
1399:         total_transferred_share_amount += recipient_share_amount;
1400:
1401:         let token_transfer_ix = token_instruction::transfer(
1402:             &spl_token::ID,
1403:             distribution_2z_token_pda_info.key,
```

Description:

When distributing rewards in **try_distribute_rewards**, the function attempts to transfer tokens to each recipient's associated token account (ATA) within a batch. If any recipient has closed their ATA, the SPL token transfer instruction will fail, causing the entire transaction to revert.

This can temporarily prevent reward distribution for all recipients within a contributor batch. However, since the instruction is permissionless, network contributors can execute it themselves once they have configured their recipients correctly.

Potential Benefit:

Prevent temporary reward distribution failures and improve user experience

Recommendation:

Although the Reward Manager can update the recipient for a given contributor, the logic should verify that the provided account is a valid associated token account (ATA) before attempting the transfer.

If the account is invalid, its iteration in the loop should be skipped to allow distribution to proceed for other valid recipients. The skipped rewards will remain available for distribution once the recipient account is properly configured.

About Us

About Adevar Labs

Adevar Labs is a boutique blockchain security firm specializing in web3 audits.

Built by a mix of experienced professionals in traditional enterprise and crypto natives who have contributed to some of the most critical projects in blockchain infrastructure.

Our team's background spans companies like Bitdefender, Asymmetric Research, Quantstamp, Chainproof, and Juicebox, and includes experience securing smart contracts, bridges, and L1 and L2 protocols across ecosystems like Solana, Ethereum, Polkadot, Cosmos, and MultiversX.

With over 100 audits completed and a portfolio that includes custom fuzzers, exploit modeling, and runtime testing frameworks, Adevar Labs brings both depth and precision to every engagement.

Our auditors have discovered critical vulnerabilities, built high-impact tooling, and placed in top positions in premier audit competitions including Code4rena and Sherlock.

Team members hold distinctions such as PhDs in software protection, and key roles at Fortune 500 companies, and leadership of flagship conferences like ETH Bucharest.

With team members having publications with over 1,100 academic citations and trusted by projects securing over $500M in on-chain value, Adevar Labs blends elite technical rigor with real-world security impact.

We also collaborate with some of the best independent security researchers in the web3 space.

Projects may optionally request specific contributors to be part of their audit.

Audit Methodology

Our audit methodology is specialized to provide thorough security assessments of Solana programs. We focus explicitly on rigorous manual analysis, detailed threat modeling, and careful validation of implemented fixes to ensure your programs operate securely and as intended.

1. Program Context and Architecture Analysis

Our auditors begin by deeply examining your Solana program's documentation, intended functionality, and account design. We meticulously map out program interactions, instruction processing flows, and state management logic. Special attention is given to understanding how your program interfaces with critical Solana system programs, such as the SPL Token Program, Stake Program, and System Program. Last but not least we check external integrations with other Solana projects and verify if the inputs and return values are handled properly.

2. Threat Modeling

We conduct targeted threat modeling tailored specifically for Solana's execution environment. We carefully define attacker capabilities and identify potential vulnerabilities that may arise from Solana-specific issues, including but not limited to:

- Unauthorized account data manipulation
- Improper ownership or signer verification

- Misuse of Program-Derived Addresses (PDAs)
- Incorrect use of Cross-Program Invocations (CPI)
- Failure to adequately handle account privileges or account states
- Risks stemming from rent-exemption and account initialization logic

3. In-depth Manual Security Review

Our experienced auditors perform an extensive manual security review of your Rust-based Solana programs. This involves a comprehensive line-by-line inspection of source code, focusing on common Solana vulnerabilities including, but not limited to:

- Missing or insufficient ownership checks
- Inadequate signer checks
- Incorrect handling of CPI calls and invocation privileges
- Arithmetic and integer overflow or underflow errors
- Unsafe deserialization and serialization of account data structures
- Improper token transfers and SPL-token logic issues
- Logic flaws in financial operations or state transitions
- Edge-case handling in instruction input validation
- Potential denial-of-service vectors related to transaction execution and account handling

During this stage, we clearly document any discovered vulnerabilities, including detailed descriptions, precise severity ratings, and recommendations for secure implementation. In situations where it is not clear how the vulnerability might be exploited we may also include a detailed proof-of-concept exploit including code snippets and instructions on how the exploit could be performed.

4. Detailed Fix Review and Validation

After the initial audit and your team's subsequent remediation efforts, we perform a comprehensive fix review to ensure the vulnerabilities identified have been effectively resolved.

We verify each fix individually, confirming that:

- Corrections effectively eliminate the security risks
- Changes do not inadvertently introduce new vulnerabilities or regressions
- The fixes align closely with Solana best practices and secure coding guidelines

Our detailed validation ensures that security improvements are robust, complete, and aligned with best practices specific to the Solana development ecosystem.

Confidentiality Notice

This report, including its content, data, and underlying methodologies, is subject to the confidentiality and feedback provisions in your agreement with Adevar Labs. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Adevar Labs.

Legal Disclaimer

The review and this report are provided by Adevar Labs on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Adevar Labs disclaims all warranties, expressed or implied, in connection with this report, its content, and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

You agree that access to and/or use of the report and other results of the review, including any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided.

You acknowledge that blockchain technology remains under development and is subject to unknown risks and flaws. Adevar Labs does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open-source or third-party software, code, libraries, materials, or information accessible through the report. As with the purchase or use of a product or service in any environment, you should use your best judgment and exercise caution where appropriate.