Zellic

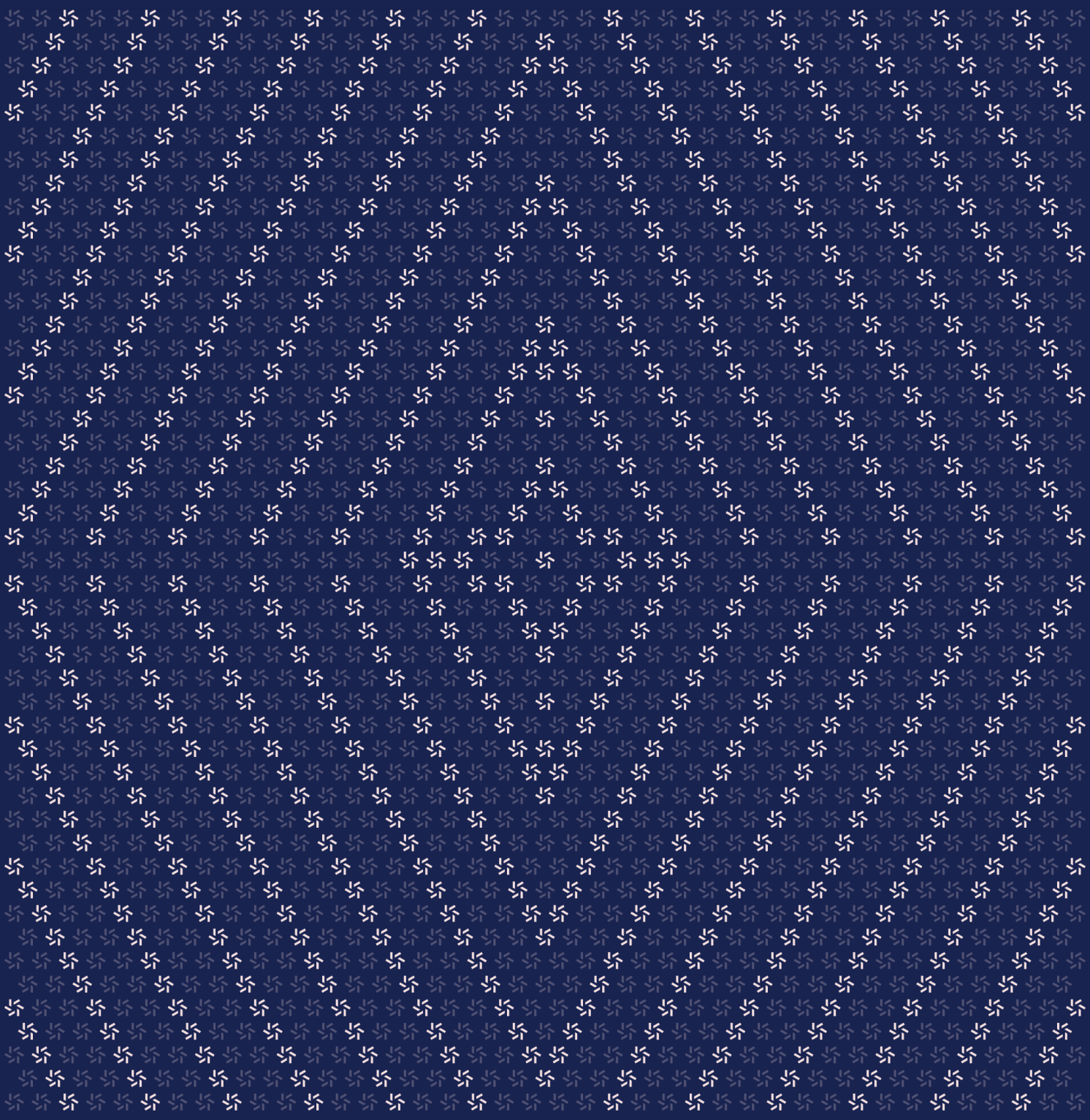
Prepared for
Ryan Zarick
Isaac Zhang
LayerZero Labs

Prepared by
Aaron Esau
Zellic

September 8, 2025

VeDistributor

Smart Contract 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 LayerZero Labs on September 5th, 2025. During this engagement, Zellic reviewed VeDistributor'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:

- Do all functions implement access control properly?
- Which values can increase or decrease over time? Are underflow or overflow reversions possible?
- Is it possible for a malicious entity to extract more USDC than intended?

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 VeDistributor contracts, there were no security vulnerabilities discovered.

Breakdown of Finding Impacts

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

2. Introduction

2.1. About VeDistributor

LayerZero Labs contributed the following description of VeDistributor:

> VeDistributor is a contract to distribute Stargate protocol revenue in form of USDC to veSTG holders, based on a snapshot taken on a given block of veSTG balances.

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 contracts.

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.

2.3. Scope

The engagement involved a review of the following targets:

VeDistributor Contracts

Type	Solidity
Platform	EVM-compatible
Target	stg-zro-swapper
Repository	https://github.com/LayerZero-Labs/stg-zro-swapper ↗
Version	`95e27590c91caa9d170ca987f0b2b0693068ceea`

Programs	`VeDistributor.sol`
	`interfaces/IVeDistributor.sol`
	`interfaces/IVotingEscrow.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.5 person-days. The assessment was conducted by one consultant over the course of two calendar days.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

Pedro Moura
Engagement Manager
pedro@zellic.io

The following consultant was engaged to conduct the assessment:

Aaron Esau
Engineer
aaron@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 5, 2025	Start of primary review period
September 5, 2025	End of primary review period
September 10, 2025	Scope changed from `8810a902` to `95e27590`

3. Threat Model

This section provides a threat model description of the assessed smart contract.

3.1. Chain VE total supply (`CHAIN_TOTAL_VE`)

This value is set during the contract's deployment. It is never updated in contract storage otherwise.

Therefore, it is assumed that regardless of what address is passed as the VotingEscrow `_ve`, the total supply at the specified chain snapshot block (`VE.totalSupplyAt(CHAIN_SNAPSHOT_BLOCK)` where `VE` is `_ve`) will neither increase nor decrease after contract deployment.

3.2. User VE balance (`veBalance`)

This value is used in two places: in `getUsdcClaimable` and in `_getAndCacheVeBalance`.

The former does not cache the value and will use whatever value is returned from the VE contract. The latter will use the cached value where the key is the `msg.sender`. This value is never updated once cached.

Therefore, it is assumed that the `veBalance` value returned by `VE.balanceOfAt(_user, CHAIN_SNAPSHOT_BLOCK)` will never increase nor decrease after contract deployment.

3.3. Distributed USDC amount (`usdcDistributedAmount`)

This value is incremented when `distribute` is called by the owner — and that function may be called at any time after the contract is deployed. It is never decremented.

It is used for calculating the amount of USDC the user may claim:

```
function _calculateUsdcClaimable(address _user, uint256 _veBalance)
    internal view returns (uint256 usdcClaimable) {
    uint256 usdcClaimed = userClaimedAmounts[_user];
    /// @dev VE max total supply at snapshot block in one chain is `9.148e24`.
    ///      It will never overflow when `usdcDistributedAmount < 1e52`.
    return ((usdcDistributedAmount * _veBalance) / CHAIN_TOTAL_VE)
    - usdcClaimed;
}
```

The amount claimed in `claimUsdc` is stored in a mapping keyed by the `msg.sender`. So it would not be safe for the value to be able to decrease. Increasing is safe, because it only means users may claim additional funds.

3.4. Actual contract balance (`address(this).balance`)

This value is intended to be incremented when `distribute` is called. It is never directly used, and accounting for which funds are safe to use is done using the `usdcDistributedAmount` variable.

If the owner called `emergencyWithdraw`, the contract balance can decrease without `usdcDistributedAmount` being decreased. The contract would be insolvent in this condition. However, funds would not be stuck. The owner (or otherwise someone) would need to transfer funds directly to the contract to resume operation after an emergency withdrawal, to avoid incrementing `usdcDistributedAmount`.

4. Assessment Results

During our assessment on the scoped VeDistributor contracts, there were no security vulnerabilities discovered.

4.1. 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.