Zellic

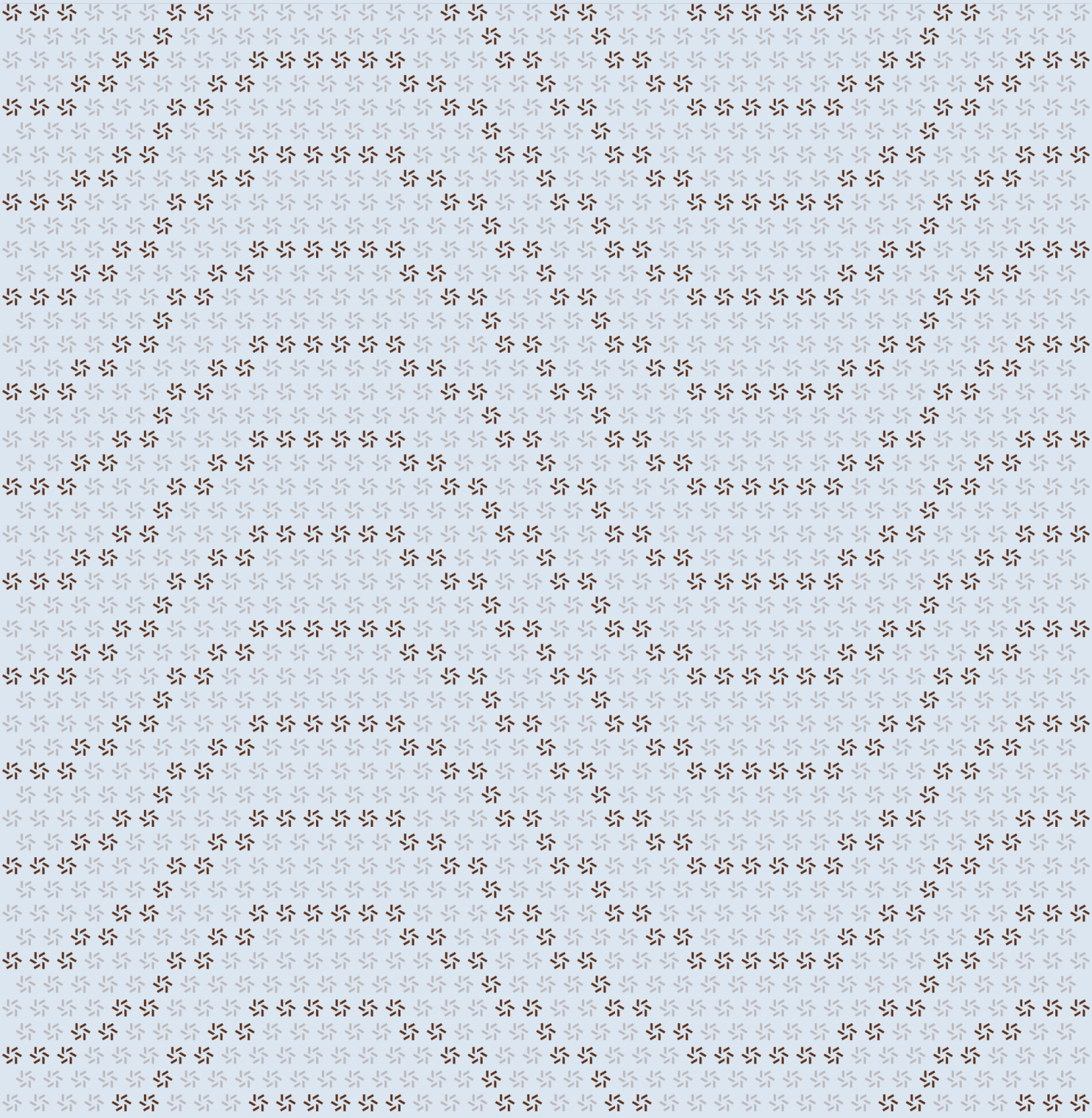
Prepared for
susruth
harsha
Garden Finance

Prepared by
Sunwoo Hwang
Varun Verma
Zellic

June 4, 2025

Garden Move Deploy

Move 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 Garden Finance from May 27th to May 28th, 2025. During this engagement, Zellic reviewed Garden Move Deploy'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:

- Is it ensured that funds move only under correct state transitions (e.g., a swap can be redeemed when `secret` hashes to `secret_hash` or refunded when `initiated_at + timelock < now`)?
- Is transfer integrity (where all coin flows are hardcoded to the initiator or redeemer addresses and no other recipient path exists) ensured?
- Is order uniqueness (the prevention of duplicate or replayed orders) ensured?
- Is it ensured that early refunds require authenticity (e.g., `instant_refund` validates an Ed25519 signature, binding the action to the rightful redeemer)?

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

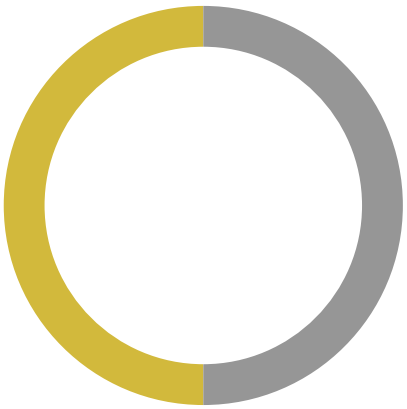
During our assessment on the scoped Garden Move Deploy modules, we discovered two findings. No critical issues were found. One finding was of medium impact and the other finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of

Garden Finance in the Discussion section (4. ↗).

Breakdown of Finding Impacts

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



2. Introduction

2.1. About Garden Move Deploy

Garden Finance contributed the following description of Garden Move Deploy:

> Garden Finance is the fastest Bitcoin bridge, enabling cross-chain Bitcoin swaps in as little as 30 seconds. It is built using an intents-based architecture with trustless settlements, ensuring zero custody risk for the users. With over a billion dollars in volume facilitated, Garden stands apart from other bridges thanks to these benefits ↗.

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 modules.

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 modules 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:

Garden Move Deploy Modules

Type	Move
Platform	Sui
Target	htlc-sui
Repository	https://github.com/catalogfi/htlc-sui ↗
Version	`14bb988d88afca5016a132a3e552f6b8302a8db1`
Programs	`sources/main.move`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2 person-days. The assessment was conducted by two consultants 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

The following consultants were engaged to conduct the assessment:

Sunwoo Hwang
Engineer
sunwoo@zellic.io

Varun Verma
Engineer
varun@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

May 22, 2025	Start of primary review period
May 23, 2025	End of primary review period

3. Detailed Findings

3.1. Duplicate-order denial of service via front-running deterministic `order_id`

Target	atomic_swapv1::AtomicSwap		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The `order_id` is calculated as `sha256(secret_hash || initiator || redeemer || timelock)`. All four inputs are present in a pending transaction's calldata, so they are visible to any mempool listener. Additionally, `initiate_on_behalf` allows *any* address to supply these same values without proving the ownership of `initiator`.

An attacker can therefore copy the four fields from a victim's pending swap and submit `initiate_on_behalf` with identical fields and `amount = 1`. If their transaction confirms first, the registry stores a dust order under that `order_id`, causing the victim's later transaction to abort with `EDuplicateOrder`.

Impact

During high-volatility windows or trading opportunities, the attacker can deny a competitor access to an arbitrage opportunity at negligible cost. A potential exploit scenario is as follows:

1. Spot a lucrative cross-chain price gap; watch the mempool for victims opening HTLCs to execute the arb.
2. Copy their four pre-image fields and front-run with `initiate_on_behalf` dust order, blocking their swap via `EDuplicateOrder`.
3. With their liquidity frozen, capture the arbitrage window yourself

Recommendations

Add an unpredictable salt (e.g., a fresh `UID` or nonce) to the `order_id` preimage.

Remediation

This issue has been acknowledged by Garden Finance, and a fix was implemented in commit `e85e06c9` ↗.

3.2. Unbounded timelock allows accidental permanent lock

Target	atomic_swapv1::AtomicSwap		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The contract enforces timelock expiry with `initiated_at + timelock < now`, which is correct. However, `timelock` itself has no maximum value, so a user could accidentally pass an extremely large number and trap funds indefinitely.

Impact

This is purely a user-experience/safety risk; user error could create an unredeemable swap.

Recommendations

Apply a sensible upper bound on `timelock` (e.g., `<= 30 days`) in `safe_params`.

Remediation

This issue has been acknowledged by Garden Finance, and a fix was implemented in commit `4b9c871f` ↗.

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. Test suite

The suite does well regarding happy-path correctness — every main flow (initiate → redeem, initiate → refund, instant refund) is exercised, and each custom abort code appears in at least one negative test.

What is missing is more of an adversarial lens. No test front-runs a pending swap with a 1-unit dust order to demonstrate the EDuplicateOrder denial of service, refund-signature replay, or extreme range timelocks (`u256::MAX`). Additionally, adding some lightweight fuzzing of `secret_hash`, amount, and timelock would move coverage from honest misuse to a more grounded real-world stress test.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the modules and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

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

5.1. Module: main.move

Function: `create_orders_registry<CoinType>(ctx: &mut TxContext): ID`

The `create_orders_registry` function creates a new shared `OrdersRegistry` of a specific coin type.

- ☑ Anyone can create a registry without restrictions.
- ☑ A new `OrdersRegistry<CoinType>` is created with a unique ID.
- ☑ The registry is shared as a global object for public access.
- ☑ The registry ID is returned for reference.

Function: `initiate<CoinType>(...)`

The `initiate` function allows users to create a new `Order`.

- ☑ The amount must be nonzero.
- ☑ The timelock must be nonzero.
- ☑ It must ensure the provided coins match the specified amount.
- ☑ It must ensure no duplicate order exists with the same order ID.
- ☑ The redeemer address is correctly derived from the provided public key.
- ☑ The initiator and redeemer must be different addresses.

Function: `initiate_on_behalf<CoinType>(...)`

The `initiate_on_behalf` function allows third parties to fund `Order` on behalf of another user.

- ☑ The amount must be nonzero.
- ☑ The timelock must be nonzero.
- ☑ It must ensure the provided coins match the specified amount.
- ☑ It must ensure no duplicate order exists with the same order ID.
- ☑ The redeemer address is correctly derived from the provided public key.
- ☑ The initiator and redeemer must be different addresses.

Function: `redeem_swap<CoinType>(...)`

The `redeem_swap` function allows the redeemer to claim tokens by providing the correct secret.

- ☑ It must ensure the order exists in the registry.
- ☑ It must ensure the order is not already fulfilled.
- ☑ The provided secret must hash to match the order's secret hash.
- ☑ The calculated order ID must match the provided order ID.
- ☑ The order is marked as fulfilled after successful redemption.
- ☑ Tokens are transferred to the redeemer address.

Function: `refund_swap<CoinType>(...)`

The `refund_swap` function allows the initiator to reclaim tokens after timelock expiration.

- ☑ It must ensure the order exists in the registry.
- ☑ It must ensure the order is not already fulfilled.
- ☑ It must ensure the timelock has expired based on current timestamp.
- ☑ The order is marked as fulfilled after successful refund.
- ☑ Tokens are transferred back to the initiator.

Function: `instant_refund<CoinType>(...)`

The `instant_refund` function allows immediate refund with the redeemer's signature authorization.

- ☑ It must ensure the order exists in the registry.
- ☑ It must ensure the order is not already fulfilled.
- ☑ If the caller is not the redeemer, it must verify the provided Ed25519 signature.
- ☑ The signature must be valid against the redeemer's public key and refund digest.
- ☑ The order is marked as fulfilled after successful refund.
- ☑ Tokens are transferred back to the initiator.

6. Assessment Results

During our assessment on the scoped Garden Move Deploy modules, we discovered two findings. No critical issues were found. One finding was of medium impact and the other finding was informational in nature.

The core swap logic is clean, and balances move only under the correct conditions, so the foundation appears solid. More stress testing against hostile behavior that randomizes `secret_hash`, amount, and timelock, and that explicitly simulates adversarial scenarios would provide additional confidence.

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.