Zellic

Prepared for
Wesley Graham
Thala Labs

Prepared by
Aaron Esau
Varun Verma
Zellic

April 1, 2025

LPT/xLPT
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 Thala Labs from March 18th to March 24th, 2025. During this engagement, Zellic reviewed LPT/xLPT'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:

- Can xLPT be manipulated to result in unfair distribution of rewards?
- Can dispatchable function logic result in malicious behavior?
- Can boosted farming logic be manipulated in an attacker's favor?
- Is xLPT always redeemable 1:1 with LPT?

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.

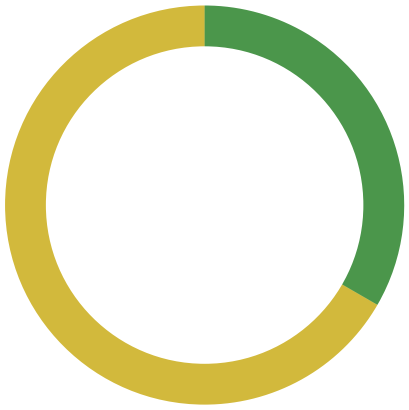
1.4. Results

During our assessment on the scoped LPT/xLPT modules, we discovered three findings. No critical issues were found. Two findings were of medium impact and one was of low impact.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Thala Labs in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	2
🟩 Low	1
⬜ Informational	0



2. Introduction

2.1. About LPT/xLPT

Thala Labs contributed the following description of LPT/xLPT:

> xLPT is Thala's evolution on staked LPT (Liquidity Pool Tokens). xLPT accomplishes four goals:
>
> 1. **Yield bearing:** Whoever holds xLPT is eligible for rewards.
> 2. **Boosted:** Whoever holds xLPT, if equipped with veTHL, is eligible for boosted rewards.
> 3. **Transferrable:** xLPT can be traded on dex and sent to other accounts.
> 4. **Composable:** xLPT can be used as collaterals in Echelon market.

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:

LPT/xLPT Modules

Type	Move
Platform	Aptos

Target	thala-modules
Repository	https://github.com/ThalaLabs/thala-modules ↗
Version	`2f4ca63654568e6cebb6dc0dc4172a130fbf7f8b`
Programs	`thala_staked_lpt/sources/*` `masterchef_lib/sources/*`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.8 person-weeks. The assessment was conducted by two consultants over the course of two calendar weeks.

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:

Aaron Esau
Engineer
aaron@zellic.io ↗

Varun Verma
Engineer
varun@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

March 18, 2025	Start of primary review period
March 20, 2025	Kick-off call
March 24, 2025	End of primary review period

3. Detailed Findings

3.1. Potential integer overflow in stable_swap invariant calculation

Target	Oracle.move		
Category	Coding Mistakes	**Severity**	High
Likelihood	Medium	**Impact**	Medium

Description

The stable_swap invariant calculation in Oracle.move involves downcasting from `u128` to `u64`, which could potentially lead to integer overflow in certain scenarios. The code performs this conversion where the result of `stable_math::compute_invariant(&balances, amp)` is cast to `u64`. This downcasting operation might fail for pools that hold extremely large token balances or use a high amplification parameter, as the computed invariant could exceed the maximum value representable by `u64`.

Impact

If the computed invariant exceeds $2^{64} - 1$, the downcasting operation will cause an overflow, which could lead to incorrect invariant calculations. This might result in improper price calculations or pool operations, potentially affecting the reliability of the oracle.

Recommendations

Consider using `u128` throughout the calculation to avoid potential overflow issues.

Remediation

This issue has been acknowledged by Thala Labs, and a fix was implemented in commit `f623d9a5` ↗.

3.2. Missing boost-update mechanism for veTHL balance changes

Target	thala_staked_lpt::staked_lpt		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

The staked_lpt module uses a boost-multiplier system where users with veTHL tokens receive amplified farming rewards. The boost calculation in the `boost_multiplier` function correctly accounts for a user's veTHL balance relative to their staked LPT position. However, there is no automatic mechanism to update a user's boost factor when their veTHL balance changes.

Currently, the `update_farming` function (which updates the boosted stake amount) is only called when a user's xLPT balance changes via the `deposit_internal` or `withdraw_internal` functions. If a user acquires additional veTHL tokens or loses some, their boost factor will not be automatically recalculated until they perform a staking/unstaking action.

This means users must manually trigger a recalculation by depositing or withdrawing tokens (potentially a zero-amount transaction) to update their boost; otherwise, they miss out on potential additional rewards.

Impact

Users who increase their veTHL holdings will not receive their deserved higher boost factor until they manually interact with the contract.

Recommendations

Implement a mechanism to periodically update boost factors for all users.

Remediation

This issue has been acknowledged by Thala Labs, and a fix was implemented in commit `f623d9a5` ↗.

3.3. Unbounded loop in `mass_update_pools` function

Target	masterchef.move		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The `mass_update_pools` function in masterchef.move contains an unbounded loop that iterates through all pools to update rewards. The function uses `vector::for_each` to iterate through all pool IDs without any limit on the number of pools processed:

```
fun mass_update_pools(farming: &mut FarmingCore, reward_id: String) {
...
   vector::for_each(simple_map::keys(pools), |pool_id| { // Unbounded iteration
       let pool = simple_map::borrow_mut(pools, &pool_id);
       accrue_pool_reward(pool, reward);
   });
}
```

This unbounded iteration could potentially cause the function to exceed gas limits if there are too many pools, affecting critical operations like `end_epoch`, which calls this function.

Impact

If the number of pools grows significantly over time, the `mass_update_pools` function might fail due to exceeding gas limits. This could prevent important protocol operations like ending reward epochs through the `end_epoch` function. However, this is unlikely to occur under normal circumstances, as it would require a very large number of pools.

Recommendations

Consider implementing pagination for pool updates to process a limited number of pools at a time.

Remediation

This issue has been acknowledged by Thala Labs.

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. Rate limiter

The rate limiter tracks withdrawals at the pool level rather than per user. This design means all users share a common withdrawal quota for each pool. When one user makes a large withdrawal, they reduce the available capacity for all other users, potentially preventing others from unstaking their tokens during high-activity periods.

Note that while this approach has the advantage of being simpler to implement and helps prevent Sybil attacks (where a single entity might create multiple accounts to circumvent limits), it can lead to less predictable user experiences since an individual's ability to withdraw depends on the collective behavior of all participants. This represents a design trade-off between system simplicity and individualized access control, with the protocol prioritizing overall pool stability over guaranteed individual withdrawal rights.

4.2. Min formula

The pricing formula using the minimum asset price `(min(asset_1_spot_price / f, asset_2_spot_price / f) * D / LPT_Supply)` is deliberately conservative for collateral valuation purposes. This approach protects the protocol by undervaluing LP tokens when used as collateral, ensuring the system remains overcollateralized. This is appropriate for risk management in lending protocols like Echelon where the tokens serve as collateral. However, this formula would not be suitable for scenarios requiring accurate market valuation, such as swap pricing or fee calculations, where using the actual market value of the LP token would be more appropriate.

4.3. Test suite

When building a complex contract ecosystem with multiple moving parts and dependencies, comprehensive testing is essential. This includes testing for both positive and negative scenarios. Positive tests should verify that each function's side effect is as expected, while negative tests should cover every revert, preferably in every logical branch.

It is important to test the invariants required for ensuring security and also verify mathematical properties from the project's specifications.

Good test coverage has multiple effects.

- It finds bugs and design flaws early (preaudit or prerelease).
- It gives insight into areas for optimization (e.g., gas cost).
- It displays code maturity.
- It bolsters customer trust in your product.
- It improves understanding of how the code functions, integrates, and operates — for developers and auditors alike.
- It increases development velocity long-term.

The last point seems contradictory, given the time investment to create and maintain tests. To expand upon that, tests help developers trust their own changes. It is difficult to know if a code refactor — or even just a small one-line fix — breaks something if there are no tests. This is especially true for new developers or those returning to the code after a prolonged absence. Tests have your back here. They are an indicator that the existing functionality most likely was not broken by your change to the code.

In our opinion, test coverage for LPT/xLPT is acceptable. All of the major internal functions are tested, with both positive and negative scenarios.

Some recommendations include the following:

- Ensure all publicly exposed functions are tested too. For example, the tests call `deposit_internal` through the public function `staked_lpt::stake_entry` but never through the `staked_lpt::deposit` function. While the internal function is well-tested, testing all functions is ideal to ensure a future refactor does not break security properties (as previously described).
- While the unit tests are high quality, consider writing more integration tests for additional confidence that the entire system works with usage similar to real-world usage.
- Consider creating fuzzing tests.

4.4. Missing balance assertion in `withdraw`

In the thala_staked_lpt::staked_lpt module, many of the functions assert the balance before using the `fungible_asset::withdraw_with_ref` function.

The `withdraw_internal` function, which uses the aforementioned function, is called by the `unstake_entry` and `withdraw` functions. While the former has a balance assertion, the latter does not.

```
public entry fun unstake_entry(account: &signer, staked_lpt_metadata:
    Object<Metadata>, staked_lpt_amount: u64) acquires Farming, Management,
    RateLimit, RateLimitWhitelist {
    let account_addr = signer::address_of(account);
    assert!(primary_fungible_store::balance(account_addr, staked_lpt_
            metadata) >= staked_lpt_amount, ERR_STAKED_LPT_USER_INSUFFICIENT_
            STAKED_LPT_BALANCE);

        // withdraw xLPT from caller
        let staked_lpt = withdraw_internal(
            primary_fungible_store::ensure_primary_store_exists(account_addr,
                staked_lpt_metadata),
            staked_lpt_amount,
            &borrow_global<Management>(object::object_address(&staked_lpt_
                metadata)).transfer_ref
        );

        // unstake xLPT into LPT
        let lpt = unstake_whitelist_override(account, staked_lpt);
        primary_fungible_store::deposit(account_addr, lpt);
    }

    // [...]

    public fun withdraw<T: key>(
        store: Object<T>,
        amount: u64,
        transfer_ref: &TransferRef,
    ): FungibleAsset acquires Farming {
        withdraw_internal(store, amount, transfer_ref)
    }
```

There is no security impact of this because the fungible_asset module checks the balance internally.

5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. Component: Thala Staked LPT (xLPT)

Description

The Thala Staked LPT module is responsible for handling the staking, unstaking, and farming rewards for liquidity pool tokens. It does the following:

- Allows users to stake their LPT tokens and receive xLPT tokens in return
- Manages the distribution of farming rewards to xLPT holders based on their holdings and veTHL boost
- Enables transferability of farming positions, a unique feature to the protocol
- Implements security features like pausing and rate limiting for protection against attacks

The system works by converting user LPT to xLPT at a 1:1 ratio through the stake function, tracking user positions and reward accruals using the MasterChef farming algorithm, applying boost multipliers based on veTHL holdings, and managing epochs and reward distributions across different pools and reward tokens.

Invariants

- The total supply of xLPT always equals the total amount of underlying LPT locked in the contract.
- A user's boosted stake amount is always greater than or equal to their actual xLPT balance.
- Rewards are distributed proportionally based on boosted stake amounts.
- Rate limits ensure maximum withdrawal amounts within time windows.
- All state-changing operations respect pause flags when active.

Test coverage

Cases covered

- Basic functionality — staking, unstaking, transfers, and reward distribution
- Security features — rate limiting, whitelisting, and pausing mechanism
- Administrative operations — reward setup and pool configuration
- Boosted farming mechanics — veTHL integration

- Edge cases — zero amounts

Cases not covered

- Automatic boost updates when veTHL balances change without xLPT movement
- Economic attack vectors against the rate limiter
- Performance testing with large numbers of users and pools
- Complex composability scenarios with other DeFi protocols

Attack surface

- Boosting mechanics can be manipulated by timing veTHL acquisitions strategically.
- Transferring tokens can be used to avoid certain checks or optimizations.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to Aptos.

During our assessment on the scoped LPT/xLPT modules, we discovered three findings. No critical issues were found. Two findings were of medium impact and one was of low impact.

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.