Zellic

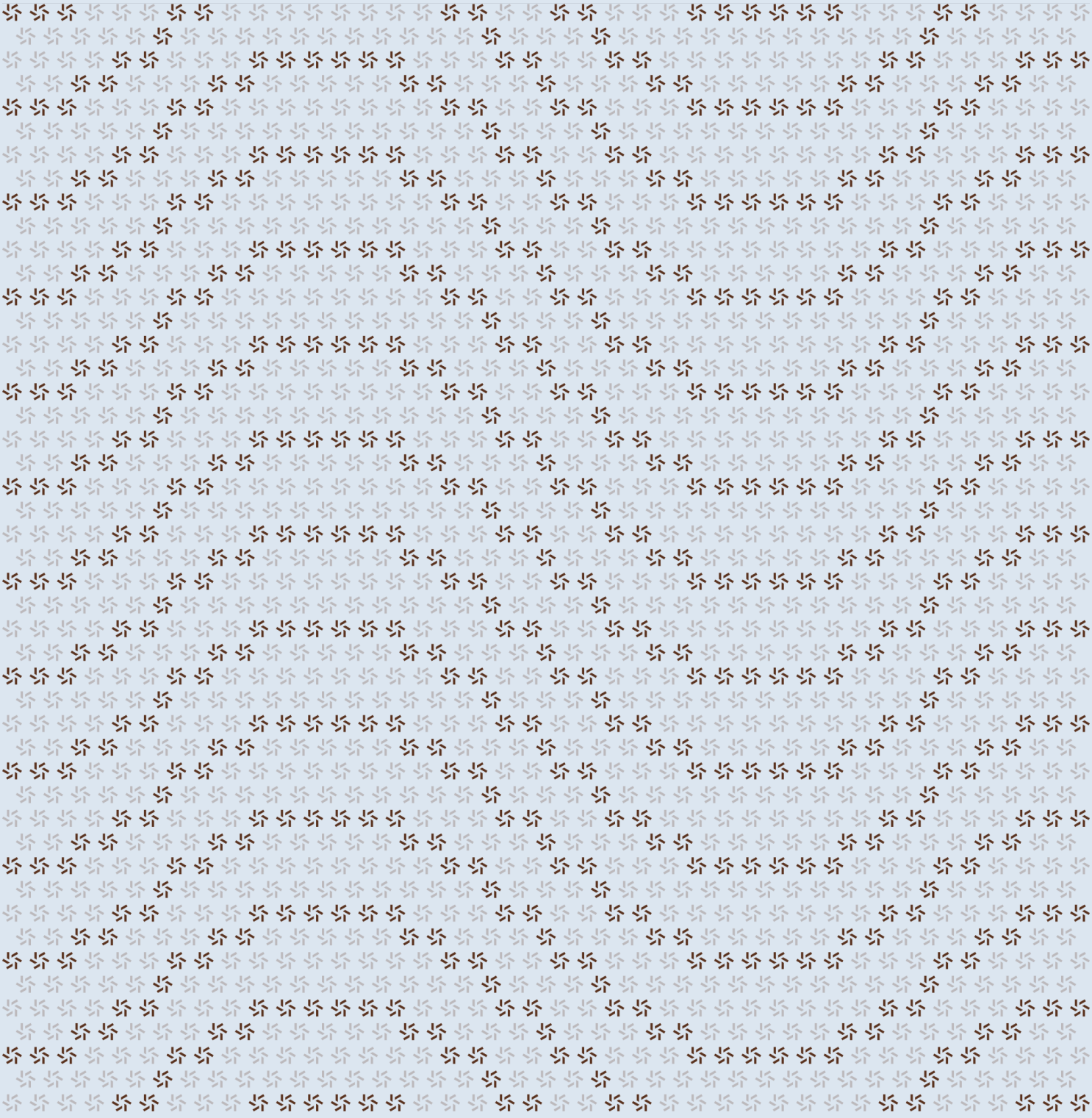
Prepared for
Henry
Cetus

Prepared by
Sunwoo Hwang
Juchang Lee
Zellic

April 11, 2025

CetusProtocol

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 Cetus from March 12th to April 8th, 2025. During this engagement, Zellic reviewed CetusProtocol'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:

- Are there any asset-related bugs that can affect the capital safety of the protocol?
- Are authorization and access properly validated?
- Can attackers impact protocol liquidity?

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

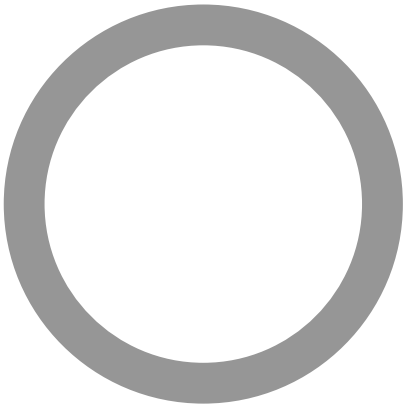
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped CetusProtocol contracts, we discovered three findings, all of which were informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	3



2. Introduction

2.1. About CetusProtocol

Cetus contributed the following description of CetusProtocol:

> Cetus is a leading decentralized exchange, which serves as the key liquidity and swap infra of the Sui ecosystem. The mission of Cetus is building a powerful and flexible underlying liquidity network to make trading easier for any users and assets. It focuses on delivering the best trading experience and superior liquidity efficiency to DeFi users through the process of building its concentrated liquidity protocol, swap aggregator and a series of affiliate interoperable functional modules.

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:

CetusProtocol Contracts

Type	Move
Platform	Sui
Target	cetus-clmm-sui
Repository	https://github.com/CetusProtocol/cetus-clmm-sui ↗
Version	`390ee5b030de44a782dc0fc6f23061611a92a64b`
Programs	`cetus_clmm/sources/**.move` `integrate/sources/**.move`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 5.7 person-weeks. The assessment was conducted by two consultants over the course of 3.8 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:

Sunwoo Hwang
Engineer
sunwoo@zellic.io

Juchang Lee
Engineer
lee@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

March 12, 2025	Kick-off call
March 12, 2025	Start of primary review period
April 8, 2025	End of primary review period

3. Detailed Findings

3.1. Incorrect access control

Target	partner.move		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

During swap or flash-loan operations, authorized partners can use the `flash_swap_with_partner` or `flash_loan_with_partner` functions to collect fees. The pool module invokes the `receive_ref_fee` function to transfer fees to the partner. However, the `receive_ref_fee` function lacks a friend modifier, allowing anyone to call it.

```
/// Receive ref fee.
/// This method is called when swap and partner is provided.
public fun receive_ref_fee<T>(
    partner: &mut Partner,
    fee: Balance<T>
) {
```

Impact

While this access-control oversight does not directly create a security issue, it does allow public deposit of partner-fee amounts, which is not necessary.

Recommendations

Add a friend modifier to the `receive_ref_fee` function.

Remediation

This issue has been acknowledged by Cetus.

3.2. Redundant functions

Target	pool.move		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `create_pool_with_liquidity` function in the factory module appears to be deprecated. Yet the following functions in integrate/pool.move still depend on it:

- `create_pool_with_liquidity_with_all`
- `create_pool_with_liquidity_only_a`
- `create_pool_with_liquidity_only_b`

Impact

Unnecessary functions could cause confusion when interacting with the pool module.

Recommendations

Remove the redundant functions.

Remediation

This issue has been acknowledged by Cetus.

3.3. Using `TypeName` for display

Target	position.move		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

When creating a position NFT, a display object is also created. The code uses property syntax to access `coin_type_[a/b]`:

```
struct Position has key, store {
    id: UID,
    pool: ID,
    index: u64,
    coin_type_a: TypeName,
    coin_type_b: TypeName,

// [...]

    let values = vector[
        utf8(b"{name}"),
        utf8(b"{coin_type_a}"), // <@
        utf8(b"{coin_type_b}"), // <@
        link,
        utf8(b"{url}"),
        description,
        website,
        creator
    ];
    let display = display::new_with_fields<Position>(
        publisher, keys, values, ctx
    );
```

The `coin_type_[a/b]` fields are defined as a `TypeName` rather than as a string. When querying objects with the `showDisplay` option, `TypeName` values appear with escape characters:

```
"coin_type_a": "\n  \u001b[1;90mtype\u001b[0m:
    0x1::type_name::TypeName\n  \u001b[1;90mname\u001b[0m:
    61e1...d59e::position::CoinTypeA"
```

Impact

Unnecessary escape characters are included in the display object.

Recommendations

We recommend using the string type for `coin_type_a` and `coin_type_b`.

Remediation

This issue has been acknowledged by Cetus.

4. 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.

4.1. Component: cetus_clmm/tick.move

Description

The tick module is responsible for managing ticks within pools using the `TickManager`. Each tick contains price information, liquidity amounts, and growth data for fees, points, and rewards. The module provides functionality to create, modify, and remove ticks as well as query tick data across specific price ranges. These ticks serve as boundaries for liquidity positions and are crucial for the concentrated-liquidity mechanism. The module tracks liquidity changes at each tick, fee accumulation, and reward distribution. When swaps cross price thresholds, the module updates liquidity and fee accounting of the ticks.

Invariants

- For any tick, `liquidity_gross` must be positive. When it reaches zero, the tick is removed from the tick list.
- Fee and reward accounting must remain accurate across all tick crossings.
- When a tick is crossed during a swap, its fee and reward-growth values are properly updated.
- The fee and reward-growth parameters cannot be manipulated directly, as access is restricted by the `friend` modifier to ensure only the pool module can call functions requiring these parameters.

Test coverage

Cases covered

- Tick initialization and default values
- Liquidity addition to ticks (`increase_liquidity`)
- Liquidity removal from ticks (`decrease_liquidity`)
- Tick crossing during swaps
- Fee, point, and reward calculation within specific tick ranges
- Tick-spacing validation
- Points calculation for reward distribution

4.2. Component: cetus_clmm/factory.move

Description

The factory module is responsible for the creation of new liquidity pools for given coin pairs. The pool manager can create pools with any specifications using the `create_pool` function as well as register permissioned pairs and manage allowed coins. Other users can create pools via the `create_pool_v2_` function — but only with a coin pair that is not in the denylist. Coin creators who have a `CreationCap` can register a base coin with allowed quote coins to prevent pool creation.

Invariants

- Each pool key (combination of base coin, quote coin, and `tick_spacing`) must remain unique.
- The initial price of a pool must be within the valid range.
- When registering a token pair, the quote coin must be included in the allowed pair configuration.
- The `CreationCap` can only register pools for their associated coin types.
- Tick spacing must exist within the designated fee-tier configuration.
- When creating a pool with the `create_pool_v2_` function, the initial liquidity added must be nonzero.
- The token pair must not be in the denylist.

Test coverage

Cases covered

- Coin allowlist and denylist management
- Correct coin-type–ordering validation
- Pool creation with `CreationCap`
- Fee-tier validation during pool creation
- Init-price validation for pool creation

4.3. Component: cetus_clmm/rewarder.move

Description

The rewarder module manages liquidity incentives in the concentrated liquidity market maker (CLMM) protocol. It handles the allocation, tracking, and distribution of rewards to liquidity providers based on their contributed liquidity. Reward growth is tracked using Q64.64 fixed-point arithmetic, accumulating proportionally to time and emission amounts per rewarder. When liquidity providers claim rewards (when their price range contains the current pool price), they receive rewards based on their share of total liquidity and withdrawal from the

`RewarderGlobalVault`. For accurate reward distribution, the protocol must maintain sufficient balances for each reward type.

Invariants

- Each reward-token type can only be registered once in a `RewarderManager`.
- The `RewarderManager` can have at most `REWARDER_NUM` (five) different reward tokens.
- When increasing emission rate, the vault must contain at least one day's worth of rewards.
- Reward-growth values can only increase, never decrease.
- Points emissions occur at a fixed rate.
- Reward withdrawals cannot exceed available balance.
- Only the protocol admin can perform emergency withdrawals.
- Reward-settlement calculations skip zero liquidity or zero time-change cases to prevent division by zero.

Test coverage

Cases covered

- Adding and registering new reward tokens
- Handling reward-token limits (max five rewards)
- Proper reward settlement with time passage
- Updating emission rates with sufficient balance validation
- Deposit and withdrawal of reward tokens
- Points-system accumulation and tracking
- Emergency withdrawal by admin
- Time validation in reward settlement

4.4. Component: cetus_clmm/pool.move

Description

The pool module is the core of the Cetus CLMM protocol. It manages token trading pairs and implements CLMMs that enhance capital efficiency beyond automated market makers (AMMs). This module adjusts liquidity and token swaps while tracking liquidity positions, price ranges, fee, and reward.

Each pool manages token trading pairs, monitors current price and liquidity, and executes key operations including position management, liquidity adjustments, token swaps, flash loans, and fee collection. This module provides comprehensive API for interaction and connects with tick, position, and rewarder components.

Invariants

- The current price must be in valid bounds defined by `min_sqrt_price()` and `max_sqrt_price()`.
- The price after swap must respect the provided price limit.
- Pool liquidity must reflect correctly the sum of all currently active liquidity positions.
- Fee and reward-growth accumulators must accurately track the global amount of fee and reward generated.
- Global fee growth and reward-growth value must be distributed to positions based on liquidity contribution.
- During a swap, all crossed ticks must be updated with the latest fee and reward-accumulator values.
- When a tick is crossed, liquidity must be added or removed from the pool's active liquidity.
- The protocol fee and partner-referral fee must be collected according to respective rates.
- Flash loans and flash swaps must be repaid with appropriate fees before the transaction end.
- Pool must not operate when paused — enforced by the `is_pause` flag.
- Position operations (open, add liquidity, remove liquidity, close) must maintain consistency between position and the pool's state.

Test coverage

Cases covered

- Pool initialization with parameters
- Opening and closing positions
- Adding and removing liquidity (both fixed liquidity and fixed coin amounts)
- Swapping with various parameters (a2b/b2a, `by_amount_in`, price limits)
- Fee calculation and collection for positions
- Reward calculation and collection
- Protocol-fee collection
- Flash loans and flash swaps
- Partner-fee handling
- Pool pausing and unpausing
- Edge cases such as swapping to price limits and tick boundaries
- Liquidity distribution across multiple positions
- Swap verification with complex tick setups
- Tick-crossing boundary handling
- Fee distribution across multiple positions
- Flash-loan execution and repayment

- Partner-based swap with referral-fee handling
- Multiposition fee collection
- Full range liquidity positions
- Swap calculations with different fee tiers

4.5. Component: cetus_clmm/position.move

Description

The position module manages liquidity positions within concentrated-liquidity pools. It implements dual-structure design with position NFTs representing ownership and `PositionInfo` containing computational data. The module provides to create, modify, close positions, and collect accumulated fees and rewards. Each position defines specific price ranges using lower- and upper-tick indexes within which the liquidity is concentrated. The position NFT can transfer between users, and it maintains its state data in the `PositionManager`.

Invariants

- Position tick ranges must be valid — `lower < upper`, both aligned to `tick_spacing`, and within min/max tick boundaries.
- A position can be closed when it has no liquidity, unclaimed fees, or unclaimed rewards.
- Fee and reward-accumulation calculation must remain accurate across all price movements.
- Position liquidity must be tracked and synchronized with the corresponding ticks.
- The growth calculation for fee and reward must use fixed-point math with 64-bit precision to ensure accuracy without overflow.
- Access to critical functions is restricted through the `friend` modifier to ensure only the pool module can call them.

Test coverage

Cases covered

- Position creation with various tick ranges
- Liquidity addition and removal operations
- Fee accumulation and collection mechanisms
- Reward distribution and claiming functionality
- Position-closure conditions
- Tick-range validation
- Growth-calculation precision for fee and reward
- Position-state queries and updates
- NFT metadata management for positions

5. Assessment Results

At the time of our assessment, the reviewed code was deployed to Sui Mainnet.

During our assessment on the scoped CetusProtocol contracts, we discovered three findings, all of which were informational in nature.

5.1. 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.