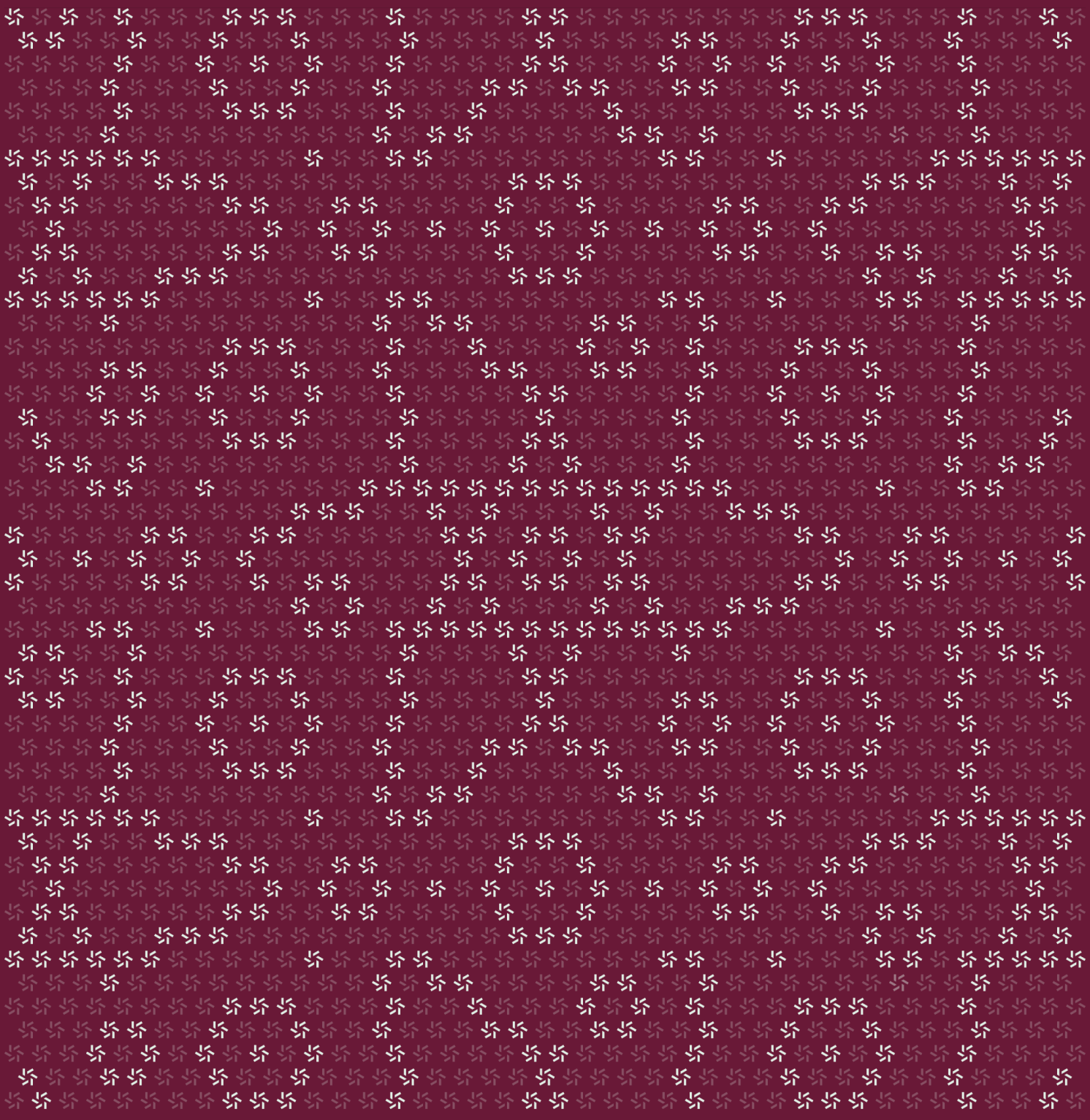
Zellic

Prepared for
Leven Wilson
Matt Ye
Lorenzo

Prepared by
Jasraj Bedi
Maik Robert
Zellic

April 30, 2024

Lorenzo Protocol
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 Lorenzo from April 8th to April 23rd, 2024. During this engagement, Zellic reviewed Lorenzo Protocol'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 the staking functionality working as intended?
- Are there any issues with the mint or burn functionality?
- Are BTC block headers synchronized and verified correctly?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Verification of test cases
- Interactions with off-chain components required for the ecosystem

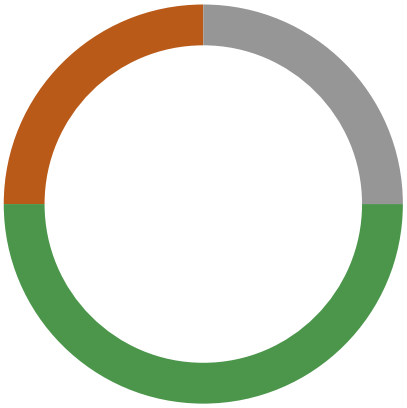
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Lorenzo Protocol modules, we discovered four findings. No critical issues were found. One finding was of high impact, two were of low impact, and the remaining finding was informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	1
Medium	0
Low	2
Informational	1



2. Introduction

2.1. About Lorenzo Protocol

Lorenzo contributed the following description of Lorenzo Protocol:

> Lorenzo has implemented a blockchain based on the Cosmos architecture that is EVM-compatible. It listens for users sending BTC to our MPC deposit address, and relayers synchronize block headers to the Lorenzo chain. It verifies users' deposit transactions using BTC tx proof. After successful verification, it mints stBTC to the user's EVM address.

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.

2.3. Scope

The engagement involved a review of the following targets:

Lorenzo Protocol Modules

Repositories	https://github.com/Lorenzo-Protocol/lorenzo ↗ https://github.com/Lorenzo-Protocol/lorenzo-relayer ↗ https://github.com/Lorenzo-Protocol/lorenzo-sdk ↗
Versions	lorenzo: `583bd99aca3ba7e1576a9868c4f311ca2cfe02e4` lorenzo-relayer: `3fddd88a18e830eb0703eadc87182056413a59ff` lorenzo-sdk: `1bc6a025aa14154e6c427d4c34f20b46f92ec95d`
Program	*.go
Type	Go
Platform	EVM-compatible

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of three and a half person-weeks. The assessment was conducted over the course of two calendar weeks.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Jasraj Bedi
Co-founder
jazzy@zellic.io ↗

Maik Robert
Engineer
maik@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

April 8, 2024	Start of primary review period
April 13, 2024	Kick-off call
April 23, 2024	End of primary review period

3. Detailed Findings

3.1. Centralization Risk (see details for mitigations)

Target	Project wide		
Category	Business Logic	**Severity**	Critical
Likelihood	Low	**Impact**	High

Description

The Lorenzo Bitcoin staking module allows users to stake their Bitcoin in return for stBTC coins, minted by the Lorenzo protocol. For this, the module implements two messages:

1. `CreateBTCStaking` - This takes in a raw BTC transaction provided by the user. The transaction hash is verified against the existing headers in the Bitcoin light client module. It checks that the transaction output is sent to an address specified by the Lorenzo node's configuration. It then proceeds to mint the same amount of stBTC to the user.
2. `Burn` - This transfers stBTC coins from the user to the module and burns them. A burn event is emitted, which can be used to verify the burning of stBTC from the user. The receiving Bitcoin address is an arbitrary wallet that is assigned when the Lorenzo protocol is initialized. The Lorenzo team has ensured that this wallet will only be accessible through a multi-sig with the Lorenzo validators as the signers. Moreover, there is an off-chain service that is responsible for returning the Bitcoin to users upon burning stBTC. This service was not a part of the audit.

Impact

We must note that this presents an important centralization risk. This is because Lorenzo protocol is not programatically bound to return the Bitcoin to users. The module itself only manages the burning/minting of stBTC. This means users are not programatically guaranteed to receive their funds back upon burning stBTC. The users are required to entrust the wallet owner with the deposited funds.

Recommendations

The users should be aware of these centralization risks. We recommend that these centralization risks be clearly documented for users so that they are aware of the extent of the owner's control over the platform. This can help users make informed decisions about their participation in the project. Additionally, clear communication about the circumstances in which the owner may exercise these powers can help build trust and transparency with users. Therefore, it is recommended to implement additional measures to mitigate these risks. This can be achieved by maintaining strong and transparent multi-signature requirements for the staking BTC wallet, and explicit documentation to

enhance user awareness and engagement.

Remediation

This issue has been acknowledged by Lorenzo.

Lorenzo provided the following response regarding their actions to mitigate the centralization risk:

> Lorenzo utilizes the Cobo digital asset custody solution, where users' staked BTC is stored in Cobo's MPC wallet. Neither Lorenzo nor Cobo can unilaterally transfer user assets. Furthermore, Lorenzo employs the TSS Node Callback Mechanism, which ensures that during user withdrawals, an open-source withdraw verifier callback is used to validate and verify the correspondence between on-chain events and user withdrawals.
>
> Lorenzo will also integrate Cubist's secure hardware, using well-defined Lorenzo safety policies within the key manager to sign transactions and verify messages. This dual verification process further enhances the security of users' custodial assets.
>
> Currently, there is no fully decentralized Bitcoin LSD protocol in the industry. Lorenzo aims to gradually achieve a decentralized LSD solution by leveraging flexible Bitcoin scripts and implementing anchoring mechanisms between the Lorenzo chain state and the Bitcoin chain.

3.2. BTC script unhandled opcode

Target	x\btcstaking\keeper\msg_server.go		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `extractPaymentToWithOpReturnId` function extracts the user's Lorenzo address from the Bitcoin transaction metadata. This is the address to which stBTC is minted. It only handles the `OP_PUSHDATA1` case when extracting the address.

```
if pkScriptLen > 1 &&
    pkScriptLen <= maxOpReturnPkScriptSize &&
    pkScript[0] == txscript.OP_RETURN {

    // if this is OP_PUSHDATA1, we need to drop first 3 bytes as those are
    related
    // to script iteslf i.e OP_RETURN + OP_PUSHDATA1 + len of bytes
    if pkScript[1] == txscript.OP_PUSHDATA1 {
        opReturnId = pkScript[3:]
    } else {
        // this should be one of OP_DATAXX opcodes we drop first 2 bytes
        opReturnId = pkScript[2:]
    }
    foundOpReturnId = true
}
```

However, it is possible for the client to use `OP_PUSHDATA2` or `OP_PUSHDATA4` opcodes when making the BTC transaction. These opcodes use two and four bytes for the data length, respectively. In this case, the extracted Lorenzo address will be incorrect.

Impact

The stBTC mint will be unsuccessful as the user address extracted is invalid. The user does not receive any stBTC despite transferring BTC to the Lorenzo wallet.

Recommendations

Handle the `OP_PUSHDATA2` and `OP_PUSHDATA4` opcodes by correctly accounting for the length bytes when extracting the user address.

Remediation

The platform only uses the `OP_PUSHBYTES` opcode to store the user address. The length is later verified in `CreateBTCStaking` to be exactly 20 bytes. The call would simply return an error if the extracted address is not 20 bytes long.

3.3. Fee amount not being burned

Target	x\btcstaking\keeper\msg_server.go		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

In the `Burn` function, a fee is added that will be deducted on top of the amount that a user wants to burn.

```
fee := sdk.NewInt64Coin(types.NativeTokenDenom,
    int64(params.BurnFeeFactor*btcFeeRate))
...
err = ms.bankKeeper.SendCoinsFromAccountToModule(ctx, signer,
    types.ModuleName, []sdk.Coin{amount.Add(fee)})
if err != nil {
    return nil, types.ErrBurn.Wrap(err.Error())
}
```

The fee is added to the total amount, which is then deducted from the user's account. Then the coins are burned, but only `amount` gets passed to the function responsible for burning the coins.

```
err = ms.bankKeeper.BurnCoins(ctx, types.ModuleName, []sdk.Coin{amount})
if err != nil {
    return nil, types.ErrBurn.Wrap(err.Error())
}
```

Impact

Not burning the fee leaves the fee amount in the pool. Over time, this may dilute the pool with unwanted stBTC.

Recommendations

Add the fees to the amount to be burned.

Remediation

This issue was acknowledged by Lorenzo and fixed in commit f0f901ec ↗.

The withdrawal fee was entirely removed. Now, no fee is charged when burning stBTC.

3.4. No genesis state validation in btcstaking

Target	x\btcstaking\types\genesis.go		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The btcstaking module does not validate the genesis state. The function that is supposed to validate the genesis state simply returns nil.

```
func (gs GenesisState) Validate() error {
    return nil
}
```

Impact

If a wrong state is ever initialized, the `Validate()` function will not catch any mistakes and simply report that everything is set up as intended.

Recommendations

Add validation code that verifies all parameters present in the genesis state.

Remediation

This issue was acknowledged by Lorenzo and fixed in commit b2f3d260 ↗.

4. 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.

4.1. Component: Lorenzo Bitcoin Light Client

Message: `InsertHeaders`

This message is only accessible by a whitelisted set of users. If the whitelist is empty, it is accessible by everyone.

The following parameter can be controlled by the calling user:

- `Headers` — This is a list of Bitcoin block headers to be added to the light client.

It is used by the relayer to add block headers to the light client. This can later be used by the staking module to verify Bitcoin transactions.

To ensure that the headers are valid, certain sanity checks are performed before they are added to the light client. These are a subset of BTC protocol rules ↗.

The first set of checks are performed in the `BtcValidation` AnteHandler. This validates the proof of work for the provided header. The other checks take place in the module itself when inserting the headers.

For a header to be inserted, the block with its previous block hash must already be present in the light client. In case of a fork, the light client state is rolled back. The state is updated with the new fork.

Message: `UpdateFeeRate`

This message is only accessible by a whitelisted set of users. If the whitelist is empty, it is accessible by everyone.

The following parameter can be controlled by the calling user:

- `FeeRate` — The updated fee rate.

This message simply updates the protocol-fee rate with the newly provided rate.

Message: `UpdateParams`

This message is only accessible by a whitelisted set of users. If the whitelist is empty, it is accessible by everyone.

The following parameter can be controlled by the calling user:

`InsertHeadersAllowList` — The list of allowed users.

This message updates the list of whitelisted users that can access any messages on this module.

4.2. Component: Lorenzo Bitcoin Staking

Message: `CreateBTCStaking`

This message can be invoked by any user.

The following parameter can be controlled by the calling user:

- `StakingTx` — This is a raw Bitcoin transaction provided by the user.

This message takes in a raw Bitcoin transaction and validates it to mint stBTC to the user.

The raw transaction provided by the user is hashed and queried against the Bitcoin light client. It is checked if a staking record already exists with this hash. In this case, the transaction is aborted to avoid double minting. It then verifies that the transaction is valid by performing a Merkle proof.

The address of the receiver is extracted from the transaction, and it is validated that it is equal to the receiving Bitcoin address set in the params. Then it extracts the BTC amount and mint address from the transaction and mints the same amount of stBTC to the mint address.

A staking record with the transaction hash gets added to the module state to prevent double minting.

Message: `Burn`

This message can be invoked by any user.

The following parameter can be controlled by the calling user:

- `BtcTargetAddress` — Address of the target BTC wallet to transfer funds.

This message transfers stBTC from the user to the module and burns them. The request amount needs to be at least `btcDustThreshold`. This is set to a constant of 546 SATS.

A burn event is emitted, which can be used to verify the burning of stBTC from the user.

4.3. Component: Lorenzo relayer

Relaying BTC block headers

The relayer began as a fork of Babylon's Vigilante. A large portion of Vigilante's original functionality has been removed; what is left is mainly the reporter, which is responsible for relaying block headers and checkpoints. The reporter itself was also stripped of some of its main components, namely the code dealing with extracting and forwarding checkpoints, which leaves the relaying of block headers. Reviewing the remaining code that handles the aforementioned relaying of block headers did not surface any issues.

5. Assessment Results

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

During our assessment on the scoped Lorenzo Protocol modules, we discovered four findings. No critical issues were found. One finding was of high impact, two were of low impact, and the remaining finding was 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.