Zellic

Prepared for
Tom Lehman
Facet

Prepared by
Jisub Kim
Jinseo Kim
Zellic

November 5, 2024

Facet Node
Comprehensive Security Assessment

Contents

About Zellic **4**

1. Overview **4**

1.1. Executive Summary 5

1.2. Goals of the Assessment 5

1.3. Non-goals and Limitations 5

1.4. Results 5

2. Introduction **6**

2.1. About Facet Node 7

2.2. Methodology 7

2.3. Scope 9

2.4. Project Overview 10

2.5. Project Timeline 10

3. Detailed Findings **10**

3.1. More fields than expected will cause an error 11

3.2. RLP-decoded element can be a list 12

4. Discussion **12**

4.1. ERC-55 validation in `validated_address` 13

4.2. IRB console is called when an error occurs 13

5. Scope extension **13**
5.1. FCT mint mechanism updates 14
5.2. Scope 14
5.3. FCT mint mechanism discussion 15

6. System Design **16**
6.1. Facet modules 17

7. Assessment Results **22**
7.1. Disclaimer 23

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 Facet from September 30th to October 7th, 2024. During this engagement, Zellic reviewed Facet Node'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:

- Could a malicious message trigger a denial of service on Facet node?
- Could a malicious user steal/mint an FCT gas token (Facet Compute Token) arbitrarily?
- Are there other ways to slow down the Facet node?

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.

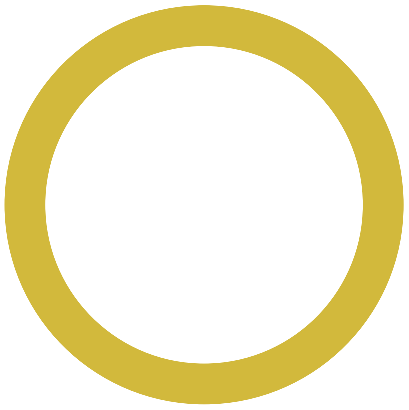
1.4. Results

During our assessment on the scoped Facet Node modules, we discovered two findings, both of which were medium impact.

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

Breakdown of Finding Impacts

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



2. Introduction

2.1. About Facet Node

Facet contributed the following description of Facet Node:

Facet Protocol is an EVM-compatible rollup that offers a novel approach to scaling Ethereum without introducing new dependencies or trust assumptions. As a fork of Optimism's OP Stack, the framework behind many of the largest Layer 2 rollups, Facet differentiates itself by eliminating all sources of centralization and privilege, resulting in the first rollup that preserves Ethereum's liveness, censorship resistance, and credible neutrality.

Facet node, the focus of this audit, is a specialized adaptation of the standard Ethereum node infrastructure, designed to facilitate the execution of Facet's off-chain compute. It is responsible for detecting Facet transactions, extracting and delivering the payloads to Facet geth for execution, and securely storing the resulting state to be queried by Facet users and dApps. Facet node also issues and consumes Facet Compute Token (FCT), the protocol's native gas token. By maintaining synchronization with Ethereum, Facet nodes ensure consistent state derivation from Ethereum's transaction history while processing and validating Facet transactions.

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.

Architecture risks. This encompasses potential hazards originating from the blueprint of a system, which involves its core validation mechanism and other architecturally significant constituents influencing the system's fundamental security attributes, presumptions, trust mode, and design.

Arithmetic issues. This includes but is not limited to integer overflows and underflows, floating-point associativity issues, loss of precision, and unfavorable integer rounding.

Implementation risks. This encompasses risks linked to translating a system's specification into practical code. Constructing a custom system involves developing intricate on-chain and off-chain elements while accommodating the idiosyncrasies and challenges presented by distinct programming languages, frameworks, and execution environments.

Availability. Denial-of-service attacks are another leading issue in custom systems. Issues including but not limited to unhandled panics, unbounded computations, and incorrect error handling can potentially lead to consensus failures.

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:

Facet Node Modules

Type	Ruby
Platform	Ruby

Target	facet-node
Repository	https://github.com/0xFacet/facet-node ↗
Version	`75877a548c3b515cb8d08e8920016dcfa2523bee`

Programs	`app/models/eth_block.rb`
	`app/models/eth_call_struct.rb`
	`app/models/eth_transaction.rb`
	`app/models/facet_block.rb`
	`app/models/facet_transaction.rb`
	`app/models/predeploy_manager.rb`
	`lib/eth_block_importer.rb`
	`lib/geth_client.rb`
	`lib/geth_driver.rb`
	`lib/l1_attributes_tx_calldata.rb`
	`lib/transaction_helper.rb`
	`lib/sys_config.rb`
	`lib/chain_id_manager.rb`
	`lib/address_alias_helper.rb`
	`lib/fct_mint_calculator.rb`

2.4. Project Overview

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

After the initial audit, the scope was expanded to include a review of the FCT mint mechanism updates, as outlined in Section 5.1. ↗.

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:

Jisub Kim
Engineer
jisub@zellic.io ↗

Jinseo Kim
Engineer
jinseo@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 30, 2024	Kick-off call
September 30, 2024	Start of primary review period
October 7, 2024	End of primary review period
November 4, 2024	Start of primary review period for FCT mint mechanism upgrade
November 7, 2024	End of primary review period for FCT mint mechanism upgrade

3. Detailed Findings

3.1. More fields than expected will cause an error

Target	facet_transaction.rb		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The `from_payload` function of FacetTransaction contains the following code:

```
# So people can add "extra data" to burn more gas
# TODO: require it to be exactly 7
unless tx.size <= 7
  raise Eth::Tx::ParameterError, "Transaction missing fields!"
end
```

It appears that this function raises an error when the transaction has more fields than expected, not when it has fewer fields.

Impact

This can lead to confusion and improper handling of transactions.

Recommendations

Consider implementing more precise validation for the sizes of TX fields.

Remediation

This issue has been acknowledged by Facet, and a fix was implemented in commit `65287c61` ↗.

3.2. RLP-decoded element can be a list

Target	facet_transaction.rb		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

In the `from_payload` function of FacetTransaction, the types of the decoded elements (i.e., `tx[i]`) should be properly checked. This is important because an RLP-decoded element can be a list as well as bytes.

Impact

This can lead to errors or crashes when the application tries to handle data in an unexpected format.

Recommendations

Consider validating that none of the Facet transaction fields RLP decode to lists.

Remediation

This issue has been acknowledged by Facet, and a fix was implemented in commit `e7f7fe86` ↗.

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. ERC-55 validation in `validated_address`

The current implementation of `validated_address(str)` uses the regex `str.match?(/\A0x[0-9a-f]{40}\z/)` to ensure that Ethereum addresses are properly formatted. This check only allows lowercase hexadecimal characters, effectively bypassing the need to handle ERC-55 checksum addresses, which include uppercase letters for validation.

The Facet team has responded that addresses are provided in raw bytes within the RLP, eliminating the necessity to consider ERC-55. The `str.bytes_to_hex` method converts these raw bytes to a lowercase hex string, ensuring consistency in address formatting.

4.2. IRB console is called when an error occurs

In the codebase, this pattern is spotted frequently:

```
rescue => e
    binding.irb
    raise
```

While this approach was invaluable during the early development stages for quickly inspecting and resolving issues, it no longer aligns with best practices as the codebase matures and the number of errors has significantly decreased.

This has been acknowledged by Facet, and they are planning to phase out these IRB statements. Instead, they will implement more structured error handling and comprehensive logging, ensuring a robust and scalable application.

5. Scope extension

5.1. FCT mint mechanism updates

The scope of this audit was extended to include a review of the changes made to the minting mechanism used for the L2-native FCT gas token. Facet native gas currency (FCT) is minted by burning ETH in the form of gas [1].

Prior to the changes being reviewed, the amount of ETH burned was converted into FCT by applying a coefficient which decreased over time (halving every N blocks). This mechanism was modified to include a dynamic adjustment mechanism for the conversion coefficient that accounts for FCT demand, by providing incentives that aim at maintaining the effective mint rate of FCT near a target level. The target mint rate starts at 40 FCT per block and is subject to periodic halving, which happens approximately once per year.

5.2. Scope

Node

At the time of this writing (November 6, 2024), the latest commit on the main branch was `64bc9f90` ↗, and the in-scope files were evaluated at that point in time.

The review was limited to the changes pertinent to the FCT minting mechanism, and focused on the following files:

- `lib/fct_mint_calculator.rb`
- `lib/l1_attributes_tx_calldata.rb`

The changes were initially introduced in PRs #17, modified by PR #19, and modified again by an individual commit. More specifically, the following commits are relevant to reconstruct the history of the in-scope files:

- For PR #17, the following commits (merged onto the main branch in commit `ed1f6e71` ↗):
  - `a79f853b` ↗
  - `a7d07936` ↗
  - `a18f6f78` ↗
- For PR #19, the following commits (merged onto the main branch in commit `05774acd` ↗):
  - `088348c4` ↗
  - `bbe54129` ↗
- The following commits also contain relevant changes:
  - `fa791b62` ↗

[1] Facet transactions can be initiated by externally owned accounts (EOAs) (by sending a transaction to the 0xface7 address) or by smart contracts (by emitting an event with a specific topic). The gas for EOA transactions is the number of nonzero bytes sent as calldata multiplied by 16, plus the number of zero bytes multiplied by 4. The gas for transactions initiated by smart contracts is the number of bytes contained in the event multiplied by 8. These coefficients match the gas cost for data usage on the EVM.

Other repositories

The review also included minor related changes to other repositories:

facet-geth

A minor modification to function `extractL1GasParamsPostEcotone` in `core/types/rollup_cost.go`, to account for additional metadata added to L1 block attributes. The change was introduced in commit `ca19563e` ↗.

facet-optimism

At the time of this writing, the latest commit on branch `facetv1.9.1` was `7bd9331b` ↗. The in-scope files were evaluated at this commit.

A minor modification to function `version` in `packages/contracts-bedrock/src/L2/L1Block.sol`, to add the FCT conversion rate and the amount of FCT minted in the current period to the data returned by the function. The change was introduced in commit `3bb8307d` ↗.

Minor modifications to `op-node/rollup/derive/l1_block_info.go`, to modify the expected length of the L1 block info data. This change was introduced in commit `3bb8307d` ↗.

5.3. FCT mint mechanism 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.

Conversion coefficient changes

The conversion coefficient is initially set to the maximum of 200,000 aFCT[2] per gas. The coefficient is adjusted every 10,000 L2 blocks; the coefficient C_{new} is computed as

$$C_{new} = \frac{M_{\text{Target}}}{G_{\text{prev. block}}}$$

where M_{Target} is the target mint rate after halving is applied, and $G_{\text{prev. block}}$ is the total amount of L1 gas spent to mint FCT in the previous 10,000 L2 blocks[3].

The new conversion coefficient is subject to a global hard cap of 1-10,000,000 aFCT per unit of gas; the ratio of the previous conversion coefficient and the new conversion coefficient is also capped, so that the coefficient can at most be halved or doubled each 10,000 L2 blocks.

After 79 years, the mint target will eventually reach zero FCT per block due to integer math used to

2 aFCT is the smallest unit of FCT, corresponding to 10^{-18} FCT

3 The amount of gas burned since the last conversion coefficient update is cached by the custom GETH and stored in a custom metadata property associated with each block. The conversion coefficient and the amount of FCT minted during the current adjustment period are accessible on the L2, by parsing the result of the L1Block precompile `version` function.

compute the target mint rate; from that point on, the conversion rate will permanently be zero.

No security issues emerged from the review of the new FCT minting mechanism.

6. 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.

6.1. Facet modules

EthTransaction

The EthTransaction module represents an Ethereum transaction and is responsible for processing Ethereum transaction data, including converting them into Facet transactions when applicable.

Function analysis

- `from_rpc_result(block_result, receipt_result)`

Constructs a list of EthTransaction instances from Ethereum RPC results by processing block data and transaction receipts.

- `facet_txs_from_rpc_results(block_results, receipt_results)`

Extracts and sorts Facet transactions from Ethereum block and receipt results.

- `to_facet_tx`

Converts the Ethereum transaction into a Facet transaction, determining whether to use the latest Facet logic or the previous implementation based on the block's timestamp and number.

- `to_facet_tx_v2`

Handles conversion of Ethereum transactions to Facet transactions according to the updated protocol logic, attempting to parse the transaction from input or events.

- `to_facet_tx_v1`

Converts the Ethereum transaction into a legacy Facet transaction using Ethscriptions if applicable.

- `to_ethscription`

Attempts to convert the Ethereum transaction into an Ethscription by parsing the input or finding relevant events.

- `facet_tx_from_input`

Constructs a Facet transaction from the transaction's input data if the `to_address` matches the Facet inbox address, which is `0x00000000000000000000000000000000000face7`.

- `try_facet_tx_from_events`

Attempts to create a Facet transaction from the transaction's logs by checking for specific Facet-related events.

- `find_ethscription_from_input`

Creates an Ethscription from the transaction's input data if the transaction meets certain criteria.

- `find_ethscription_from_events`

Searches the transaction's logs for events that signify an Ethscription creation and constructs an Ethscription from the event data.

- `create_ethscription_from_event(event)`

Constructs an Ethscription from a given event by decoding necessary data like the initial owner and content URI.

Facet notified us that they plan to remove the functions performing the migration and handling the legacy protocol from the codebase once Facet chain launches.

EthscriptionEVMConverter

The EthscriptionEVMConverter module provides functionality to convert Ethscriptions into EVM-compatible transactions, handling argument conversion, address calculations, and contract interactions.

Function analysis

- `facet_tx_to`

Determines the `to` address for the Facet transaction by calculating the appropriate address based on the operation specified in the Ethscription's content.

- `facet_tx_input`

Constructs the input data for the Facet transaction based on the Ethscription's operation and data, handling different function calls and cases.

- `is_upgrade?`

Checks if the transaction is an upgrade operation, which affects cache-clearing logic to ensure new contract implementations are used.

- `clear_caches_if_upgrade`

Clears caches if the transaction is an upgrade, ensuring that the latest contract data is utilized.

- `calculate_to_address(legacy_to)`

Calculates the new address mapping for a given legacy address, handling exceptions and creating legacy value mappings.

- `convert_args(contract, function_name, args)`

Converts function arguments to the appropriate types for ABI encoding, handling special cases like `withdrawalId` and argument normalization.

- `normalize_args(args, inputs)`

Normalizes argument values based on input types defined in the contract's ABI, ensuring type correctness.

- `normalize_arg_value(arg_value, input)`

Normalizes a single argument value according to its expected type, handling address aliasing and type conversions.

- `real_withdrawal_id(user_withdrawal_id)`

Resolves the actual withdrawal ID from a user-provided withdrawal ID, handling legacy mappings and possible exceptions.

- `is_smart_contract_on_l1?(address)`

Checks whether a given address is a smart contract on L1, which affects address aliasing and transaction processing.

- `alias_address_if_necessary(address)`

Applies address aliasing if the address corresponds to an L1 smart contract, ensuring correct address usage in transactions.

FacetBlock

The FacetBlock module represents a block in the Facet Protocol. It encapsulates the state and attributes derived from an Ethereum L1 block and includes methods for block creation, sequencing, and comparison.

Function analysis

- `from_eth_block(eth_block)`

Creates a new FacetBlock instance from an Ethereum block by extracting relevant block information.

- `attributes_tx`

Generates the L1 attributes transaction for the block, which includes essential block metadata.

- `from_rpc_response(resp)`

Parses RPC response data to populate the FacetBlock instance with block details like number, hash, and timestamps.

- `calculated_base_fee_per_gas`

Calculates the base fee per gas for the block, using the previous block's base fee if necessary.

- `compare_geth_instances(other_rpc_url, geth_rpc_url)`

Compares two Geth instances to find discrepancies in block hashes and transactions, useful for debugging and ensuring consistency.

FacetTransaction

The FacetTransaction module represents a transaction within the Facet blockchain. It includes methods to construct transactions from Ethscriptions or Ethereum transactions to generate Facet payloads and to manage transaction attributes like gas limits and fees.

Function analysis

- `from_ethscription(ethscription)`

Constructs a Facet transaction from an Ethscription, extracting necessary data like `to_address`, input, and `from_address`.

- `from_payload(l1_tx_origin, from_address, input, tx_hash, block_hash)`

Creates a Facet transaction from given payload parameters by decoding the transaction input and validating fields.

- `l1_attributes_tx_from_blocks(facet_block)`

Generates the L1 attributes transaction from the Facet block, which is used to propagate block metadata in the network.

- `compute_source_hash(payload, source_domain)`

Computes the source hash for the transaction, which is used in constructing the transaction payload and ensuring uniqueness.

- `to_facet_payload`

Encodes the transaction into the Facet payload format, including RLP encoding and applying the correct transaction type.

- `assign_gas_limit_from_tx_count_in_block(tx_count_in_block)`

Calculates and assigns the gas limit based on the number of transactions in the block, ensuring that block gas limits are respected.

- `calculated_from_address`

Computes the effective sender address, applying address aliasing if necessary based on whether the transaction was contract initiated.

- `calculate_calldata_cost(hex_string, contract_initiated)`

Calculates the gas cost of the transaction's calldata, considering whether it was initiated by a contract, which affects gas pricing.

FacetTransactionReceipt

The FacetTransactionReceipt module represents the receipt of a Facet transaction, containing information about the transaction's execution outcome. It includes methods to handle legacy contract address mappings and to process transaction traces for debugging and analysis.

Function analysis

- `set_legacy_contract_address_map`

Updates the legacy contract address map based on the transaction receipt and associated events, ensuring correct address mappings for legacy contracts.

- `update_real_withdrawal_id`

Updates the mapping for withdrawal IDs based on the transaction's logs, linking legacy IDs to new ones.

- `decoded_logs`

Decodes the logs of the transaction receipt using contract ABIs, translating raw log data into structured event information.

- `calculate_legacy_contract_address`

Calculates the expected legacy contract address based on the sender's address and nonce, used for address mapping in legacy systems.

- `trace`

Retrieves and processes the transaction trace, decoding function calls and events to provide detailed execution information.

EthBlockImporter

The EthBlockImporter module is responsible for importing Ethereum blocks and processing them to produce Facet blocks and transactions. It handles synchronization with the Ethereum network, manages block caches, and orchestrates the conversion of Ethereum data into Facet blockchain data.

Function analysis

- `import_blocks_until_done`

Continuously imports blocks until there are no more blocks ready to import, managing the import loop.

- `import_blocks(block_numbers)`

Imports a batch of Ethereum blocks, converting them into Facet blocks and processing the contained transactions, including proposing them to the Facet network.

- `populate_l1_rpc_results(block_numbers)`

Fetches Ethereum block and receipt data asynchronously for the given block numbers, preparing data for processing.

- `propose_facet_block(facet_block, facet_txs)`

Proposes a new Facet block to the Geth driver, including transactions and block headers, integrating it into the Facet blockchain.

- `next_blocks_to_import(n)`

Determines the next set of Ethereum block numbers to import based on the current state and import batch size.

- `set_eth_block_starting_points`

Determines the starting points for importing based on the current state of the Ethereum and Facet blockchains, handling cases where the Facet blockchain is empty or needs synchronization.

- `current_facet_block(type)`

Retrieves the current Facet block of the specified type (`:head`, `:safe`, `:finalized`), used for block synchronization and network consensus.

7. Assessment Results

At the time of our assessment, the reviewed code was deployed to the Ethereum Sepolia testnet.

During our assessment on the scoped Facet Node modules, we discovered two findings, both of which were medium impact.

7.1. 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.