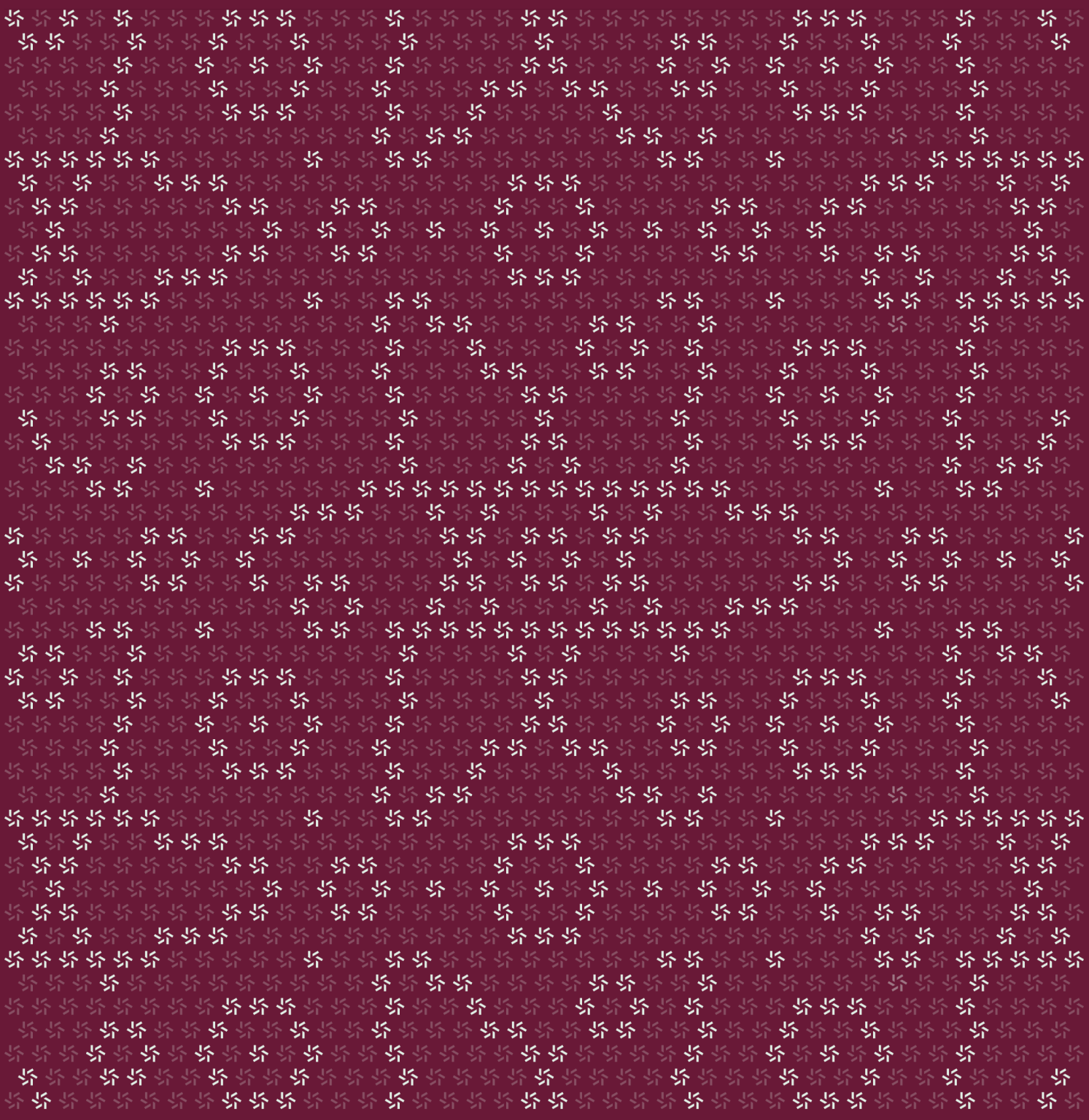
Zellic

Prepared for
Oliver Bell
Monoma Ltd.

Prepared by
Syed Faraz Abrar
Avraham Weinstock
Zellic

August 13, 2024

Packet Forward Middleware
Blockchain 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 Monoma Ltd. from July 29th to August 1st, 2024. During this engagement, Zellic reviewed Packet Forward Middleware'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 accounting around transfers correct?
- Are errors handled appropriately?
- Are there sources of nondeterminism?
- Are IBC denom paths parsed correctly?

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

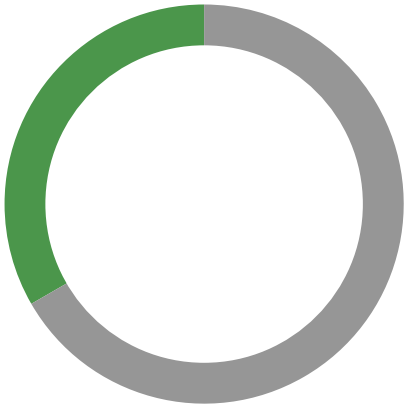
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Packet Forward Middleware modules, we discovered three findings. No critical issues were found. One finding was of low impact and the other findings were informational in nature.

Breakdown of Finding Impacts

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



2. Introduction

2.1. About Packet Forward Middleware

Monoma Ltd. contributed the following description of Packet Forward Middleware:

> The packet-forward-middleware is an IBC middleware module built for Cosmos blockchains utilizing the IBC protocol. A chain which incorporates the packet-forward-middleware is able to route incoming IBC packets from a source chain to a destination chain.

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:

Nondeterminism. Nondeterminism is a leading class of security issues on Cosmos. It can lead to consensus failure and blockchain halts. This includes but is not limited to vectors like wall-clock times, map iteration, and other sources of undefined behavior (UB) in Go.

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

Complex integration risks. Several high-profile exploits have been the result of unintended consequences when interacting with the broader ecosystem, such as via IBC (Inter-Blockchain Communication Protocol). Zellic will review the project's potential external interactions and summarize the associated risks. If applicable, we will also examine any IBC interactions against the ICS Specification Standard to look for inconsistencies, flaws, and vulnerabilities.

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:

Packet Forward Middleware Modules

Type	Golang
Platform	Cosmos
Target	ibc-apps
Repository	https://github.com/cosmos/ibc-apps ↗
Version	`26d8080d75662ea2f1155fce446ebda7a915d521`
Programs	`PacketForwardMiddleware`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.1 person-weeks. The assessment was conducted by two consultants over the course of four calendar days.

Contact Information

The following project manager was associated with the engagement:

Jacob Goreski
Jr. Engagement Manager
jacob@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Syed Faraz Abrar
Engineer
faith@zellic.io ↗

Avraham Weinstock
Engineer
avi@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

July 29, 2024	Kick-off call
July 29, 2024	Start of primary review period
August 1, 2024	End of primary review period

3. Detailed Findings

3.1. There is no upper limit to the time-out on PFM packets

Target	ibc_middleware.go		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

In the `OnRecvPacket()` function of the IBC middleware, there are checks to ensure that the time-out on the packet cannot be negative. However, there is not an upper limit set on the time-out.

Since there is no way to cancel a packet once it is sent, and because the maximum amount of retries possible is 255, it is possible for a packet to get stuck in a time-out for an extremely long time. The time-out uses a `time.Duration` type, which is an `int64`. The highest possible `int64` is a very large number.

Impact

This issue requires an external issue to trigger, because time-outs do not occur normally. This can occur due to an external bug on a chain or due to external issues with relayers. Because of this, the likelihood is low. However, because a user can lose access to their funds for a long time, the severity is medium. This leads to a low impact.

Recommendations

Add a check that ensures the time-out does not exceed an upper bound.

Remediation

This issue has been acknowledged by Monoma Ltd..

Monoma Ltd. provided the following response:

> Although it is possible that a MsgRecvPacket may not be possible to be relayed, whether due to a chain issue or a relayer issue, we have discussed internally and have determined that the likelihood is minimal and likely a concern of the chain or the relayer implementations. We don't think this is a concern that the packet-forward-middleware should try to account for by creating an upper bound on the timeout value.

3.2. Missing prefix for `RefundPacketKey`

Target	ibc_middleware.go		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Informational

Description

The `RefundPacketKey` function does not include a prefix that disambiguates it from other possible store paths.

Impact

While there are currently no other slash-separated store paths in Packet Forward Middleware's `Keeper` than `RefundPacketKey`, if one is added in the future, values stored under `RefundPacketKey` may collide with it.

Recommendations

We suggest adding a prefix that disambiguates `RefundPacketKey`.

```
 func RefundPacketKey(channelID, portID string, sequence uint64) []byte {
- return []byte(fmt.Sprintf("%s/%s/%d", channelID, portID, sequence))
+ return []byte(fmt.Sprintf("refund_packet/%s/%s/%d", channelID, portID,
    sequence))
 }
```

Remediation

This issue has been acknowledged by Monoma Ltd..

Tracked by https://github.com/cosmos/ibc-apps/issues/217 ↗.

3.3. The `processed` flag check may lead to bugs in the future

Target	ibc_middleware.go		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	Medium	**Impact**	Informational

Description

In the `OnRecvPacket()` function of the IBC middleware, there is a check that ensures that this packet has not already been processed by another middleware:

```
// if this packet has been handled by another middleware in the stack there may
    be no need to call into the
// underlying app, otherwise the transfer module's OnRecvPacket callback could
    be invoked more than once
// which would mint/burn vouchers more than once
if !processed {
    if err := im.receiveFunds(ctx, packet, data, overrideReceiver, relayer);
    err != nil {
        logger.Error("packetForwardMiddleware OnRecvPacket error receiving
    packet", "error", err)
        return newErrorAcknowledgement(fmt.Errorf("error receiving packet:
    %w", err))
    }
}
```

Currently, there are no other middlewares that can set this flag to `true` (it is by default set to `false`). Therefore, this check will always pass, and `im.receiveFunds()` will always be called, which will mean the `overrideReceiver` is always minted the vouchers that were transferred over IBC.

However, if, in the future, some middleware were to set this `processed` flag but not actually mint vouchers to the receiver, then it would lead to loss of funds for the user.

Impact

Currently, the severity and impact are both informational because this bug is only possible in the future, assuming that other middlewares can set this `processed` flag. However, the future impact and severity can be critical, because a user can lose their funds (due to no vouchers being minted).

Recommendations

Document this check extensively, preferably in the code itself as a comment. This will make sure developers are aware of this and do not introduce this issue in the future.

Remediation

This issue has been acknowledged by Monoma Ltd., and a fix was implemented in commit `8cb681e3` ↗.

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. Type: `IBCMiddleware`

The `IBCMiddleware` type implements the `IBCModule` and `ICS4Wrapper` interfaces, and it has nontrivial implementations of `{OnRecvPacket, OnAcknowledgementPacket, OnTimeoutPacket}`. All the other methods for the `IBCModule` interface pass the arguments unmodified to the middleware's `app`'s corresponding methods, and all the methods for the `ICS4Wrapper` interface pass the arguments unmodified to the middleware's `keeper`'s corresponding methods.

Function: `OnRecvPacket`

The `OnRecvPacket` function does the following:

- It passes packets of types other than `FungibleTokenPacketData` up the module stack.
- If the `FungibleTokenPacketData`'s `memo` field is not JSON or does not have a "forward" key, it passes it up the module stack.
- If the `memo` field does not unmarshal as a `PacketMetadata`, it acknowledges the packet with an error.
- If the metadata specifies an empty receiver, invalid port, or invalid channel, it acknowledges the packet with an error.
- It computes the temporary address `Bech32(SHA256(SHA256("packetforwardmiddleware") || packet.DestinationChannel || "/" || data.Sender))`.
- If `ProcessedKey` is unset or set to false in the context (see Finding 3.3. ↗), `receiveFunds` receives the transfer by passing a modified packet (with the replaced recipient and an empty memo) to receive the funds to the temporary address on the current chain; if an error occurs in receiving the transfer (including a deferred acknowledgment), it acknowledges the original packet with an error.
- If `DisableDenomCompositionKey` is unset or set to false in the context, it removes the counterparty's path prefix from the denom and converts it to a native denom if no path remains.
- If the metadata specifies a nonpositive time-out duration, it uses the `IBCMiddleware`'s `forwardTimeout` as a default.
- If the metadata does not specify an amount of retries, it uses the `IBCMiddleware`'s `retriesOnTimeout` as a default.
- It invokes the `Keeper.ForwardTransferPacket` function to forward the packet with the above modifications to the next chain; if an error occurs in forwarding the packet, it acknowledges the original packet with an error.

- It does not synchronously acknowledge the packet; it defers acknowledgment to `OnAcknowledgementPacket`/`OnTimeoutPacket` for the forwarded packet.

Function: `OnAcknowledgementPacket`

The `OnAcknowledgementPacket` function does the following:

- It passes acknowledgments for packets of types other than `FungibleTokenPacketData` up the module stack.
- If the acknowledgment data does not unmarshal as an `Acknowledgement`, it returns an error.
- It calls `Keeper.GetAndClearInFlightPacket` to get the packet `InFlightData` and number of remaining retries.
- If the `InFlightData` is present, it calls and returns the result of `Keeper.WriteAcknowledgementForForwardedPacket`.
- If the `InFlightData` is absent, it passes the acknowledgment up the module stack.

Function: `OnTimeoutPacket`

The `OnTimeoutPacket` function does the following:

- It passes time-outs for packets of types other than `FungibleTokenPacketData` up the module stack.
- It retrieves the `InFlightData` via `Keeper.TimeoutShouldRetry`.
- If the `InFlightData` is out of retries, it removes it via `RemoveInFlightPacket` and notifies the previous hop of the error with `Keeper.WriteAcknowledgementForForwardedPacket`.
- If the `InFlightData` has remaining retries, it handles the underlying time-out on the current chain and then calls `Keeper.RetryTimeout`.
- If there is no `InFlightData`, it passes the time-out up the module stack.

4.2. Type: `Keeper`

The `Keeper` type implements the `ICS4Wrapper` interface but passes arguments for the corresponding methods to the underlying `ics4Wrapper` provided during construction. The `Keeper` stores a reference to a `TransferKeeper`, which allows submitting ICS-20 transfer messages; a `ChannelKeeper` for communicating over channels; and a `BankKeeper` to handle manual balance updates.

Function: `ForwardTransferPacket`

The `ForwardTransferPacket` function does the following:

- It calls `TransferKeeper.Transfer` to transfer the funds (either to the receiver if there is no next hop, or to the next hop, with the remaining forwarding metadata).

- It decrements the `InFlightPacket`'s remaining retries, if this is not the first forwarding attempt.
- It stores `InFlightPacket` data with the number of remaining retries under `RefundPacketKey` in the `Keeper`'s store.

Function: `GetAndClearInFlightPacket`

The `GetAndClearInFlightPacket` function retrieves `InFlightPacket` data for the specified channel ID, port ID, and sequence number in the store under the `RefundPacketKey` if present and deletes it, returning `nil` if the key was absent from the store and panicking if data is present in the store but fails to unmarshal.

Function: `WriteAcknowledgementForForwardedPacket`

The `WriteAcknowledgementForForwardedPacket` function does the following:

- It looks up the channel corresponding to the `InFlightPacket`'s `(RefundPortId, RefundChannelId)` tuple.
- If the acknowledgment from the next hop is successful, it forwards the acknowledgment to the previous hop via the underlying `ics4Wrapper`.
- If the packet is nonrefundable (which other middleware can mark by setting `NonrefundableKey` to `true` in the context), it calls `Keeper.moveFundsToUserRecoverableAccount` and propagates the success/failure of that in the acknowledgment.
- It does the following if the tokens in the acknowledgment are issued by a previous hop.
    - If the tokens to be refunded are issued by a forward hop, it moves tokens from the forward escrow to the previous escrow.
    - If the tokens to be refunded are not issued by a forward hop, it burns tokens from the forward escrow.
    - It decreases the total escrow balance (via `unescrowToken`).
- It does the following if the tokens in the acknowledgment are not issued by a previous hop.
    - It mints tokens to the forward escrow.
    - It increases the total escrow balance (inlined, but symmetric logic to `unescrowToken`).

Function: `moveFundsToUserRecoverableAccount`

The `moveFundsToUserRecoverableAccount` function does the following:

- It determines an address on the current chain that the original sender can operate via `userRecoverableAccount`.
- If the tokens are not issued by a previous hop, it mints them and transfers them to the user's account on the current chain.

- If the tokens are issued by a previous hop, it transfers them from the escrow account associated with the previous hop to the user's account on the current chain and updates the escrow balance via `unescrowToken`.

Function: `TimeoutShouldRetry`

The `TimeoutShouldRetry` function retrieves `InFlightPacket` data for the specified channel ID, port ID, and sequence number in the store under the `RefundPacketKey`. It returns `nil` if the key was absent from the store, panics if the data is present from the store but fails to unmarshal, returns both the `InFlightPacket` and an error if it has a nonpositive amount of retries remaining, and returns the `InFlightPacket` with no error otherwise.

Function: `RetryTimeout`

The `RetryTimeout` function populates the `metadata.Next` field from the packet's `memo` field (returning an error if it did not contain valid JSON), validates that the packet's `Amount` parses (returning an error if not), and calls `Keeper.ForwardTransferPacket` to attempt to forward the packet again.

5. Assessment Results

At the time of our assessment, the reviewed code was deployed to Cosmos Hub and several other chains ↗.

During our assessment on the scoped Packet Forward Middleware modules, we discovered three findings. No critical issues were found. One finding was of low impact and the other findings 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.