Zellic

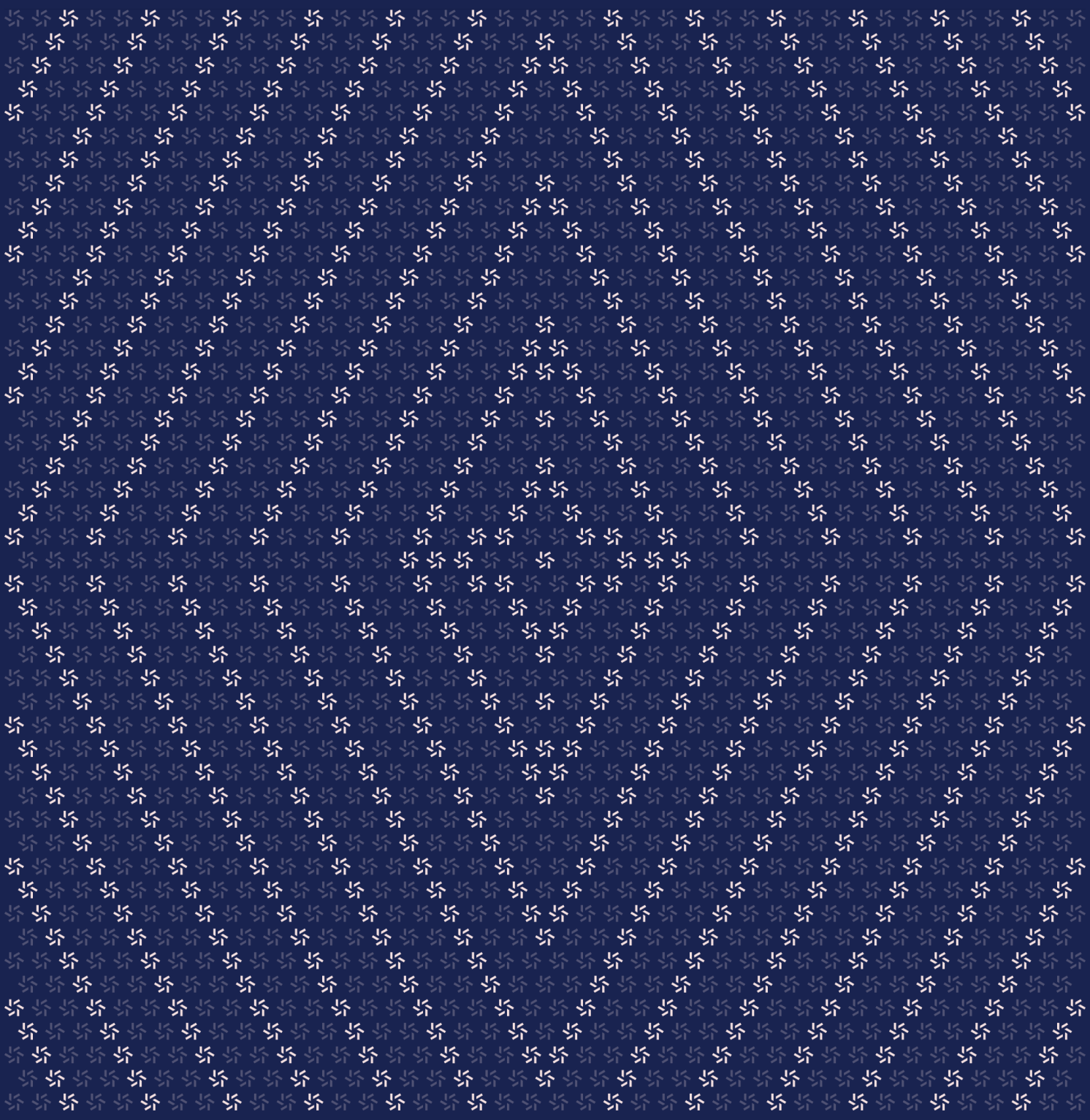
Prepared for
kled
eBridge Exchange

Prepared by
Jasraj Bedi
Kuilin Li
Zellic

June 13, 2024

eBridge AElf Bridge

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 eBridge Exchange from May 14th to June 7th, 2024. During this engagement, Zellic reviewed eBridge AElf Bridge'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:

- How are the cross-chain transactions handled within the AElf ecosystem?
- Where are the actual assets stored during the cross-chain transactions?
- Which contracts handle the actual transfers of assets?
- How is the accounting performed for the incoming and outgoing assets?
- Are there any potential ways to lock the assets in the system?

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 eBridge AElf Bridge contracts, we discovered 10 findings. One critical issue was found. Four were of high impact, one was of medium impact, and four were of low impact.

Additionally, Zellic recorded its notes and observations from the assessment for eBridge Exchange's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	1
High	4
Medium	1
Low	4
Informational	0

2. Introduction

2.1. About eBridge AElf Bridge

eBridge Exchange contributed the following description of eBridge AElf Bridge:

> eBridge is a community-driven organization and is the first user-oriented decentralized cross-chain bridge in the aelf ecosystem.

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.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts 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:

eBridge AElf Bridge Contracts

Type	C#
Platform	AElf

Target: ebridge-contracts.aelf

Repository	https://github.com/eBridgeCrosschain/ebridge-contracts.aelf ↗
Version	`68be67c8e66513dfde7d4007fdcdf7a6ce1f5529`

Programs	`EBridge.Contracts.Bridge`
	`EBridge.Contracts.MerkleTree`
	`EBridge.Contracts.Oracle`
	`EBridge.Contracts.Regiment`
	`EBridge.Contracts.Report`
	`EBridge.Contracts.StringAggregator`

2.4. Project Overview

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

Kuilin Li
Engineer
kuilin@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

May 14, 2024	Kick-off call
May 14, 2024	Start of primary review period
June 7, 2024	End of primary review period

3. Detailed Findings

3.1. Unchecked sender in `QueryOracle` function leading to potential oracle manipulation

Target	BridgeContract		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

In the BridgeContract's flow from AElf to another chain, the BridgeContract calls `ReportContract.QueryOracle` after receiving payment. However, the ReportContract does not verify the sender before eventually emitting the `ReportProposed` event. This absence of sender verification creates an opportunity for an attacker to directly call `ReportContract.QueryOracle` and trigger the `ReportProposed` event without locking tokens.

Additionally, the OracleContract calls `FulfillQuery` when executing the callback, where `OracleNodes` is the designated node list. An attacker can call `OracleContract.Query` with any node list, including those they control, as neither BridgeContract nor ReportContract check the `OracleNodes` in `FulfillQuery`. This allows attackers to manipulate the oracle responses and cause oracle clients to sign and transmit data for tokens that were never locked.

Impact

The primary impact is the potential for fraudulent oracle data transmission. Attackers can exploit this flaw to generate false reports, causing oracle clients to sign and transmit data for tokens that were never locked.

Recommendations

The primary fix for this issue would be the introduction of access control to ensure the caller of `ReportContract.QueryOracle` is the BridgeContract.

Remediation

eBridge Exchange addressed this issue with a change to ebridge-oracle in 37108e20 ↗

3.2. Invalid signature handling in `ConfirmReport` function

Target	ReportContract		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

In the ReportContract, the `ConfirmReport` function is supposed to validate and confirm reports. However, it currently reverts the transaction on the `IsRejected` assertion without invalidating the signature in any way. This means that even if a report is correctly rejected by the system, the signatures themselves remain valid and visible on chain. Even if the transactions they were sent in were rolled back, an attacker would be able to acquire valid signatures for the report. This could lead to an attacker forwarding the signatures along themselves and validating a rejected report.

Impact

The primary impact of this vulnerability is the potential for fraudulent oracle data transmission and incorrect report confirmations.

Recommendations

It is recommended to consider an alternative design where signatures are only sent after enough parties have committed to the legitimacy of the report. Alternative design decisions would be valid here as well.

Remediation

This issue has been acknowledged by eBridge Exchange, and a fix was implemented in commit `50da9c54` ↗.

3.3. Unauthorized report rejection by `SkipList` regiment members

Target	ReportContract		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

In the ReportContract, the `RejectReport` function allows members of the skip-list regiment to reject reports even though they should not have this capability. The skip list is intended to prevent certain members from confirming reports, but it does not prevent them from rejecting them. This oversight enables unauthorized members to influence the report-validation process by rejecting reports they are not supposed to interact with.

Impact

Allowing skip-list members to reject reports undermines the integrity of the reporting process. This could lead to the malicious or erroneous rejection of valid reports, resulting in incorrect data being propagated through the system.

Recommendations

Consider adding additional checks to the `RejectReport` function to ensure that only authorized members can reject reports.

Remediation

This issue has been acknowledged by eBridge Exchange, and a fix was implemented in commit 50da9c54 ↗.

3.4. Discrepancy in ReportContract `SkipList` implementation

Target	BridgeContract		
Category	Coding Mistakes	**Severity**	High
Likelihood	Medium	**Impact**	High

Description

If the `SkipList` is managed unanimously rather than through a threshold mechanism, there is a potential discrepancy between the AElf and Ethereum implementations. On Ethereum, the skip list is threshold-based, meaning a certain number of signatures are required to confirm a report. However, on AElf, if the skip list requires unanimous consensus, this could lead to inconsistencies, particularly if the last signer is on the skip list. They would be unable to confirm the report on AElf but could add their signature on the Ethereum side, causing a transmit even though they are skipped.

Impact

The discrepancy in skip-list handling between AElf and Ethereum could lead to inconsistent report confirmations, potentially causing disputes and improper validation of reports across chains.

Recommendations

Revisit the design of the signature-based validation mechanism for reports.

Remediation

This issue has been acknowledged by eBridge Exchange, and a fix was implemented in commit `c21c9f2f` ↗.

3.5. Removing a regiment member does not remove member from `ConfirmedNodeList`

Target	ReportContract		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

When a regiment member is removed during report confirmation, they remain on the `ConfirmedNodeList`. This ensures that even when a member is no longer part of the regiment, their confirmation is still considered valid, which may be undesirable if the removal was for malicious reasons.

Impact

Retaining removed regiment members on the `ConfirmedNodeList` can lead to unauthorized confirmations, where reports are validated by members who are no longer authorized, thereby undermining the integrity of the report-validation process.

Recommendations

Consider if ex-regiment members should be removed from any `ConfirmedNodeLists` or if they should at least be filtered from the list of confirmations when determining quorum.

Remediation

This issue has been acknowledged by eBridge Exchange, and a fix was implemented in commit `c21c9f2f`.

3.6. Bypassable oracle fee via manipulated payment input to `QueryOracle`

Target	ReportContract		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

In the ReportContract, the `QueryOracle` function is designed to handle oracle queries and collect oracle fees. However, it has been identified that the oracle fee can be skipped entirely by passing a negative value for `input.Payment` that equals the fee amount. This manipulation allows an attacker to bypass the fee payment, which is critical for the fair use and operation of the oracle service.

The `PayToTheContract` function, which handles the payment, does not currently check for negative values, enabling this exploit.

```
private void PayToTheContract(long payment)
{
    var totalPayment = State.ReportFee.Value.Add(payment);
    if (totalPayment > 0)
    {
        State.TokenContract.TransferFrom.Send(new TransferFromInput
        {
            From = Context.Sender,
            To = Context.Self,
            Symbol = State.OracleTokenSymbol.Value,
            Amount = totalPayment
        });
    }
}
```

Impact

The primary impact of this issue is financial. By bypassing the oracle-fee payment, malicious actors can utilize the oracle service without compensating the service providers.

Recommendations

Ensure that the `input.Payment` value is positive before processing the payment. This can be done by adding an assertion to check the payment amount.

Remediation

This issue has been acknowledged by eBridge Exchange, and a fix was implemented in commit `c21c9f2f`.

3.7. Lack of two-step or null check in critical functions

Target	BridgeContract		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The BridgeContract currently lacks a two-step process or null check in several critical functions, specifically `ChangeController`, `ChangeAdmin`, and `ChangePauseController`. Introducing a two-step process or at least a null check in these functions would improve the safety of interfacing with these functions and mitigate against misconfiguration.

Impact

Without a two-step process or null check, there is a risk of unauthorized or erroneous updates to critical contract roles, which could result in denial of service or privilege escalation.

Recommendations

Implement both fixes if possible, but either or should be sufficient to reduce operational risk.

1. Implement a two-step process:

   Introduce a two-step process for changing critical roles. This involves first proposing the change and then confirming it in a subsequent transaction. This provides an opportunity to validate the change and prevent accidental or malicious updates.

2. Add null checks:

   Add null checks to ensure that the new addresses being set are valid and not null. This can prevent the contract from entering an invalid state.

Remediation

This issue has been acknowledged by eBridge Exchange, and a fix was implemented in commit `c21c9f2f` ↗.

3.8. Missing sender check in `Initialize` function of RegimentContract

Target	RegimentContract		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

In the RegimentContract, the `Initialize` function does not include a sender check to verify that the caller has the appropriate permissions to initialize the contract. This check is performed by the OracleContract during its initialization, but it should not be reliant on another contract for such a fundamental security check.

Impact

The lack of a sender check in the `Initialize` function can lead to unauthorized initialization of the RegimentContract. This could allow an attacker to initialize the contract with their own configuration.

Recommendations

Add a sender check to the `Initialize` function in the RegimentContract contract to ensure that only authorized users can initialize the contract.

Remediation

This issue has been acknowledged by eBridge Exchange, and a fix was implemented in commit `c21c9f2f`.

3.9. Missing approval for oracle token transfers in `ChangeOracleContractAddress`

Target	ReportContract		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

In the ReportContract, the `ChangeOracleContractAddress` function allows changing the oracle contract address. However, it does not reinitialize the allowance for the new oracle contract, which means that any subsequent token transfers to the new oracle will fail. The `Initialize` function sets up the initial approval for oracle token transfers, but this needs to be replicated in `ChangeOracleContractAddress` to ensure continuity of payments.

Impact

Without reinitializing the token allowance, the new oracle contract will not be able to receive token payments, leading to failed transactions and disruption of oracle-related operations. This could cause significant issues in the system's functionality and reliability.

Recommendations

It is recommended to reinitialize the token allowance for the new oracle contract in the `ChangeOracleContractAddress` function to ensure that the new oracle can receive token payments without any issues.

Remediation

This issue has been acknowledged by eBridge Exchange, and a fix was implemented in commit `c21c9f2f` ↗.

3.10. Enforce maximum for `SetFeeFloatingRatio` and `SetPriceRatio`

Target	BridgeContract		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

In the BridgeContract, the functions `SetFeeFloatingRatio` and `SetPriceRatio` allow setting values for fee floating ratios and price ratios, respectively. However, there is no enforced upper limit on these values, which could result in setting excessively high ratios that may disrupt the system's economic balance. Setting a maximum limit of 100 for these ratios can prevent such disruptions.

Impact

Allowing excessively high values for fee floating ratios and price ratios can lead to unintended economic consequences, such as exorbitant fees or unrealistic price ratios. This can also result in denial of service.

Recommendations

Include a check in these functions to ensure that the maximum value is 100 or some other upper ceiling meant to represent 100%.

Remediation

This issue has been acknowledged by eBridge Exchange, and a fix was implemented in commit `c21c9f2f` ↗.. This addressed the issue for `SetFeeFloatingRatio` but not `SetPriceRatio`.

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. Calling `GetTokenInfo` returns null unexpectedly

The BridgeContract contract, specifically in `BridgeContract_LockToken`, calls `GetTokenInfo` in `AddToken` to retrieve information about the provided token symbol. In the case that the token does not exist, `null` is returned. The following assertion checks if the symbol on the returned `tokenInfo` is `null` or an empty string. If so, a friendly error message is emitted and the transaction is reverted.

There is a problem with this logic as-written, though. The `tokenInfo` itself will be `null` when the token does not exist. This means that the assertion will not even have a chance to execute as the field access of `tokenInfo` will yield an exception.

This is not, as written, a security issue, but it may be a usability issue that eBridge Exchange may want to address.

5. Assessment Results

At the time of our assessment, the reviewed code was partly deployed to the Ethereum Mainnet and the AElf mainnet.

During our assessment on the scoped eBridge AElf Bridge contracts, we discovered 10 findings. One critical issue was found. Four were of high impact, one was of medium impact, and four were of low impact.

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.