Zellic

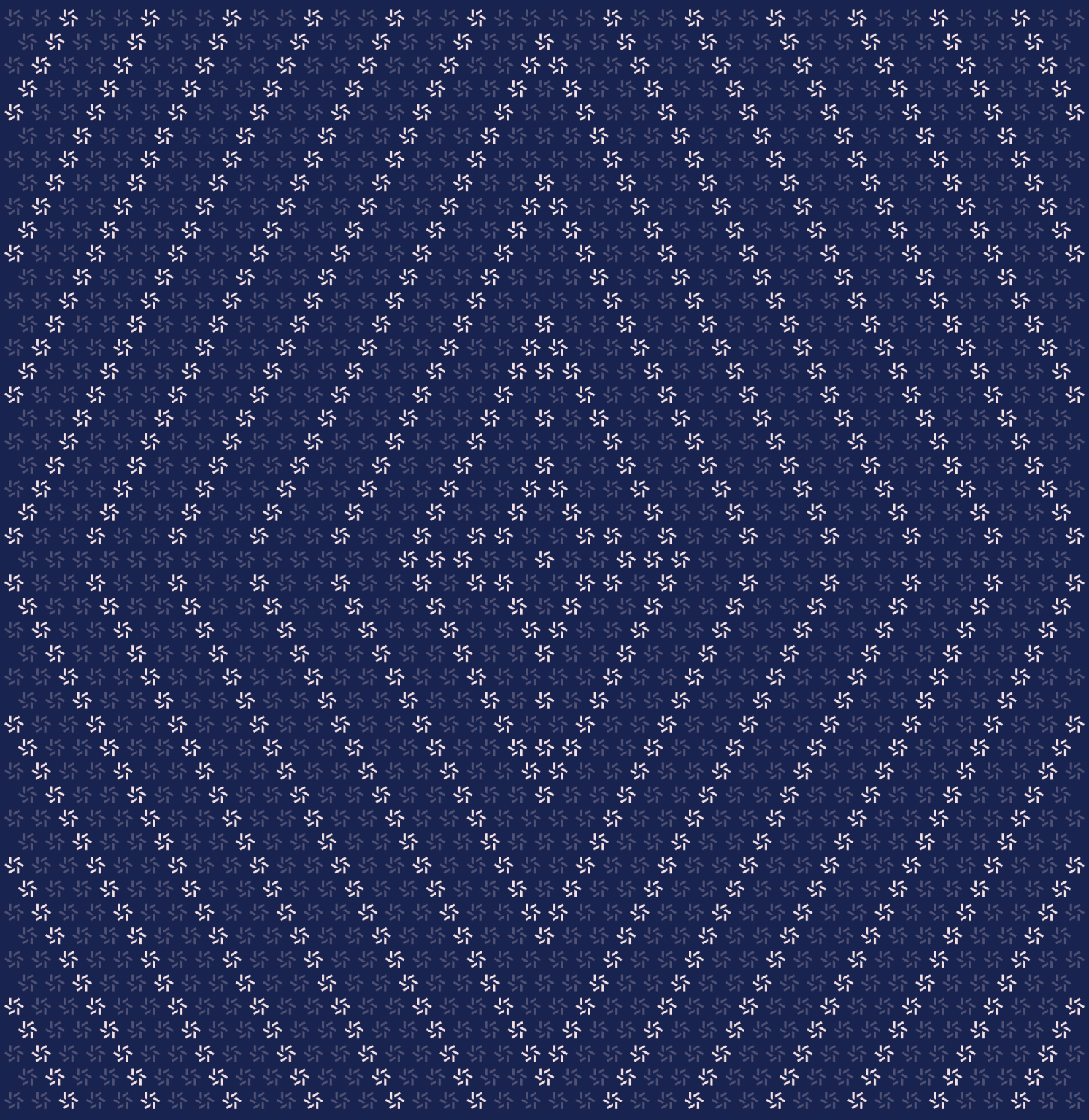
Prepared for
Trillion
Trillion

Prepared by
Katerina Belotskaia
Dimitri Kamenski
Zellic

November 29, 2024

Trillion EVM cross chain contract

Smart Contract 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 Trillion EVM cross chain contract 7

2.2. Methodology 7

2.3. Scope 9

2.4. Project Overview 9

2.5. Project Timeline 10

3. Detailed Findings **10**

3.1. Message Nonce uniqueness lacks guarantees 11

3.2. Fee lacks upper bound restrictions 12

3.3. Lack of verification on burn tokens 13

3.4. Unused functionality should be removed 14

4. Threat Model **14**

4.1. Module: NonceManager.sol 15

4.2. Module: TokenBurner.sol 15

4.3. Module: TokenMessenger.sol 16

5. Assessment Results 19

5.1. Disclaimer 20

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 Trillion from November 25th to November 27th. During this engagement, Zellic reviewed Trillion EVM cross chain contract'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:

- Can the bridge be susceptible to common bridge vulnerabilities?

1.3. Non-goals and Limitations

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

- Off-chain / server code related to the minting of tokens on the other chain
- Front-end components
- Infrastructure relating to the project
- Key custody

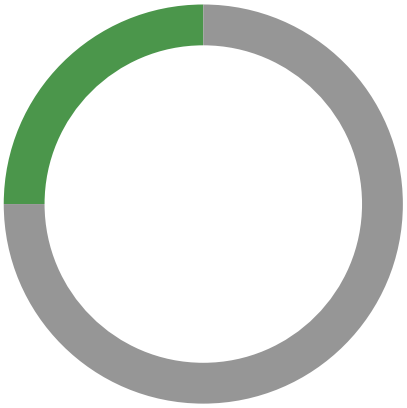
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Trillion EVM cross chain contract contracts, we discovered four 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	3



2. Introduction

2.1. About Trillion EVM cross chain contract

Trillion contributed the following description of Trillion EVM cross chain contract:

> The Trillion EVM Cross-Chain contracts provides a suite of smart contracts designed to facilitate secure and efficient cross-chain interactions. Built using the Foundry framework, these contracts enable seamless communication and asset transfers between Ethereum Virtual Machine (EVM)-compatible blockchains.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case

basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

Trillion EVM cross chain contract Contracts

Type	Solidity
Platform	EVM-compatible
Target	EVM cross chain contracts
Repository	https://github.com/trillion-network/evm-cross-chain-contracts ↗
Version	`8bbb1d14360af789e1e1cea606ce5f8d5f06e856`

Programs	`NonceManager`
	`TokenBurner`
	`TokenMessenger`
	`Message`
	`Ownable2Step`
	`Pausable`
	`Rescuable`
	`TokenController`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 0.9 person weeks. The assessment was conducted by two consultants over the course of 3 calendar days.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia
Engineer
kate@zellic.io ↗

Dimitri Kamenski
Engineer
dimitri@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

November 25, 2024	Kick-off call
November 26, 2024	Start of primary review period
November 27, 2024	End of primary review period

3. Detailed Findings

3.1. Message Nonce uniqueness lacks guarantees

Target	TokenMessenger.sol		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The `TokenMessenger` contract is an event driven bridge that triggers off-chain minting processes on a remote chain after burning tokens on the local chain. The minting process relies on nonce for uniqueness guarantees, however, this is managed separately by `NonceManager` contract. The addition of an external `NonceManager` contract is to ensure consistency in the nonce if the `TokenMessenger` contract is changed or replaced then existing nonce values can be reused by pointing back to the existing `NonceManager`. However, since the `NonceManager` contract can change, guarantees made for the nonce uniqueness may fail in the `TokenManager`.

If the nonce manager is changed or updated, this could result in a nonce being used that has already been used before. Since there is no check made in the `_depositForBurn()` function, that reused nonce may cause unexpected behavior for off-chain event listeners.

Impact

Low severity issue, requires administrative errors involving the removal `NonceManager` followed by the addition of a new one.

Recommendations

Nonce uniqueness should be a strict invariant, avoid circumstances where an external contract can be changed in such a way that could result in nonce reuse. We recommend the removal of the external `NonceManager`, enforcing the nonce increments at the `TokenMessenger` level.

If there is a re-deployment of the `TokenMessenger`, supply a migration process that insures a specific nonce start point, based on where it was previously left at.

Remediation

This issue has been acknowledged by Trillion, and a fix was implemented in commit 56450339 ↗.

3.2. Fee lacks upper bound restrictions

Target	TokenMessenger.sol		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

Fee's in the token bridge contract accrue through native token transfers. An admin set fee minimum is required to transact with the bridge. Although the fee cannot be zero, it can be set arbitrarily high in the `setFee()` function.

Impact

Informational issue as this requires a malicious admin and would simply stop transfers from happening. This same threat exists off-chain if the protocol administrators wanted to prevent transfers from happening.

Recommendations

We recommend adding an upper bound to the fee.

Remediation

This issue has been acknowledged by Trillion.

3.3. Lack of verification on burn tokens

Target	TokenMessenger.sol		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

During token bridging, an input token is burned to ensure 1:1 bridging (excluding fees) against token's minted through off-chain processes. However, at this stage any token can be used as the `_burnTokenAddress`, whereas bridging is only intended to support two tokens `TNUSD` and `TNSGD`.

Impact

The lack of input token validation could potentially result in the burning of arbitrary tokens. However, the impact is currently limited to informational, as the use of arbitrary tokens is mitigated by the burn limit per-message check implemented in the `burn` function of the TokenBurner contract.

```
modifier onlyWithinBurnLimit(address token, uint256 amount) {
    uint256 _allowedBurnAmount = burnLimitsPerMessage[token];
    require(_allowedBurnAmount > 0, "Burn token not supported");
    require(amount <= _allowedBurnAmount, "Burn amount exceeds per tx limit");
    _;
}
```

Recommendations

Since the `onlyWithinBurnLimit` modifier is not included in the TokenMessenger contract and the localBurner contract address is mutable, we recommend adding explicit checks in the `depositForBurn` and `depositForBurnWithCaller` functions that the burn token matches the intended token, this could be done via immutable or constant variables to encourage gas optimizations during bridging operations.

Remediation

This issue has been acknowledged by Trillion.

3.4. Unused functionality should be removed

Target	TokenController.sol		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `TokenController` contract is a base contract that assists child contracts in matching local to remote token addresses. This contract adds complexity, however, important functions are not used elsewhere. Specifically `linkTokenPair()` and `unlinkTokenPair()` intended for matching local tokens to remote tokens present functionality that is not consumed.

Impact

Informational issue, some functionality is adding unnecessary complexity.

Recommendations

We recommend removal of unnecessary functions, and simplification of contract inheritance where possible.

Remediation

This issue has been acknowledged by Trillion, and a fix was implemented in commit `75db8873` ↗.

4. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts 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. Module: NonceManager.sol

Function: `reserveAndIncrementNonce()`

This function allows the current nonce to be incremented. It can only be called by the `localTokenMessenger` address, which is set by the contract owner.

Branches and code coverage

Intended branches

- The nonce is incremented successfully.
  - ☐ Test coverage

Negative behavior

- The caller is not the `localTokenMessenger`
  - ☐ Negative test
- The `localTokenMessenger` is not set.
  - ☐ Negative test

4.2. Module: TokenBurner.sol

Function: `burn(address burnToken, uint256 burnAmount)`

This function allows burning the provided `burnAmount` of tokens. The amount to be burned per message is limited by the `burnLimitsPerMessage`, which is controlled by the `_tokenController` address.

Inputs

- `burnToken`
  - **Control**: Full control.
  - **Constraints**: The `burnLimitsPerMessage` for the `burnToken` must not be zero. The `burnToken` contract must support the `burnByBurnerOnly` functionality.

- **Impact**: The `burnByBurnerOnly` function of the `burnToken` contract is executed to burn the provided amount of tokens.

- `burnAmount`
  - **Control**: Full control.
  - **Constraints**: The amount to be burned must not exceed the `burnLimitsPerMessage` for the burnToken.
  - **Impact**: The amount of tokens to be burned.

Branches and code coverage

Intended branches

- Tokens were burned successfully.
  - ☐ Test coverage

Negative behavior

- The caller is not `localTokenMessenger`
  - ☐ Negative test
- The `burnToken` is not supported
  - ☐ Negative test
- The `burnAmount` exceeds the `burnLimitsPerMessage`
  - ☐ Negative test

Function call analysis

- _token.burnByBurnerOnly(burnAmount)
  - **What is controllable?** `burnAmount`
  - **If the return value is controllable, how is it used and how can it go wrong?** n/a
  - **What happens if it reverts, reenters or does other unusual control flow?** This function burns the specified amount of tokens and will revert if the contract does not own enough tokens to burn.

4.3. Module: TokenMessenger.sol

Function: `depositForBurnWithCaller(uint256 amount, uint32 destinationDomain, bytes32 mintRecipient, address burnToken, bytes32 destinationCaller)`

This function provides the same functionality as the `depositForBurn` function, but the caller controls the `destinationCaller` address. The `destinationCaller` address specifies the particular caller on the destination domain. It is assumed that, if `destinationCaller` is set to `bytes32(0)`, any address can call `receiveMessage()` during the token receiving process.

Function: `depositForBurn(uint256 amount, uint32 destinationDomain, bytes32 mintRecipient, address burnToken)`

This function initiates the cross-chain transfer of tokens. The desired amount of tokens to be transferred will be transferred to the `_localBurner` account and burned from this `_localBurner` account. The caller must also provide the address of the token to be burned before the transfer to the destination chain. This function does not verify the provided token itself, verification is handled in the TokenBurner contract. The `onlyWithinBurnLimit` modifier ensures that the amount does not exceed the `burnLimitsPerMessage` value for the specified token. Additionally, this function emits a `DepositForBurn` event.

Inputs

- `amount`
  - **Control**: Full control.
  - **Constraints**: The caller must hold the specified number of tokens. The amount must be greater than zero.
  - **Impact**: This is the amount of tokens to be burned and transferred to the destination chain.
- `destinationDomain`
  - **Control**: Full control.
  - **Constraints**: The `remoteTokenMessengers` for the provided `destinationDomain` must be set by the contract owner.
  - **Impact**: This represents the domain of the remote TokenMessenger.
- `mintRecipient`
  - **Control**: Full control.
  - **Constraints**: `mintRecipient != bytes32(0)`
  - **Impact**: The recipient of the burned tokens on the destination chain.
- `burnToken`
  - **Control**: Full control.
  - **Constraints**: n/a
  - **Impact**: The token to be burned.

Branches and code coverage

Intended branches

- The specified amount of tokens was successfully burned.
  - ☐ Test coverage

Negative behavior

- The token is not supported.
  - ☐ Negative test

- The caller does not have enough tokens to transfer.
  - ☐ Negative test
- The `destinationDomain` is not supported.
  - ☐ Negative test
- The `mintRecipient` address is zero.
  - ☐ Negative test
- The `amount` exceeds the `burnLimitsPerMessage`.
  - ☐ Negative test

Function call analysis

- _depositForBurn(amount,destinationDomain,mintRecipient,burnToken,bytes32(0)) -> _getRemoteTokenMessenger(_destinationDomain)
  - **What is controllable?** `_destinationDomain`
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the address of the `TokenMessenger` on `_destinationDomain`. This address is controlled by the contract owner.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the caller provides a non-existent `_destinationDomain` value.
- _depositForBurn(amount,destinationDomain,mintRecipient,burnToken,bytes32(0)) -> _getLocalBurner()
  - **What is controllable?** n/a
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the address of the local burner contract, which is set by the contract owner.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the burner contract address is not set.
- _depositForBurn(amount,destinationDomain,mintRecipient,burnToken,bytes32(0)) -> _burnToken.transferFrom(_msgSender(), address(_localBurner), _amount)
  - **What is controllable?** `_burnTokenAddress`, `_amount`
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns true if the transfer of tokens was successful.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the caller does not own enough tokens to transfer.
- _depositForBurn(amount,destinationDomain,mintRecipient,burnToken,bytes32(0)) -> _localBurner.burn(_burnTokenAddress, _amount)
  - **What is controllable?** `_burnTokenAddress`, `_amount`
  - **If the return value is controllable, how is it used and how can it go wrong?** n/a
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the provided amount exceeds the burn limit.
- _depositForBurn(amount,destinationDomain,mintRecipient,burnToken,bytes32(0)) -> _getNonceManager()
  - **What is controllable?** n/a

- **If the return value is controllable, how is it used and how can it go wrong?** Returns the address of the nonce manager contract, which is set by the contract owner.
- **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the `nonceManager` is not set.

- _depositForBurn(amount,destinationDomain,mintRecipient,burnToken,bytes32(0)) -> _nonceManager.reserveAndIncrementNonce()
  - **What is controllable?** n/a
  - **If the return value is controllable, how is it used and how can it go wrong?** Increments the current nonce and returns the updated nonce.
  - **What happens if it reverts, reenters or does other unusual control flow?** No problems.

5. Assessment Results

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

During our assessment on the scoped Trillion EVM cross chain contract contracts, we discovered four 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.