Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Thunder Wrapped BNB
Website: thundercore.com
Platform: Binance Network
Language: Solidity
Date: April 4th, 2025

Table of Contents

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

THIS IS A SECURITY AUDIT REPORT DOCUMENT THAT MAY CONTAIN INFORMATION THAT IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contract audit initiatives, the smart contract of BNB Token from thundercore.com was audited. The audit used manual analysis as well as automated software tools. This report presents all the findings regarding the audit performed on April 4th, 2025.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

ForeignBridgeWithNativeToken - ERC20 to ERC20 Bridge

The **ForeignBridgeWithNativeToken** is an Ethereum-based smart contract designed to facilitate cross-chain asset transfers, specifically for **Thunder Wrapped BNB (BNB)**. It extends the **ForeignBridgeErcToErcV2**, allowing secure and validated transfers between different blockchain networks.

Key Features:

- **Cross-chain Asset Transfers**: Supports ERC20-to-ERC20 token bridging.
- **Validator-based Security**: Utilizes signature verification to ensure transactions are authorized.
- **Fallback Mechanism**: Redirects failed transfers to a designated fallback recipient.
- **Token Handling**: Implements transfer and balance-check functions.
- **Upgradeability**: Extends an existing bridge implementation for better flexibility.
- **Restricted Approvals & Transfers**: Disables approve and transferFrom to prevent unauthorized token movements.

Core Functionality:

- executeSignatures(): Verifies validator signatures and processes cross-chain transactions.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

- onExecuteMessage(): Handles fund transfers, ensuring successful execution or fallback.
- setFallbackRecipient(): Allows setting a backup address for failed transactions.
- tokenTransfer(): Transfers tokens to the recipient or fallback address if necessary.

This contract ensures a robust and decentralized method of transferring **Thunder Wrapped BNB** securely between chains.

Audit scope

Name	**Code Review and Security Analysis Report for Thunder Wrapped BNB Token Smart Contract**
Platform	**Binance Network**
File	ForeignBridgeWithNativeToken.sol
Smart Contract Code	0xba15418434cc081b593b2ccf99d6ae5d7c4ef7b5
Audit Date	April 4th, 2025

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Thunder Wrapped BNB • Symbol: BNB • Decimals: 18	**YES, This is valid.**
Key Features: **1. Cross-Chain Asset Transfers** • Facilitates ERC20-to-ERC20 token bridging between networks. • Designed specifically for Thunder Wrapped BNB (BNB). **2. Validator-Based Security** • Uses multiple validator signatures for transaction authentication. • Ensures that only valid transactions are executed. **3. Execution & Message Processing** • executeSignatures(): Verifies signatures and executes cross-chain transfers. • onExecuteMessage(): Processes token transfers and handles fallback scenarios. **4. Fallback Mechanism** • If a transaction fails, tokens are redirected to a designated fallback recipient. • The fallback recipient can be set using setFallbackRecipient().	

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

5. Token Handling ● Implements transfer() to facilitate token movement. ● Supports querying total supply (totalSupply()) and balances (balanceOf()). **6. Transfer Restrictions** ● approve() and transferFrom() are permanently disabled (reverted) to prevent unauthorized token movements. **7. Upgradeable Architecture** ● Extends ForeignBridgeErcToErcV2, allowing future improvements. **8. Native Token Interaction** ● Uses native ETH-based balances instead of standard ERC20 token balances. ● Handles token claims via claimTokens(). **9. Event Emissions for Transparency** ● Transfer(from, to, value): Logs successful transfers. ● RecipientRedirected(from, to): Logs fallback transactions.	

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Audit Summary

According to the standard audit assessment, the Customer's solidity-based smart contract is **“Secured.”** Also, these contracts contain owner control, which does not make them fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed, and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section, and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 1 low, and 3 very low-level issues.

Investors' Advice: A Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inheritance, and Interfaces. This is a compact and well-written smart contract.

The libraries in BNB Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address, and its properties/methods can be reused many times by other contracts in the BNB Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contract. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a BNB Token smart contract code in the form of a bscscan.com web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure that is based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

AS-IS overview

ForeignBridgeWithNativeToken.sol : Functions

Sl.	Functions	Type	Observation	Conclusion
1	initialize	write	Passed	No Issue
2	_setFallbackRecipient	internal	Passed	No Issue
3	setFallbackRecipient	write	Critical operation lacks event log, missing zero address validation	Refer Audit Findings
4	fallbackRecipient	read	Passed	No Issue
5	erc20token	read	Passed	No Issue
6	totalSupply	read	Passed	No Issue
7	balanceOf	read	Passed	No Issue
8	allowance	read	Passed	No Issue
9	transfer	write	Passed	No Issue
10	approve	write	Passed	No Issue
11	transferFrom	write	Passed	No Issue
12	tokenTransfer	internal	Passed	No Issue
13	onExecuteMessage	internal	Passed	No Issue
14	claimTokens	write	Critical operation lacks event log, Custom error messages are missing	Refer Audit Findings
15	onExecuteMessage	internal	Passed	No Issue
16	executeSignatures	external	Passed	No Issue
17	setExecutionDailyLimit	write	Custom error messages are missing	Refer Audit Findings
18	setFeePercent	write	Critical operation lacks event log, Custom error messages are missing	Refer Audit Findings

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss, etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens being lost
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, which can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

(1) Critical operation lacks event log:

Missing event log for:

- setFeePercent
- setFallbackRecipient
- claimTokens
- setErc20token

Resolution: Please write an event log for the listed events.

Very Low / Informational / Best practices:

(1) Missing zero address validation

Detect missing zero address validation in setFallbackRecipient

Resolution: We suggest adding the required condition.

(2) Custom error messages missing:

Some “require” conditions in the code miss custom error messages.

Resolution: We suggest putting custom error messages in “require” conditions in the

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

following functions, which will be helpful in debugging the code for the failed transactions. Like: claimTokens, setExecutionDailyLimit, setFeePercent, etc.

(3) Infinite loops:

As array elements increases, it will cost more and more gas. And eventually, it will stop all the functionality. After several hundred transactions, all those functions that depend on it will stop. We suggest avoiding loops. For example, use mapping to store the array index. And query that data directly, instead of looping through all the elements to find an element.

Resolution: Adjust logic to replace loops with mapping or other code structure.

- hasEnoughValidSignatures()
- onExecuteMessage() - validators.length

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Centralization

This smart contract has some functions that can be executed by the Admin (Owner) only. If the admin wallet's private key is compromised, then it creates trouble. The following are Admin functions:

ForeignBridgeWithNativeToken.sol

- setFallbackRecipient: The owner can set _recipient address.
- claimTokens: The owner can claim tokens.

ForeignBridgeErcToErcV2.sol

- setExecutionDailyLimit: The owner can set the execution daily limit.
- setFeePercent The owner can set the fee percentage.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Conclusion

We were given a contract code in the form of bscscan.com web links. We have used all possible tests based on the given objects as files. We observed 1 low and 3 informational issues in the smart contract, and those issues are not critical. So, **it's good to go for production**.

Since possible test cases can be unlimited for such smart contract protocols, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on the standard audit procedure scope, is **"Secured".**

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early, even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation are an ongoing collaborative process after we deliver our report, and before the details are made public.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract by the best industry practices at the date of this report, about: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Since the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

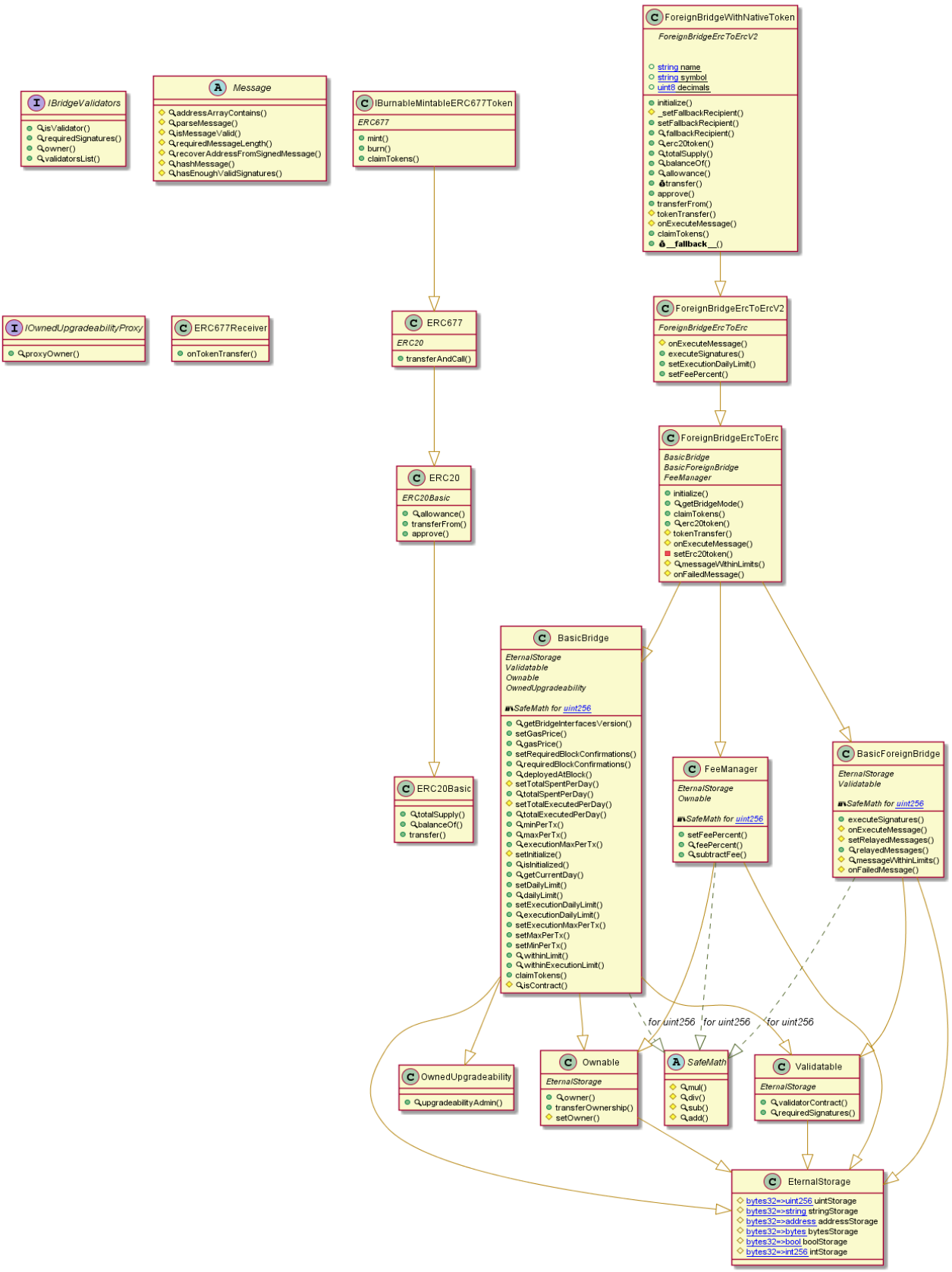
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Appendix

Code Flow Diagram - Thunder Wrapped BNB Token



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We analyzed the project altogether. Below are the results.

Slither Log >> ForeignBridgeWithNativeToken.sol

```
INFO:Detectors:
ForeignBridgeWithNativeToken.tokenTransfer(address,uint256)
(ForeignBridgeWithNativeToken.sol#880-882) sends eth to arbitrary user
	Dangerous calls:
	- _recipient.send(_amount) (ForeignBridgeWithNativeToken.sol#881)
ForeignBridgeWithNativeToken.onExecuteMessage(address,uint256)
(ForeignBridgeWithNativeToken.sol#884-895) sends eth to arbitrary user
	Dangerous calls:
	- _fallbackRecipient.transfer(_amount) (ForeignBridgeWithNativeToken.sol#888)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
ForeignBridgeWithNativeToken.balanceOf(address).owner
(ForeignBridgeWithNativeToken.sol#818) shadows:
	- Ownable.owner() (ForeignBridgeWithNativeToken.sol#285-287) (function)
ForeignBridgeWithNativeToken.allowance(address,address).owner
(ForeignBridgeWithNativeToken.sol#829) shadows:
	- Ownable.owner() (ForeignBridgeWithNativeToken.sol#285-287) (function)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
2 different versions of Solidity are used:
	- Version constraint ^0.4.24 is used by:
		-^0.4.24 (ForeignBridgeWithNativeToken.sol#7)
		-^0.4.24 (ForeignBridgeWithNativeToken.sol#507)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

```
ForeignBridgeErcToErcV2.onExecuteMessage(address,uint256)
(ForeignBridgeWithNativeToken.sol#712-714) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Reentrancy in ForeignBridgeWithNativeToken.onExecuteMessage(address,uint256)
(ForeignBridgeWithNativeToken.sol#884-895):
	External calls:
	- _fallbackRecipient.transfer(_amount) (ForeignBridgeWithNativeToken.sol#888)
	Event emitted after the call(s):
	- RecipientRedirected(_recipient,_fallbackRecipient)
(ForeignBridgeWithNativeToken.sol#889)
	- Transfer(this,_fallbackRecipient,_amount) (ForeignBridgeWithNativeToken.sol#890)
Reentrancy in ForeignBridgeWithNativeToken.onExecuteMessage(address,uint256)
(ForeignBridgeWithNativeToken.sol#884-895):
	External calls:
	- ! tokenTransfer(_recipient,_amount) (ForeignBridgeWithNativeToken.sol#885)
		- _recipient.send(_amount) (ForeignBridgeWithNativeToken.sol#881)
	Event emitted after the call(s):
	- Transfer(this,_recipient,_amount) (ForeignBridgeWithNativeToken.sol#893)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Slither:ForeignBridgeWithNativeToken.sol analyzed (19 contracts with 93 detectors), 98
result(s) found
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

ForeignBridgeWithNativeToken.sol

Gas costs:
Low level calls:
Use of "send": "send" does not throw an exception when not successful, make sure you deal with the failure case accordingly. Use "transfer" whenever failure of the ether transfer should rollback the whole transaction. Note: if you "send/transfer" ether to a contract the fallback function is called, the callees fallback function is very limited due to the limited amount of gas provided by "send/transfer". No state changes are possible but the callee can log the event or revert the transfer. "send/transfer" is syntactic sugar for a "call" to the fallback function with 2300 gas and a specified ether value.
more
Pos: 881:11:

Gas costs:
Gas requirement of function ForeignBridgeWithNativeToken.claimTokens is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 897:2:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 723:8:

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

Solhint Linter

Solhint Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

ForeignBridgeWithNativeToken.sol

```
Compiler version ^0.4.24 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:6
Provide an error message for require
Pos: 9:656
Avoid to use low level calls.
Pos: 16:665
Provide an error message for require
Pos: 9:692
Provide an error message for revert
Pos: 9:701
Constant name must be in capitalized SNAKE_CASE
Pos: 3:746
Constant name must be in capitalized SNAKE_CASE
Pos: 3:747
Constant name must be in capitalized SNAKE_CASE
Pos: 3:748
Provide an error message for require
Pos: 5:789
Variable "owner" is unused
Pos: 5:828
Variable "spender" is unused
Pos: 5:829
```

Software analysis result:

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.
Email: audit@EtherAuthority.io

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io