Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	OWC Bridge
Website:	oneworldchain.org
Platform:	Base Chain, Ethereum, and Binance Network
Language:	Solidity
Date:	June 28th, 2024

Table of contents

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH 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.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the OWC chain Provider Limited to perform the Security audit of the OWC Bridge smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on June 28th, 2024.

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

The provided Solidity contract implements a bridge for token transfers, featuring essential ERC20 functions. The contract includes:

- **Token Operations:** Functions to mint, burn, and transfer tokens.
- **Ownership Control:** Functions to transfer and accept ownership, and manage signers.
- **Fee and Reserve Management:** Setting fee and reserve wallets, and managing reserve fund thresholds.
- **Tax Handling:** Processing tax on transactions and setting tax rates.
- **Transaction Controls:** Setting minimum and maximum transaction values, managing no-control token addresses, and handling token thresholds for different tokens.

It uses interfaces for ERC20 tokens and a custom USDT token.

Audit scope

Name	Code Review and Security Analysis Report for OWC Bridge Smart Contract
Platform	**Base Chain Network, Ethereum, Binance Smart Chain / Solidity**
File	Bridge.sol
Basescan Smart Contract Address	0x3D9CE0d8dF32a0D706F29DC82E75c07a8f7B320b
Etherscan Smart Contract Address	0x768f7F738e15f8fdeeEAC4206ef1686fc21cB915
Bscscan Smart Contract Address	0x239aBE992aAFE1347dcC560A801ca352f70005A4
Audit Date	June 28th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
The owner has control over the following functions: ● The owner can update the fee wallet address. ● The owner can update the reserve wallet address. ● The owner can update the USDT token address. ● The owner can update the threshold amount. ● The owner can update the noControl address. ● The owner can modify the token reserve threshold values. ● The owner can modify the transfer tax. ● The owner can change the minimum and maximum amount of tokens that can be bridged in a single transaction only. ● The owner can update the signer address. ● The current owner can transfer ownership of the contract to a new account.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, Customer`s solidity-based smart contracts are **“secured”**. This token contract contains owner control, which does not make it 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.

Investor 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.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	9.99%
Modify Tax	Yes
Fee Check	Not Detected
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	No
Max Tax?	Yes
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	Yes
Is it a Proxy Contract?	No
Can Take Ownership?	Yes
Creator Percentage?	0.00%
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

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

The libraries in the OWC Bridge 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 OWC Bridge.

The OWC Bridge team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is used, which is a good thing.

Documentation

We were given an OWC Bridge smart contract code in the form of a basescan, etherscan, and bscscan weblink. The hash of that code is mentioned above in the table.

As mentioned above, the 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.

Another source of information was its website oneworldchain.org which provided rich information about the project architecture.

Use of Dependencies

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

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

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	receive	external	Passed	No Issue
3	coinIn	external	Passed	No Issue
4	coinOut	external	access only Signer	No Issue
5	tokenIn	external	Passed	No Issue
6	tokenOut	external	access only Signer	No Issue
7	processTax	read	Passed	No Issue
8	mintTokens	internal	Passed	No Issue
9	burnTokens	internal	Passed	No Issue
10	setFeeWallet	external	Function input parameters lack check	Refer Audit Findings
11	setReserveWallet	external	Function input parameters lack check	Refer Audit Findings
12	setUSDTAddress	external	access only owner	No Issue
13	setFundThreshold	external	The reserveFundThreshold limit is not set	Refer Audit Findings
14	transferTokenOwnership	external	access only owner	No Issue
15	setNoControl	external	access only owner	No Issue
16	setTokenReserveThreshold	external	Function input parameters lack check, The TokenFundThreshold limit is not set	Refer Audit Findings
17	setTransferTax	external	access only owner	No Issue
18	updateMinMaxTx	external	Max tax limit is not set	Refer Audit Findings
19	getMinMaxTxValues	external	Passed	No Issue
20	getTransferTax	external	Passed	No Issue
21	onlyOwner	modifier	Passed	No Issue
22	onlySigner	modifier	Passed	No Issue

23	changeSigner	write	access only owner	No Issue
24	transferOwnership	write	access only owner	No Issue
25	acceptOwnership	write	Passed	No Issue

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 lose
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that 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.

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) Function input parameters lack of check:

Variable validation is not performed in the below functions:

- setTokenReserveThreshold = forToken
- setReserveWallet = _reserveWallet
- setFeeWallet = _feeWallet

Resolution: We advise to put validation: integer type variables should be greater than 0 and address type variables should not be addressed (0).

Very Low / Informational / Best practices:

(1) Max tax limit is not set:

```
/**
     * @notice Changes the minimum and maximum amount of tokens that
can be bridge in a single transaction
     * @dev onlyOwner.
     * Emits an {minMaxTxUpdated} event
     * @param newMinTx, newMaxTx Base 1000, so 1% = 1
```

```
     */
    function updateMinMaxTx(uint256 newMinTx, uint256 newMaxTx)
external onlyOwner {
        minTx = newMinTx;
        maxTx = newMaxTx;
        emit minMaxTxUpdated(minTx, maxTx);
    }
```

The owner can set any value to the individual max tax variable. This might deter investors as they could be wary that these taxes might one day be set to 100%.

Resolution: Consider adding an explicit limit to the maximum tax value.

(2) The reserveFundThreshold limit is not set:

```
/* set Threshold*/
 function setFundThreshold(uint256 _amount) external onlyOwner
returns(uint256 oldAmount, uint256 newAmount){
        oldAmount = reserveFundThreshold;
        reserveFundThreshold = _amount;
        newAmount = _amount;
    }
```

The owner can set any value to the reserveFundThreshold variable. This might deter investors as they could be wary that this reserveFundThreshold might one day be set to 100%.

Resolution: Consider adding an explicit limit to the reserveFundThreshold value.

(3) The TokenFundThreshold limit is not set:

```
  /* Modify the token reserve threshold values*/
  function setTokenReserveThreshold(address forToken, uint256
```

```
threshold) external onlyOwner{
        tokenFundThreshold[forToken] = threshold;
    }
```

The owner can set any value to the TokenFundThreshold variable. This might deter investors as they could be wary that these TokenFundThreshold might one day be set to 100%.

Resolution: Consider adding an explicit limit to the tokenFundThreshold value.

Centralization

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

Bridge.sol

- coinOut: The signer can coin out.
- tokenOut: The signer can token out.
- setFeeWallet: The owner can update the fee wallet address.
- setReserveWallet: The owner can update the reserve wallet address.
- setUSDTAddress: The owner can update the USDT token address.
- setFundThreshold: The owner can update the threshold amount.
- transferTokenOwnership: The current owner can transfer ownership of the contract to a new account.
- setNoControl: The owner can update the noControl address.
- setTokenReserveThreshold: The owner can modify the token reserve threshold values.
- setTransferTax: The owner can modify the transfer tax.
- updateMinMaxTx: The owner can change the minimum and maximum amount of tokens that can be bridged in a single transaction only.

Ownable.sol

- changeSigner: The owner can update the signer address.
- transferOwnership: Current owner can transfer ownership of the contract to a new account.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a basescan, etherscan, and bscscan weblink., and we have used all possible tests based on given objects. We have observed 1 low and 3 Informational severity issues. but this issue is not critical. **So, the smart contract is ready for mainnet deployment.**

Since possible test cases can be unlimited for such smart contracts protocol, 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 standard audit procedure scope, is **"Secured".**

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 white box 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.

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, 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 is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: 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).

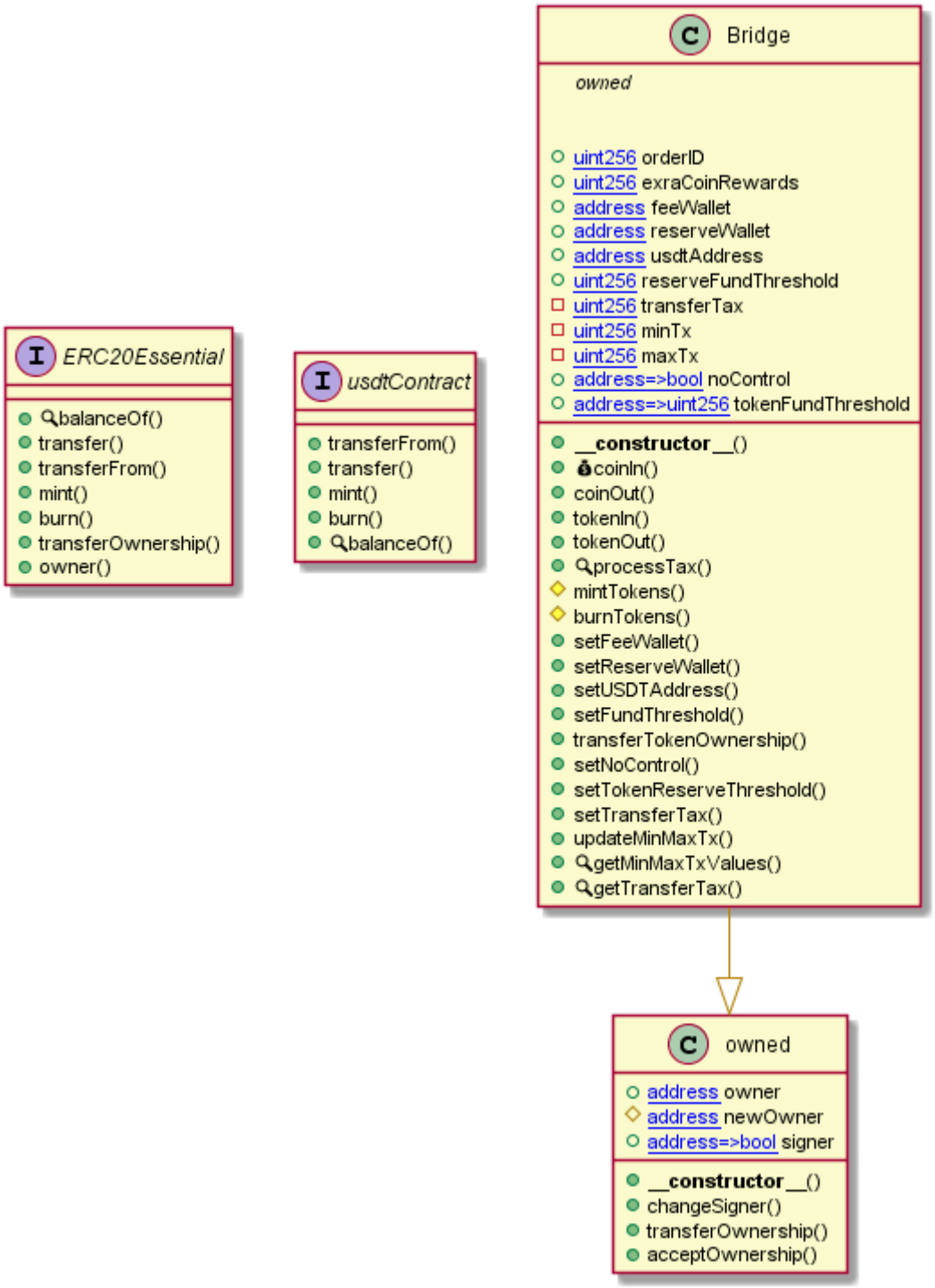
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While 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 the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - OWC Bridge



Slither Results Log

Slither Log >> Bridge.sol

```
INFO:Detectors:
Bridge.transferTokenOwnership(address,address).newOwner (Bridge.sol#265) shadows:
	- owned.newOwner (Bridge.sol#41) (state variable)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Bridge.setFundThreshold(uint256) (Bridge.sol#258-262) should emit an event for:
	- reserveFundThreshold = _amount (Bridge.sol#260)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Bridge.coinOut(address,uint256,uint256).user (Bridge.sol#151) lacks a zero-check on :
		- address(user).transfer(amount) (Bridge.sol#152)
Bridge.setFeeWallet(address)._feeWallet (Bridge.sol#236) lacks a zero-check on :
		- feeWallet = _feeWallet (Bridge.sol#238)
Bridge.setReserveWallet(address)._reserveWallet (Bridge.sol#243) lacks a zero-check on :
		- reserveWallet = _reserveWallet (Bridge.sol#245)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in Bridge.tokenIn(address,uint256,uint256,address) (Bridge.sol#158-190):
	External calls:
	- usdtContract(tokenAddress).transferFrom(msg.sender,address(this),tokenAmount)
(Bridge.sol#167)
Reentrancy in Bridge.tokenOut(address,address,uint256,uint256,uint256) (Bridge.sol#193-209):
	External calls:
	- usdtContract(tokenAddress).transfer(user,tokenAmount) (Bridge.sol#197)
	- ERC20Essential(tokenAddress).transfer(user,tokenAmount) (Bridge.sol#199)
	- (minted,None) = mintTokens(tokenAddress,user,tokenAmount) (Bridge.sol#203)
		- ERC20Essential(tokenAddress).mint(userAddress,amountToMint) (Bridge.sol#221)
	Event emitted after the call(s):
	- TokenOut(_orderID,tokenAddress,user,minted,chainID) (Bridge.sol#206)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Pragma version0.8.17 (Bridge.sol#6) allows old versions
solc-0.8.17 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Event Bridge.minMaxTxUpdated(uint256,uint256) (Bridge.sol#114) is not in CapWords
Event Bridge.transferTaxUpdated(uint256) (Bridge.sol#115) is not in CapWords
```

```
Parameter Bridge.coinOut(address,uint256,uint256)._orderID (Bridge.sol#151) is not in
mixedCase
Parameter Bridge.tokenOut(address,address,uint256,uint256,uint256)._orderID (Bridge.sol#193)
is not in mixedCase
Parameter Bridge.setFeeWallet(address)._feeWallet (Bridge.sol#236) is not in mixedCase
Parameter Bridge.setReserveWallet(address)._reserveWallet (Bridge.sol#243) is not in
mixedCase
Parameter Bridge.setUSDTAddress(address)._tokenAddress (Bridge.sol#250) is not in
mixedCase
Parameter Bridge.setFundThreshold(uint256)._amount (Bridge.sol#258) is not in mixedCase
Parameter Bridge.setTransferTax(uint256)._transferTax (Bridge.sol#288) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
Reentrancy in Bridge.coinIn(address) (Bridge.sol#134-149):
        External calls:
        - address(feeWallet).transfer(tax) (Bridge.sol#141)
        - address(reserveWallet).transfer(afterTax) (Bridge.sol#144)
        Event emitted after the call(s):
        - CoinIn(orderID,msg.sender,afterTax,outputCurrency) (Bridge.sol#147)
Reentrancy in Bridge.coinOut(address,uint256,uint256) (Bridge.sol#151-155):
        External calls:
        - address(user).transfer(amount) (Bridge.sol#152)
        Event emitted after the call(s):
        - CoinOut(_orderID,user,amount) (Bridge.sol#153)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
Bridge.slitherConstructorVariables() (Bridge.sol#89-337) uses literals with too many digits:
        - minTx = 1000000000000000 (Bridge.sol#99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
Bridge.exraCoinRewards (Bridge.sol#92) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-decl
ared-constant
INFO:Slither:Bridge.sol analyzed (4 contracts with 93 detectors), 37 result(s) found
```

Solidity Static Analysis

Bridge.sol

Gas costs:
Gas requirement of function Bridge.coinIn 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: 135:4:

Gas costs:
Gas requirement of function Bridge.tokenIn 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: 159:4:

Similar variable names:
Bridge.setFeeWallet(address) : Variables have very similar names "feeWallet" and "newWallet".
Note: Modifiers are currently not considered by this static analysis.
Pos: 240:8:

No return:
usdtContract.balanceOf(address): Defines a return type but never explicitly returns a value.
Pos: 30:4:

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.
Pos: 290:8:

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 214:22:

Solhint Linter

Bridge.sol

```
Compiler version 0.8.17 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:6
Contract name must be in CamelCase
Pos: 1:23
Contract name must be in CamelCase
Pos: 1:38
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:47
Provide an error message for require
Pos: 9:53
Use double quotes for string literals
Pos: 37:59
Provide an error message for require
Pos: 9:75
Event name must be in CamelCase
Pos: 5:114
Event name must be in CamelCase
Pos: 5:115
Code contains empty blocks
Pos: 33:120
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:124
```

Software analysis result:

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

Email: audit@EtherAuthority.io