

SMART CONTRACT

Security Audit Report

Customer:	Cardex
Website:	www.cardex.gallery
Platform:	Binance Smart Chain
Language:	Solidity
Date:	August 31st, 2021

Table of contents

Introduction	4
Project Background	4
Audit Scope	4
Claimed Smart Contract Features	5
Audit Summary	6
Technical Quick Stats	7
Code Quality	8
Documentation	8
Use of Dependencies	8
AS-IS overview	9
Severity Definitions	11
Audit Findings	12
Conclusion	16
Our Methodology	17
Disclaimers	19
Appendix	
• Code Flow Diagram	20
• Slither Results Log	22
• Solidity static analysis	25
• Solhint Linter	29

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 PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

EtherAuthority was contracted by the Cardex team to perform the Security audit of the Cardex Token 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 August 31st, 2021.

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

Cardex (CX) is a BEP20 standard token contract on the Binance Smart Chain blockchain. Cardex Token is a Native Token with the tax functionality.

Audit scope

Name	Code Review and Security Analysis Report for Cardex (CX) Token Smart Contract
Platform	BSC / Solidity
File 1	CX.sol
File 1 Commit	b6e3c56069039d373f4c4130a209981fd178b5e8
File 2	Safe.sol
File 2 Commit	8f444c65c3ebbc5bcf5337e9cdc72e023ea7b63e
File 3	ICXRecipient.sol
File 3 Commit	8f444c65c3ebbc5bcf5337e9cdc72e023ea7b63e
Audit Date	August 31st, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: <ul style="list-style-type: none">• Name: Cardex• Symbol: CX• Decimals: 18• Total Supply: 100,000• Type: BEP20• Platform: Binance Smart Chain	YES, This is valid.
100,000 tokens will be minted and sent to the contract creator at the time of contract deployment. New tokens can never be minted	YES, This is valid.
Owner has control over following functions: <ul style="list-style-type: none">• set tax receiving address• set safe contract address• transfer ownership to another wallet• renounce the ownership	YES, This is valid. Owner wallet's private key must be handled very securely. Because if that is compromised, then it will create problems.

Audit Summary

According to the standard audit assessment, Customer's solidity smart contracts are **"Nearly Secured"**. The Cardex token contains the owner's functions. So, the owner must handle those functions as per the business plan.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 1 high, 0 medium and 2 low and some very low level issues.

Investors Advice: 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	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	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	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Moderated
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Moderated
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 not set	Passed
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: PASSED

Code Quality

This audit scope has 3 smart contract files. Smart contracts also contain Libraries, Smart contracts inherits and Interfaces. These are compact and well written contracts.

The libraries in Cardex 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 Cardex token.

The Cardex Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Some code parts are **not** well commented on smart contracts.

Documentation

We were given a Cardex smart contracts code in the form of a github web link. The commits of that code are mentioned above in the table.

As mentioned above, some code parts are **not** well commented. So it is not 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 official website <https://www.cardex.gallery/> which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects. And their core code blocks are written well.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

CX.sol

(1) imports

- (a) Address.sol
- (b) SafeMath.sol
- (c) Context.sol
- (d) Ownable.sol
- (e) IERC20.sol

(2) Inherited contracts

- (a) Context
- (b) IERC20
- (c) Ownable

(3) Usages

- (a) using SafeMath for uint256;
- (b) using Address for address;

(4) Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	name	read	Passed	No Issue
3	symbol	read	Passed	No Issue
4	decimals	read	Passed	No Issue
5	totalSupply	read	Passed	No Issue
6	balanceOf	read	Passed	No Issue
7	allowance	read	Passed	No Issue
8	approve	write	Passed	No Issue
9	transferFrom	write	Passed	No Issue
10	increaseAllowance	write	Passed	No Issue
11	decreaseAllowance	write	Passed	No Issue
12	approve	internal	Passed	No Issue
13	transfer	write	Redundant code, Reentrancy possibility	Refer audit finding section
14	transfer	internal	Passed	No Issue
15	setTaxReceiveAddress	write	Owner function	No Issue
16	setSafeContractAddress	write	Owner function	No Issue

Safe.sol

(1) imports

- (f) Address.sol
- (g) SafeMath.sol
- (h) IERC20.sol
- (i) ICXRecipient.sol

(2) Inherited contracts

- (d) ICXRecipient

(3) Usages

- (c) using SafeMath for uint256;
- (d) using Address for address;

(4) Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	getSafeETHBalance	read	Passed	No Issue
3	getSafeCXBalance	read	Passed	No Issue
4	getCirculatingSupply	read	Passed	No Issue
5	tokenFallback	write	Minor logical issue	Refer audit finding section
6	sendBackEth	write	Passed	No Issue

ICXRecipient.sol

(1) Functions

Sl.	Functions	Type	Observation	Conclusion
1	tokenFallback	Interface function	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 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 significant impact on execution
Informational / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

No Critical severity vulnerabilities were found.

High

(1) Reentrancy possibility in CX.sol

```
104         if(nonTaxedAddresses[_msgSender()] == true || TAX_FRACTION == 0 || recipient
105             ICXRecipient receiver = ICXRecipient(recipient);
106             address payable payableTransferer = payable(_msgSender());
107
108             receiver.tokenFallback(payableTransferer, amount);
```

The transfer() method of CX.sol calls tokenFallback() function of safe contract at line number #108. The tokenFallback() function transfers BNB to the user using the transfer method.

This logic is not vulnerable to reentrancy for now as the transfer method to send BNB only allows 2300 gas, which is not enough to reenter.

However, according to [this article](#), if we want to build the “future-proof” smart contracts, then we can not rely on the gas cost. And thus the smart contract must implement Checks-Effects-Interactions or Reentrancy Guard pattern.

Solution: we recommend to move _transfer() function from line number #120 to the beginning of the function block at line number #101. The reason is that this will update the token balance of the user first before calling the tokenFallback() of the safe.sol contract. Another solution is to use the [reentrancy guard](#) modifier in the function.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Code lines ordering in CX.sol

```
function transferFrom(address sender, address recipient, uint256 amount) public {
    _transfer(sender, recipient, amount);
    _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount,
    return true;
```

In the transferFrom() function, we suggest putting _approve() first and then _transfer(). The reason is that if a transaction is reverted due to lack of an allowance, then it will run _transfer() and consume more gas for the failed transactions.

(2) Redundant code in CX.sol

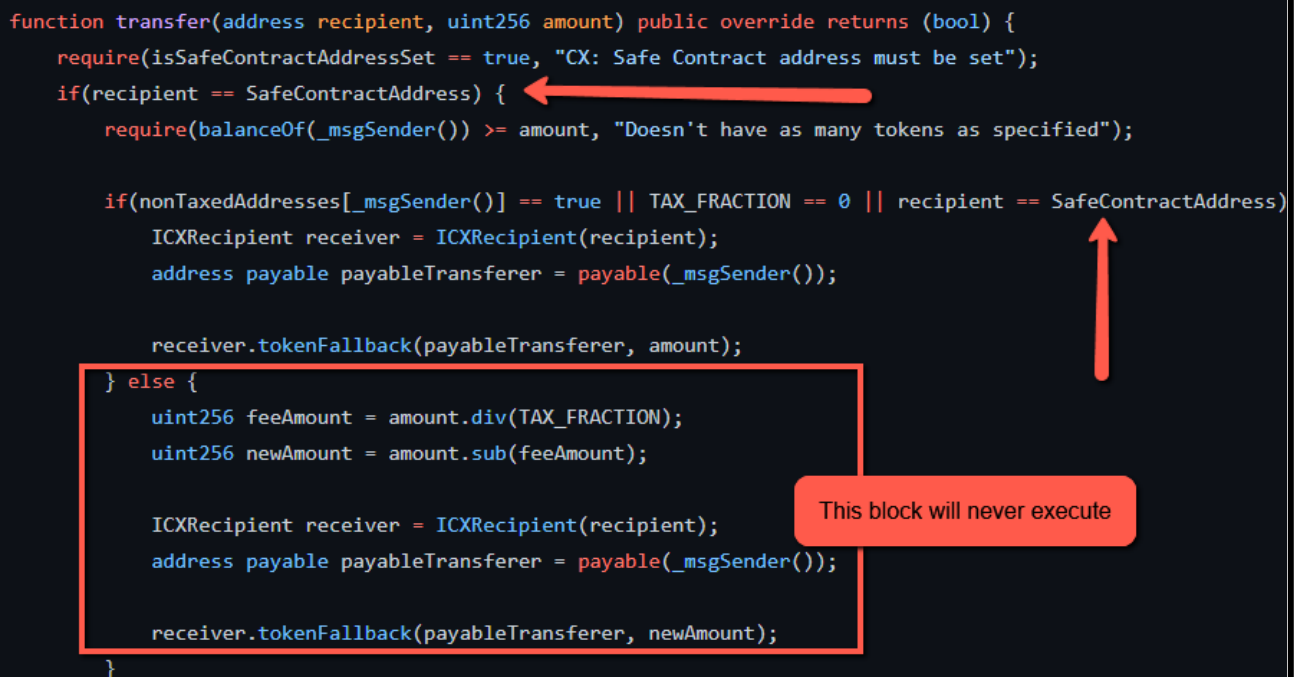
```
function transfer(address recipient, uint256 amount) public override returns (bool) {
    require(isSafeContractAddressSet == true, "CX: Safe Contract address must be set");
    if(recipient == SafeContractAddress) {
        require(balanceOf(_msgSender()) >= amount, "Doesn't have as many tokens as specified");

        if(nonTaxedAddresses[_msgSender()] == true || TAX_FRACTION == 0 || recipient == SafeContractAddress)
            ICXRecipient receiver = ICXRecipient(recipient);
            address payable payableTransferer = payable(_msgSender());

            receiver.tokenFallback(payableTransferer, amount);
        } else {
            uint256 feeAmount = amount.div(TAX_FRACTION);
            uint256 newAmount = amount.sub(feeAmount);

            ICXRecipient receiver = ICXRecipient(recipient);
            address payable payableTransferer = payable(_msgSender());

            receiver.tokenFallback(payableTransferer, newAmount);
        }
    }
```



The transfer function has conditions at line numbers #101 and #104 such that it will cause the other code block to never execute, making it redundant.

However, this does not cause any problem because the fee deduction for every token transfer to users except *safeContractAddress*, will happen inside the _transfer function.

So, we suggest to remove this part at line number #110 to #116 because token transfer among normal users should not trigger tokenFallback() of Safe.sol

Informational / Best practices:

(1) Consider using the latest solidity compiler while deploying

```
pragma solidity ^0.6.0;
```

Although this does not create major security vulnerabilities, the latest solidity version has lots of improvements, so it's recommended to use the latest solidity version, which is 0.8.7 at the time of this audit.

(2) Make variables constant

```
string private _name = "Cardex";  
string private _symbol = "CX";  
uint8 private _decimals = 18;  
uint256 private _totalSupply = 100000e18;
```

These variable's values will be unchanged. So, please make it constant. It will save some gas. Just put a constant keyword.

(3) All functions which are not called internally, must be declared as external. It is more efficient as sometimes it saves some gas.

<https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices>

(4) Approve of ERC20 / BEP20 standard:

To prevent attack vectors regarding approve() like the one described here:

https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM, clients SHOULD make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender. THOUGH the contract itself shouldn't enforce it, to allow backwards compatibility with contracts deployed before

(5) Unnecessary conditions in CX.sol

```
if(nonTaxesAddresses[_msgSender()] == true || TAX_FRACTION == 0 ||
```

The mapping nonTaxesAddress will never have any value as there are no functions to add any wallet to that mapping. The same way TAX_FRACTION variable will be constant and its value will never be zero.

Therefore, there is no need to use them in such conditions. Removing those checks will save some gas.

(6) Missing events for state changing functions:

Any function which does important state changes, must emit an event for the clients to properly track those changes. Following functions should emit an event.

- setTaxReceiveAddress in CX.sol
- setSafeContractAddress in CX.sol
- sendBackEth in Safe.sol

Conclusion

We were given a contract code. And we have used all possible tests based on given objects. We observed some issues in the smart contract. **It's good to go to production after fixing/acknowledging those issues.**

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 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 As-is overview section of the report.

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

Security state of the reviewed contract, based on standard audit procedure scope, is **"Nearly 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 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 web site 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).

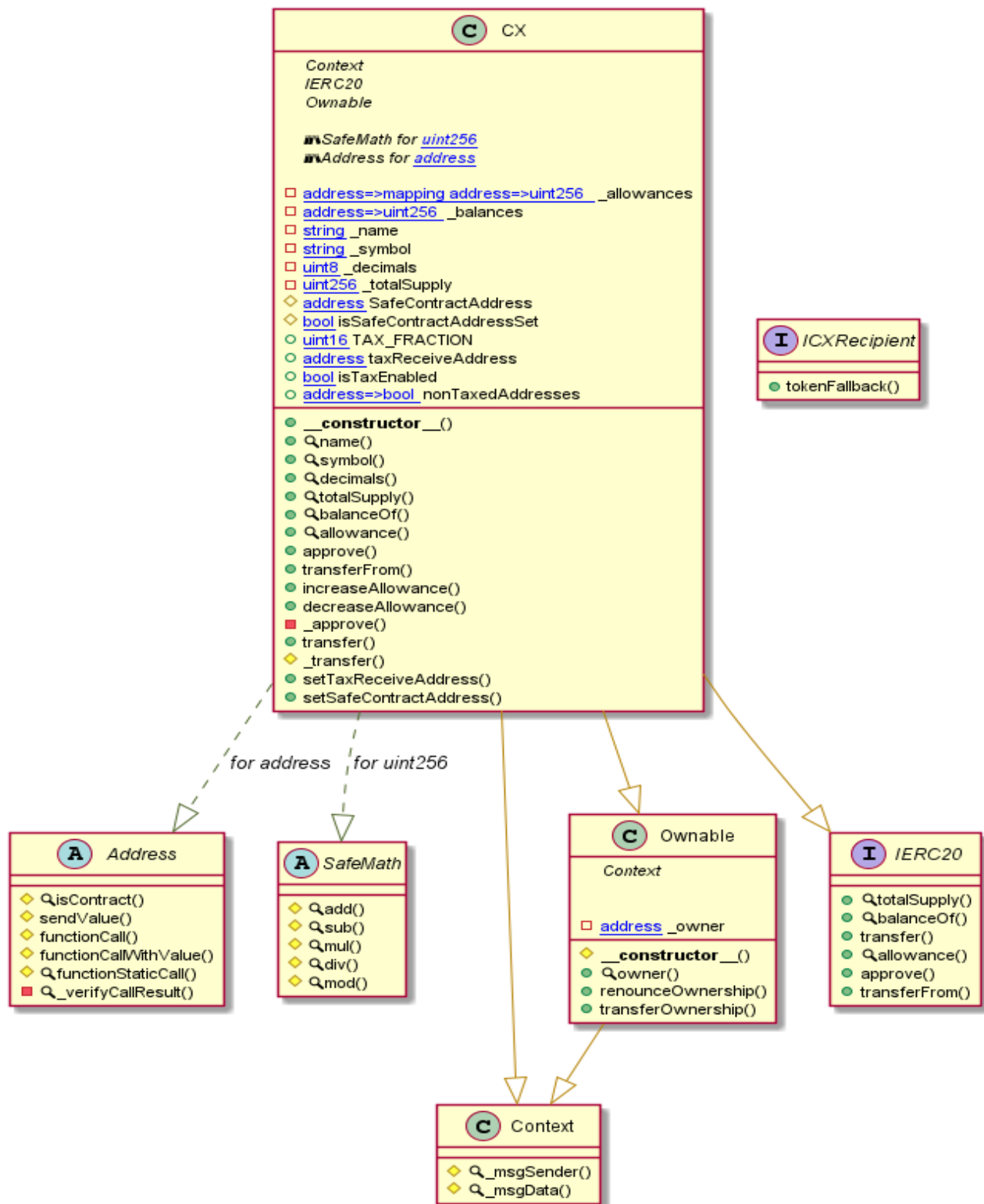
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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.

Appendix

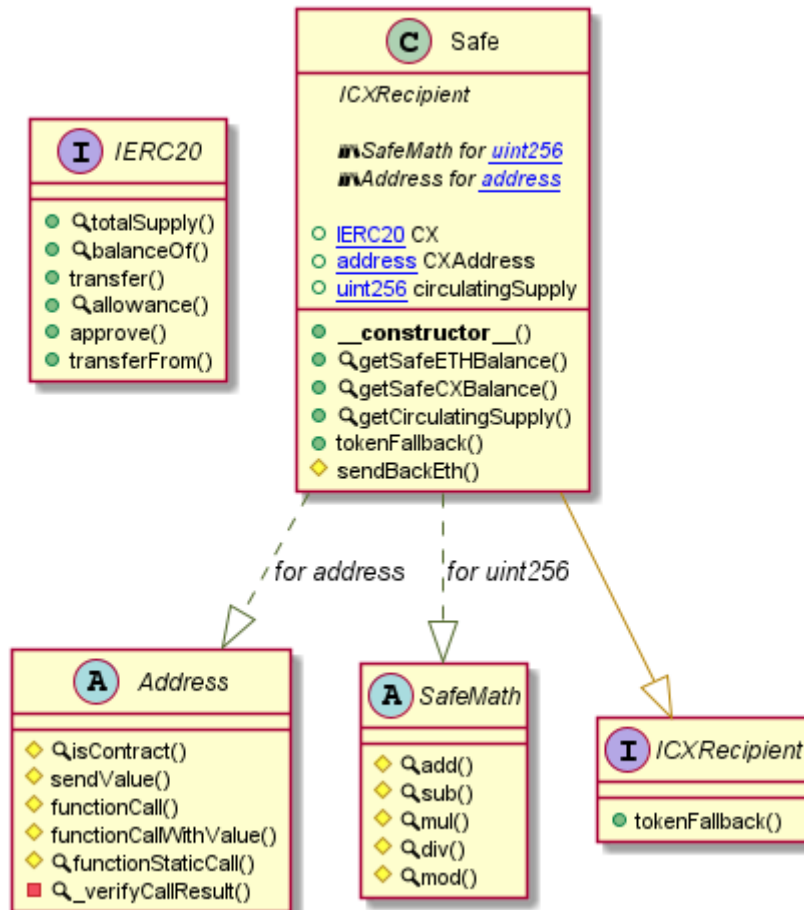
Code Flow Diagram - Cardex Token



ICXRecipient Diagram



Safe Diagram



Slither Results Log

Slither log >> CX.sol

```
INFO:Detectors:
CX.nonTaxedAddresses (CX.sol#468) is never initialized. It is used in:
- CX.transfer(address,uint256) (CX.sol#535-558)
- CX._transfer(address,address,uint256) (CX.sol#562-592)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
Reentrancy in CX.transfer(address,uint256) (CX.sol#535-558):
  External calls:
  - receiver.tokenFallback(payableTransferer,amount) (CX.sol#544)
  - receiver_scope_0.tokenFallback(payableTransferer_scope_1,newAmount) (CX.sol#552)
  State variables written after the call(s):
  - _transfer(_msgSender(),recipient,amount) (CX.sol#556)
    - _balances[sender] = _balances[sender].sub(amount,Mute: transfer amount exceeds balance) (CX.sol#568)
    - _balances[recipient] = _balances[recipient].add(amount) (CX.sol#569)
    - _balances[sender] = _balances[sender].sub(amount,Mute: transfer amount exceeds balance) (CX.sol#583)
    - _balances[recipient] = _balances[recipient].add(newAmount) (CX.sol#585)
    - _balances[taxReceiveAddress] = _balances[taxReceiveAddress].add(feeAmount) (CX.sol#587)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
CX.transfer(address,uint256) (CX.sol#535-558) ignores return value by receiver.tokenFallback(payableTransferer,amount) (CX.sol#544)
CX.transfer(address,uint256) (CX.sol#535-558) ignores return value by receiver_scope_0.tokenFallback(payableTransferer_scope_1,newAmount) (CX.sol#552)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
CX.allowance(address,address).owner (CX.sol#500) shadows:
- Ownable.owner() (CX.sol#333-335) (function)
CX._approve(address,address,uint256).owner (CX.sol#526) shadows:
- Ownable.owner() (CX.sol#333-335) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
CX.setTaxReceiveAddress(address)._taxReceiveAddress (CX.sol#594) lacks a zero-check on :
- taxReceiveAddress = _taxReceiveAddress (CX.sol#595)
CX.setSafeContractAddress(address)._SafeContractAddress (CX.sol#598) lacks a zero-check on :
- SafeContractAddress = _SafeContractAddress (CX.sol#600)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in CX.transfer(address,uint256) (CX.sol#535-558):
  External calls:
  - receiver.tokenFallback(payableTransferer,amount) (CX.sol#544)
  - receiver_scope_0.tokenFallback(payableTransferer_scope_1,newAmount) (CX.sol#552)
  Event emitted after the call(s):
  - Transfer(sender,recipient,amount) (CX.sol#572)
  - _transfer(_msgSender(),recipient,amount) (CX.sol#556)
  - Transfer(sender,recipient,newAmount) (CX.sol#590)
  - _transfer(_msgSender(),recipient,amount) (CX.sol#556)
  - Transfer(sender,taxReceiveAddress,feeAmount) (CX.sol#591)
  - _transfer(_msgSender(),recipient,amount) (CX.sol#556)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Address.isContract(address) (CX.sol#22-31) uses assembly
- INLINE ASM (CX.sol#29)
Address._verifyCallResult(bool,bytes,string) (CX.sol#143-160) uses assembly
- INLINE ASM (CX.sol#152-155)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
CX._approve(address,address,uint256) (CX.sol#526-533) compares to a boolean constant:
- require(bool,string)(isSafeContractAddressSet == true,CX: Safe Contract address must be set) (CX.sol#527)
CX.transfer(address,uint256) (CX.sol#535-558) compares to a boolean constant:
- require(bool,string)(isSafeContractAddressSet == true,CX: Safe Contract address must be set) (CX.sol#536)
CX.transfer(address,uint256) (CX.sol#535-558) compares to a boolean constant:
- nonTaxedAddresses[_msgSender()] == true || TAX_FRACTION == 0 || recipient == SafeContractAddress (CX.sol#540)
CX._transfer(address,address,uint256) (CX.sol#562-592) compares to a boolean constant:
- nonTaxedAddresses[sender] == true || TAX_FRACTION == 0 || recipient == SafeContractAddress (CX.sol#567)
CX._transfer(address,address,uint256) (CX.sol#562-592) compares to a boolean constant:
- require(bool,string)(isSafeContractAddressSet == true,CX: Safe Contract address must be set) (CX.sol#563)
CX.setSafeContractAddress(address) (CX.sol#598-602) compares to a boolean constant:
- require(bool,string)(isSafeContractAddressSet == false,Safe Contract Address already set) (CX.sol#599)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Address._verifyCallResult(bool,bytes,string) (CX.sol#143-160) is never used and should be removed
Address.functionCall(address,bytes) (CX.sol#75-77) is never used and should be removed
Address.functionCall(address,bytes,string) (CX.sol#85-87) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (CX.sol#100-102) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (CX.sol#110-117) is never used and should be removed
Address.functionStaticCall(address,bytes) (CX.sol#125-127) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (CX.sol#135-141) is never used and should be removed
Address.isContract(address) (CX.sol#22-31) is never used and should be removed
Address.sendValue(address,uint256) (CX.sol#49-55) is never used and should be removed
Context._msgData() (CX.sol#311-314) is never used and should be removed
SafeMath.mod(uint256,uint256) (CX.sol#284-286) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (CX.sol#300-303) is never used and should be removed
SafeMath.mul(uint256,uint256) (CX.sol#222-234) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (CX.sol#49-55):
- (success) = recipient.call{value: amount}() (CX.sol#53)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (CX.sol#110-117):
- (success,returndata) = target.call{value: value}(data) (CX.sol#115)
Low level call in Address.functionStaticCall(address,bytes,string) (CX.sol#135-141):
- (success,returndata) = target.staticcall(data) (CX.sol#139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter CX.setTaxReceiveAddress(address)._taxReceiveAddress (CX.sol#594) is not in mixedCase
Parameter CX.setSafeContractAddress(address)._SafeContractAddress (CX.sol#598) is not in mixedCase
Variable CX.SafeContractAddress (CX.sol#461) is not in mixedCase
Variable CX.TAX_FRACTION (CX.sol#464) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

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

```

INFO:Detectors:
Redundant expression "this (CX.sol#312)" inContext (CX.sol#306-315)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
CX.slitherConstructorVariables() (CX.sol#449-603) uses literals with too many digits:
- _totalSupply = 100000e18 (CX.sol#459)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
CX.TAX_FRACTION (CX.sol#464) should be constant
CX._decimals (CX.sol#458) should be constant
CX._name (CX.sol#456) should be constant
CX._symbol (CX.sol#457) should be constant
CX._totalSupply (CX.sol#459) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
owner() should be declared external:
- Ownable.owner() (CX.sol#333-335)
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (CX.sol#352-355)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (CX.sol#361-365)
name() should be declared external:
- CX.name() (CX.sol#479-481)
symbol() should be declared external:
- CX.symbol() (CX.sol#483-485)
decimals() should be declared external:
- CX.decimals() (CX.sol#487-489)
totalSupply() should be declared external:
- CX.totalSupply() (CX.sol#491-493)
allowance(address,address) should be declared external:
- CX.allowance(address,address) (CX.sol#500-502)
approve(address,uint256) should be declared external:
- CX.approve(address,uint256) (CX.sol#504-507)
transferFrom(address,address,uint256) should be declared external:
- CX.transferFrom(address,address,uint256) (CX.sol#509-513)
increaseAllowance(address,uint256) should be declared external:
- CX.increaseAllowance(address,uint256) (CX.sol#515-518)
decreaseAllowance(address,uint256) should be declared external:
- CX.decreaseAllowance(address,uint256) (CX.sol#520-523)
transfer(address,uint256) should be declared external:
- CX.transfer(address,uint256) (CX.sol#535-538)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:ICX.sol analyzed (7 contracts with 75 detectors), 57 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

```

Slither log >> ICXRecipient.sol

```

INFO:Slither:ICXRecipient.sol analyzed (1 contracts with 75 detectors), 0 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

```

Slither log >> Safe.sol

```

INFO:Detectors:
Safe.sendBackEth(address,uint256) (Safe.sol#430-440) sends eth to arbitrary user
Dangerous calls:
- swapInitiator.transfer(totalEth) (Safe.sol#437)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Safe.constructor(address)._CX (Safe.sol#400) lacks a zero-check on :
- CXAddress = _CX (Safe.sol#403)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Address.isContract(address) (Safe.sol#92-101) uses assembly
- INLINE ASM (Safe.sol#99)
Address.verifyCallResult(bool,bytes,string) (Safe.sol#213-230) uses assembly
- INLINE ASM (Safe.sol#222-225)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.verifyCallResult(bool,bytes,string) (Safe.sol#213-230) is never used and should be removed
Address.functionCall(address,bytes) (Safe.sol#145-147) is never used and should be removed
Address.functionCall(address,bytes,string) (Safe.sol#155-157) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Safe.sol#170-172) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Safe.sol#180-187) is never used and should be removed
Address.functionStaticCall(address,bytes) (Safe.sol#195-197) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (Safe.sol#205-211) is never used and should be removed
Address.isContract(address) (Safe.sol#92-101) is never used and should be removed
Address.sendValue(address,uint256) (Safe.sol#119-125) is never used and should be removed
SafeMath.add(uint256,uint256) (Safe.sol#244-249) is never used and should be removed
SafeMath.mod(uint256,uint256) (Safe.sol#354-356) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Safe.sol#370-373) is never used and should be removed
SafeMath.mul(uint256,uint256) (Safe.sol#292-304) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (Safe.sol#119-125):
- (success) = recipient.call{value: amount}() (Safe.sol#123)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Safe.sol#180-187):
- (success,returndata) = target.call{value: value}(data) (Safe.sol#185)
Low level call in Address.functionStaticCall(address,bytes,string) (Safe.sol#205-211):

```

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


```

low-level-calls-in-address-fallbacks#fallback(address,uint256) (Safe.sol#423-427)
- (success,returndata) = target.staticcall(data) (Safe.sol#209)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter Safe.tokenFallback(address,uint256)._from (Safe.sol#423) is not in mixedCase
Parameter Safe.tokenFallback(address,uint256)._value (Safe.sol#423) is not in mixedCase
Parameter Safe.sendBackEth(address,uint256)._from (Safe.sol#430) is not in mixedCase
Parameter Safe.sendBackEth(address,uint256)._tokenAmount (Safe.sol#430) is not in mixedCase
Variable Safe.CX (Safe.sol#391) is not in mixedCase
Variable Safe.CXAddress (Safe.sol#392) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in Safe.sendBackEth(address,uint256) (Safe.sol#430-440):
  External calls:
    - swapInitiator.transfer(totalEth) (Safe.sol#437)
  State variables written after the call(s):
    - circulatingSupply = circulatingSupply.sub(_tokenAmount) (Safe.sol#439)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
getSafeCXBalance() should be declared external:
  - Safe.getSafeCXBalance() (Safe.sol#414-416)
tokenFallback(address,uint256) should be declared external:
  - Safe.tokenFallback(address,uint256) (Safe.sol#423-427)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Safe.sol analyzed (5 contracts with 75 detectors), 29 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

```


Solidity static analysis

CX.sol

Security

Transaction origin:

INTERNAL ERROR in module Transaction origin: can't convert undefined to object
Pos: not available

Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: can't convert undefined to object
Pos: not available

Inline assembly:

INTERNAL ERROR in module Inline assembly: can't convert undefined to object
Pos: not available

Block timestamp:

INTERNAL ERROR in module Block timestamp: can't convert undefined to object
Pos: not available

Low level calls:

INTERNAL ERROR in module Low level calls: can't convert undefined to object
Pos: not available

Selfdestruct:

INTERNAL ERROR in module Selfdestruct: can't convert undefined to object
Pos: not available

Gas & Economy

This on local calls:

INTERNAL ERROR in module This on local calls: can't convert undefined to object
Pos: not available

Delete dynamic array:

INTERNAL ERROR in module Delete dynamic array: can't convert undefined to object
Pos: not available

For loop over dynamic array:

INTERNAL ERROR in module For loop over dynamic array: can't convert undefined to object
Pos: not available

Ether transfer in loop:

INTERNAL ERROR in module Ether transfer in loop: can't convert undefined to object
Pos: not available

ERC

ERC20:

INTERNAL ERROR in module ERC20: can't convert undefined to object
Pos: not available

Miscellaneous

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: can't convert undefined to object
Pos: not available

Similar variable names:

INTERNAL ERROR in module Similar variable names: can't convert undefined to object
Pos: not available

No return:

INTERNAL ERROR in module No return: can't convert undefined to object
Pos: not available

Guard conditions:

INTERNAL ERROR in module Guard conditions: can't convert undefined to object
Pos: not available

String length:

INTERNAL ERROR in module String length: can't convert undefined to object
Pos: not available

Security

Transaction origin:

INTERNAL ERROR in module Transaction origin: can't convert undefined to object
Pos: not available

Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: can't convert undefined to object
Pos: not available

Inline assembly:

INTERNAL ERROR in module Inline assembly: can't convert undefined to object
Pos: not available

Block timestamp:

INTERNAL ERROR in module Block timestamp: can't convert undefined to object
Pos: not available

Low level calls:

INTERNAL ERROR in module Low level calls: can't convert undefined to object
Pos: not available

Selfdestruct:

INTERNAL ERROR in module Selfdestruct: can't convert undefined to object
Pos: not available

Gas & Economy

This on local calls:

INTERNAL ERROR in module This on local calls: can't convert undefined to object
Pos: not available

Delete dynamic array:

INTERNAL ERROR in module Delete dynamic array: can't convert undefined to object
Pos: not available

For loop over dynamic array:

INTERNAL ERROR in module For loop over dynamic array: can't convert undefined to object
Pos: not available

Ether transfer in loop:

INTERNAL ERROR in module Ether transfer in loop: can't convert undefined to object
Pos: not available

ERC

ERC20:

INTERNAL ERROR in module ERC20: can't convert undefined to object
Pos: not available

Miscellaneous

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: can't convert undefined to object
Pos: not available

Similar variable names:

INTERNAL ERROR in module Similar variable names: can't convert undefined to object
Pos: not available

No return:

INTERNAL ERROR in module No return: can't convert undefined to object
Pos: not available

Guard conditions:

INTERNAL ERROR in module Guard conditions: can't convert undefined to object
Pos: not available

String length:

INTERNAL ERROR in module String length: can't convert undefined to object
Pos: not available

Solhint Linter

CX.sol

```
CX.sol:3:1: Error: Compiler version ^0.6.0 does not satisfy the r
semver requirement
CX.sol:461:5: Error: Explicitly mark visibility of state
CX.sol:461:13: Error: Variable name must be in mixedCase
CX.sol:462:5: Error: Explicitly mark visibility of state
CX.sol:464:19: Error: Variable name must be in mixedCase
CX.sol:598:37: Error: Variable name must be in mixedCase
```

ICXRecipient.sol

```
ICXRecipient.sol:3:1: Error: Compiler version ^0.6.0 does not satisfy
the r semver requirement
```

Safe.sol

```
Safe.sol:3:1: Error: Compiler version ^0.6.0 does not satisfy the r
semver requirement
Safe.sol:391:17: Error: Variable name must be in mixedCase
Safe.sol:392:18: Error: Variable name must be in mixedCase
Safe.sol:398:30: Error: Code contains empty blocks
Safe.sol:400:16: Error: Variable name must be in mixedCase
Safe.sol:439:5: Error: Possible reentrancy vulnerabilities. Avoid state
changes after transfer.
```

Software analysis result:

These 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