Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	CLIP convertible bond
Platform:	Ethereum
Website:	www.clip-e.co
Language:	Solidity
Date:	February 12th, 2024

Table of contents

Email: audit@EtherAuthority.io

`

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 CLIP convertible bond team to perform the Security audit of the CLIP convertible bond smart contracts 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 February 12th, 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 CLIPCB is a non-transferable token. It is possible to gradually convert from since to until the period to CLIP.
- The token is without any other custom functionality and without any ownership control, which makes it truly decentralized.
- There are 2 smart contracts that are included in the audit scope. And there are ERC20-standard smart contracts from the OpenZeppelin library. These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	**Code Review and Security Analysis Report for CLIP convertible bond Smart Contracts**
Platform	**Ethereum**
Language	**Solidity**
File 1	CLIPCB.sol
File 1 MD5 Hash	2B4B60FC997458D04E179ECB207F1E5A
File 2	CLIP.sol
File 2 MD5 Hash	9D19614BF9B98A7DD3FEFC065C957CAB
Audit Date	February 12th, 2024

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1: CLIPCB.sol **Tokenomics:** • Name: CLIP convertible bond • Symbol: CLIPCB • Decimals: 18 **Other Specifications:** • Mint the CLIPCB by depositing the CLIP. • Allow the transferable address for the claimer address. • Convert the CLIPCB to the CLIP. • renounce: Return the CLIPCB as the CLIP to the address that minted it. **Ownership Control:** • There are no owner functions, which makes it 100% decentralized.	**YES, This is valid.**
File 2: CLIP.sol • Name: Clip Token • Symbol: CLIP • Decimals: 18 • Total Supply: 100 million **Ownership Control:** • There are no owner functions, which makes it 100% decentralized.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **"Poor Secured"**. This token contract does not have any ownership control, hence it is **100% decentralized**.

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 1 critical, 0 high, 0 medium and 0 low and 3 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	Passed
	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	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Moderated
	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	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: **FAILED**

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	Yes
Cannot Sell	Yes
Max Tax	0%
Modify Tax	Not Detected
Fee Check	No
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	No
Pause Transfer?	Not Detected
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	Not Detected
Blacklist Check	No
Can Mint?	No
Is it Proxy?	No
Can Take Ownership?	Not Detected
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 2 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in CLIP convertible bond 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 CLIP convertible bond Token.

The CLIP convertible bond 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.

Code parts are **not well** commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a CLIP convertible bond Token smart contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. but 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 are used in this smart contracts infrastructure that are based on well known industry standard open source projects.

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

AS-IS overview

CLIPCB.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	mint	external	Passed	No Issue
3	allow	external	Passed	No Issue
4	allow	external	Passed	No Issue
5	claim	external	Passed	No Issue
6	renounce	external	Logical vulnerability	Refer Audit Findings
7	transfer	write	Passed	No Issue
8	transferFrom	write	Passed	No Issue
9	getClaimableTokens	read	Passed	No Issue
10	_beforeTokenTransfer	internal	Unused function parameter	Refer Audit Findings
13	_contains	internal	The unused internal function	Refer Audit Findings
14	_allow	internal	Passed	No Issue
15	decimals	read	Passed	No Issue
16	totalSupply	read	Passed	No Issue
17	balanceOf	read	Passed	No Issue
18	transfer	write	Passed	No Issue
19	allowance	read	Passed	No Issue
20	approve	write	Passed	No Issue
21	transferFrom	write	Passed	No Issue
22	_transfer	internal	Passed	No Issue
23	_update	internal	Passed	No Issue
24	_mint	internal	Passed	No Issue
25	_burn	internal	Passed	No Issue
26	_approve	internal	Passed	No Issue
27	_approve	internal	Passed	No Issue
28	_spendAllowance	internal	Passed	No Issue
29	name	read	Passed	No Issue
30	symbol	read	Passed	No Issue

CLIP.sol

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	transfer	write	Passed	No Issue
8	allowance	read	Passed	No Issue
9	approve	write	Passed	No Issue
10	transferFrom	write	Passed	No Issue
13	_transfer	internal	Passed	No Issue
14	_update	internal	Passed	No Issue
15	_mint	internal	Passed	No Issue
16	_burn	internal	Passed	No Issue
17	_approve	internal	Passed	No Issue
18	_approve	internal	Passed	No Issue
19	_spendAllowance	internal	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
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

(1) Logical vulnerability: **CLIPCB.sol**

```
/**
 * Return the CLIPCB as the CLIP to the address that minted it.
 * @param amount Amount of the CLIPCB.
 */
function renounce(uint256 amount) external {    infinite gas
    if (amount == 0) revert NoAmount();

    ClaimInfo storage originalClaimInfo = claimInfo[
        originalClaimer[msg.sender]
    ];
    if (amount > originalClaimInfo.amount - originalClaimInfo.claimed)
        revert OverAmount();

    _burn(msg.sender, amount);
    (bool success, ) = originalClaimInfo.from.call{value: amount}("");
    if (!success) revert TransferFailed();

    emit Renounce(originalClaimer[msg.sender], amount);
}
```

1. As per the comment, CLIP tokens should be transferred to the address that has minted them, Instead of that here coins are used to transfer.
2. There is no receive function, so coins will not be acceptable by this contract. Hence, the renounce function will always get reverted.

Resolution: We suggest checking renounce function functionality and correct the logic as per the requirement.

High Severity

No High severity vulnerabilities were found.

Medium

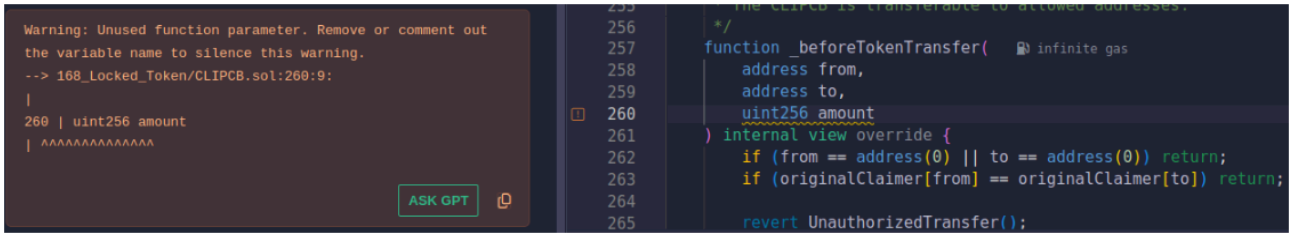
No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Unused function parameter: **CLIPCB.sol**



Unused function parameter. Remove or comment out the variable name to silence this warning.

(2) Spelling mistake: **CLIPCB.sol**

```
// Invalid mint destinaition.
error InvalidDestination();
```

Spelling mistakes.

1. “destinaition” word should be “destination”.

Resolution: Correct the spelling.

(3) The unused error is defined and unused internal function: **CLIPCB.sol**

Unused Error:

```
// Cannot renounce.
error CannotRenounce();
```

There is a "CannotRenounce" error defined in the smart contract body but not used anywhere.

Unused internal function:

```
/**
 * Whether list of the address contains the item address.
 */
function _contains(    undefined gas
    address[] memory list,
    address item
) internal pure returns (bool) {
    for (uint256 index = 0; index < list.length; index++) {
        if (list[index] == item) {
            return true;
        }
    }
    return false;
}
```

There is a "_contains" internal function that is defined but not used anywhere.

Resolution: Remove unused error and internal function from the smart contract body.

Centralization Risk

The CLIP convertible bond Token smart contract does not have any ownership control, hence it is 100% decentralized.

Therefore, there is **no** centralization risk.

Conclusion

We were given a contract code in the form of a file. And we have used all possible tests based on given objects as files. We had observed 1 critical and 3 informational issues in the smart contracts. **So, the smart contracts are ready for the mainnet deployment after resolving 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 the 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 smart contract, based on standard audit procedure scope, is **"Poor 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.

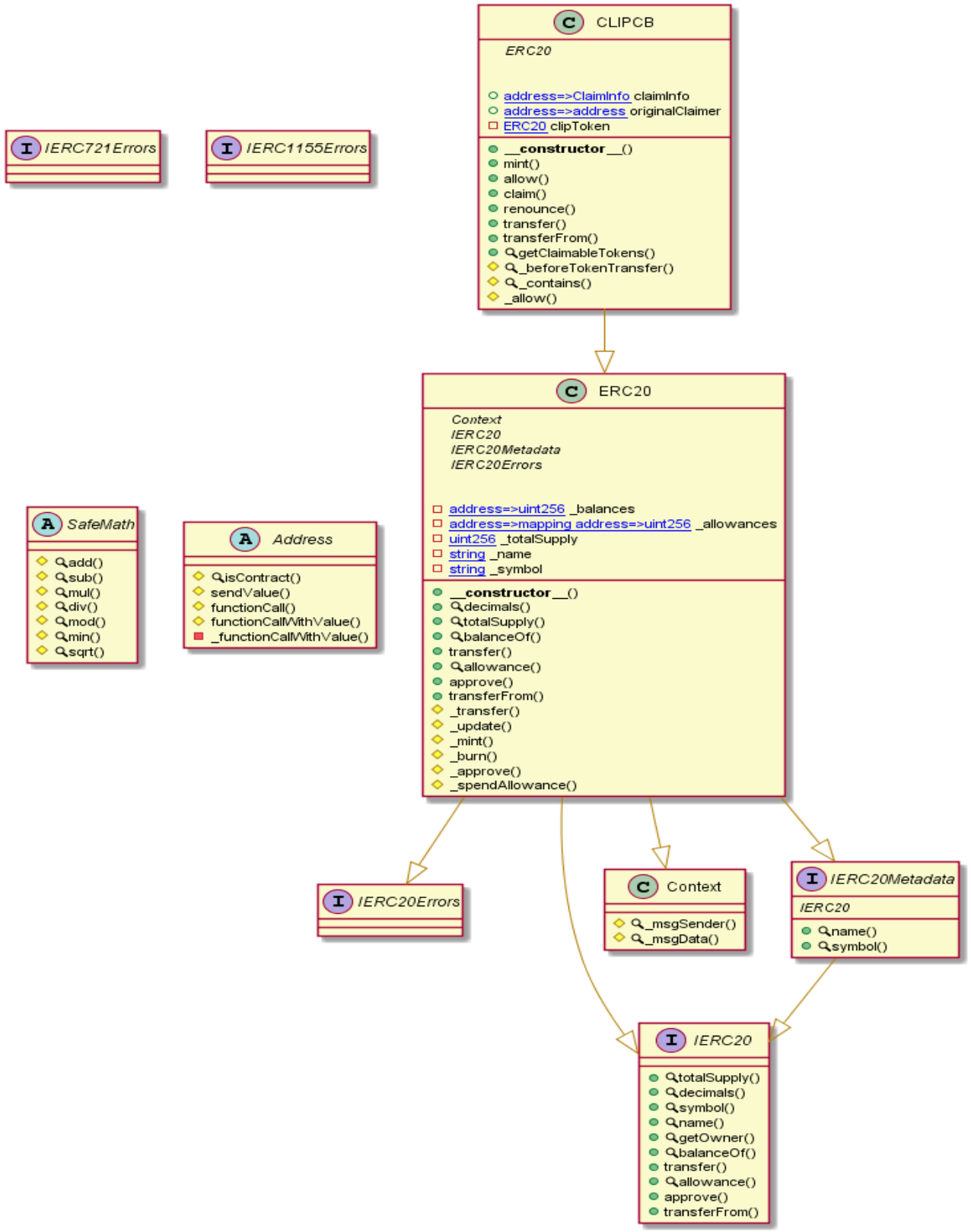
Technical Disclaimer

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.

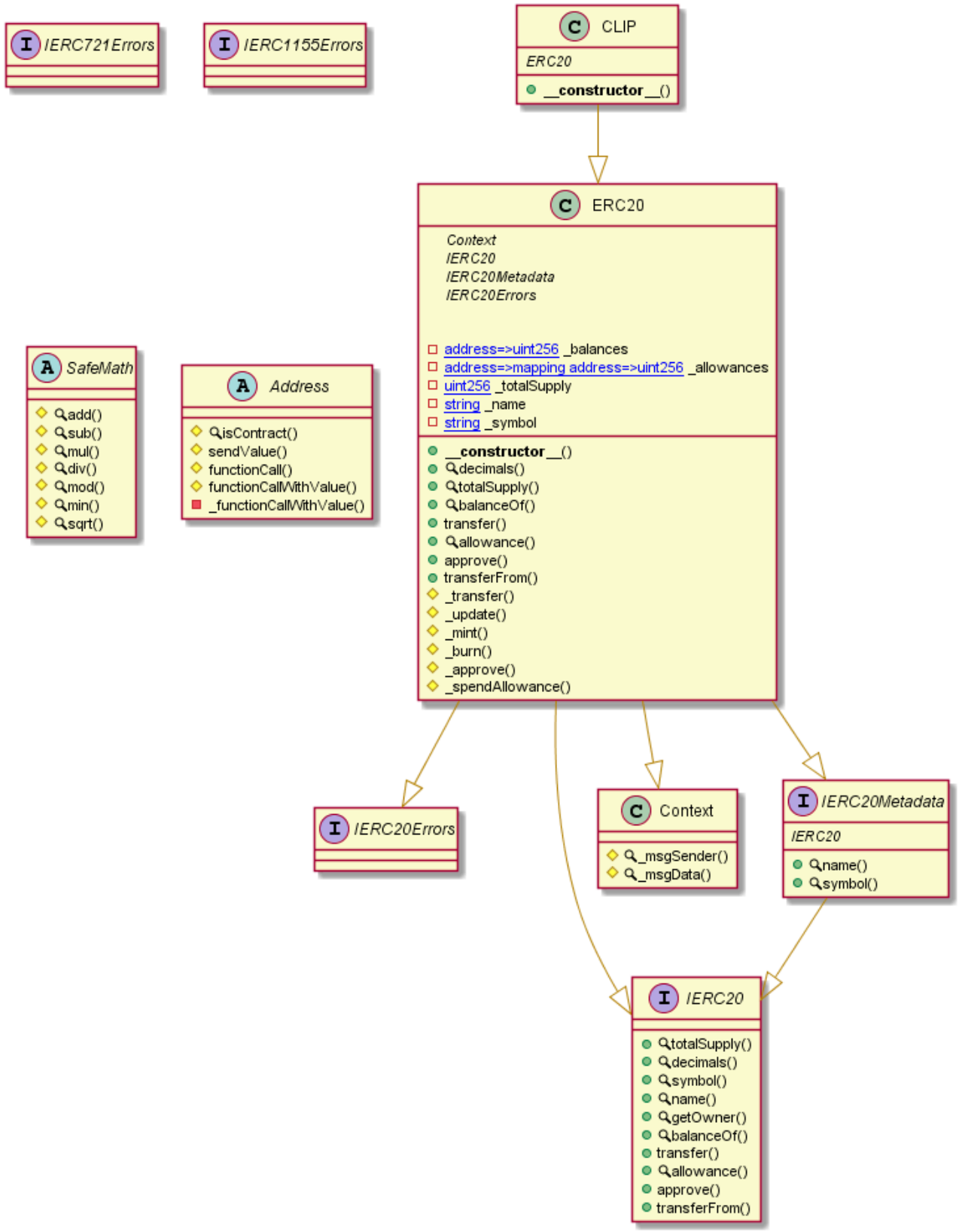
Appendix

Code Flow Diagram - CLIP convertible bond Token

CLIPCB Diagram



CLIP Diagram



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 did the analysis of the project altogether. Below are the results.

Slither Log >> CLIPCB.sol

```
INFO:Detectors:
Reentrancy in CLIPCB.mint(address,uint64,uint64,uint256) (CLIPCB.sol#557-575):
        External calls:
        - require(bool,string)(clipToken.transferFrom(msg.sender,address(this),clipAmount),CLIP transfer failed) (
CLIPCB.sol#565-568)
        State variables written after the call(s):
        - _mint(to,clipAmount) (CLIPCB.sol#570)
                - _balances[from] = fromBalance - value (CLIPCB.sol#378)
                - _balances[to] += value (CLIPCB.sol#390)
        - _mint(to,clipAmount) (CLIPCB.sol#570)
                - _totalSupply += value (CLIPCB.sol#370)
                - _totalSupply -= value (CLIPCB.sol#385)
        - claimInfo[to] = ClaimInfo(clipAmount,0,since,until,msg.sender) (CLIPCB.sol#571)
        - originalClaimer[to] = to (CLIPCB.sol#572)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in CLIPCB.claim(uint256) (CLIPCB.sol#605-622):
        External calls:
        - require(bool,string)(clipToken.transfer(msg.sender,amount),CLIP transfer failed) (CLIPCB.sol#619)
        Event emitted after the call(s):
        - Claim(originalClaimer[msg.sender],amount) (CLIPCB.sol#621)
Reentrancy in CLIPCB.mint(address,uint64,uint64,uint256) (CLIPCB.sol#557-575):
        External calls:
        - require(bool,string)(clipToken.transferFrom(msg.sender,address(this),clipAmount),CLIP transfer failed) (
CLIPCB.sol#565-568)
        Event emitted after the call(s):
        - Mint(to,clipAmount,since,until) (CLIPCB.sol#574)
        - Transfer(from,to,value) (CLIPCB.sol#394)
                - _mint(to,clipAmount) (CLIPCB.sol#570)
```

```
Reentrancy in CLIPCB.renounce(uint256) (CLIPCB.sol#628-642):
        External calls:
        - (success) = originalClaimInfo.from.call{value: amount}() (CLIPCB.sol#638)
        Event emitted after the call(s):
        - Renounce(originalClaimer[msg.sender],amount) (CLIPCB.sol#641)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
CLIPCB.getClaimableTokens(address) (CLIPCB.sol#689-707) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < originalClaimInfo.since (CLIPCB.sol#696)
        - amount > originalClaimInfo.amount (CLIPCB.sol#702)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (CLIPCB.sol#96-103) uses assembly
        - INLINE ASM (CLIPCB.sol#99-101)
Address._functionCallWithValue(address,bytes,uint256,string) (CLIPCB.sol#142-164) uses assembly
        - INLINE ASM (CLIPCB.sol#156-159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
ERC20._update(address,address,uint256) (CLIPCB.sol#367-395) has costly operations inside a loop:
        - _totalSupply += value (CLIPCB.sol#370)
ERC20._update(address,address,uint256) (CLIPCB.sol#367-395) has costly operations inside a loop:
        - _totalSupply -= value (CLIPCB.sol#385)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

```
INFO:Detectors:
Pragma version0.8.22 (CLIPCB.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.1
8.
solc-0.8.22 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (CLIPCB.sol#105-110):
        - (success) = recipient.call{value: amount}() (CLIPCB.sol#108)
```

```
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (CLIPCB.sol#142-164):
        - (success,returndata) = target.call{value: weiValue}(data) (CLIPCB.sol#150)
Low level call in CLIPCB.renounce(uint256) (CLIPCB.sol#628-642):
        - (success) = originalClaimInfo.from.call{value: amount}() (CLIPCB.sol#638)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Redundant expression "this (CLIPCB.sol#202)" inContext (CLIPCB.sol#196-205)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
CLIPCB (CLIPCB.sol#502-752) does not implement functions:
        - IERC20.getOwner() (CLIPCB.sol#175)
        - IERC20Metadata.name() (CLIPCB.sol#211)
        - IERC20Metadata.symbol() (CLIPCB.sol#216)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
INFO:Detectors:
CLIPCB.clipToken (CLIPCB.sol#522) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-im
mutable
INFO:Slither:CLIPCB.sol analyzed (10 contracts with 93 detectors), 37 result(s) found
```

Slither Log >> CLIP.sol

```
INFO:Detectors:
Address.isContract(address) (CLIP.sol#96-103) uses assembly
        - INLINE ASM (CLIP.sol#99-101)
Address._functionCallWithValue(address,bytes,uint256,string) (CLIP.sol#142-164) uses assembly
        - INLINE ASM (CLIP.sol#156-159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (CLIP.sol#142-164) is never used and should be remove
d
Address.functionCall(address,bytes) (CLIP.sol#112-114) is never used and should be removed
Address.functionCall(address,bytes,string) (CLIP.sol#116-122) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (CLIP.sol#124-130) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (CLIP.sol#132-140) is never used and should be removed
```

```
SafeMath.sub(uint256,uint256) (CLIP.sol#24-26) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (CLIP.sol#28-37) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.22 (CLIP.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

solc-0.8.22 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (CLIP.sol#105-110):
        - (success) = recipient.call{value: amount}() (CLIP.sol#108)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (CLIP.sol#142-164):
        - (success,returndata) = target.call{value: weiValue}(data) (CLIP.sol#150)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Redundant expression "this (CLIP.sol#202)" inContext (CLIP.sol#196-205)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
CLIP.constructor() (CLIP.sol#498-500) uses literals with too many digits:
        - _mint(msg.sender,100000000 * 10 ** decimals()) (CLIP.sol#499)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
CLIP (CLIP.sol#497-501) does not implement functions:
        - IERC20.getOwner() (CLIP.sol#175)
        - IERC20Metadata.name() (CLIP.sol#211)
        - IERC20Metadata.symbol() (CLIP.sol#216)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
INFO:Slither:CLIP.sol analyzed (10 contracts with 93 detectors), 29 result(s) found
```

Solidity Static Analysis

CLIPCB.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in CLIPCB.mint(address,uint64,uint64,uint256): Could potentially lead to re-entrancy vulnerability.

more

Pos: 93:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in CLIPCB.claim(uint256): Could potentially lead to re-entrancy vulnerability.

more

Pos: 146:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 241:13:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

more

Pos: 179:27:

Email: audit@EtherAuthority.io

Gas costs:

Gas requirement of function CLIPCB.allow 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: 123:4:

Gas costs:

Gas requirement of function CLIPCB.transfer 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: 190:4:

Gas costs:

Gas requirement of function CLIPCB.transferFrom 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: 211:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 137:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 275:8:

Constant/View/Pure functions:

CLIPCB._beforeTokenTransfer(address,address,uint256) : Is constant but potentially should not be.

more

Pos: 257:4:

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.

more

Pos: 216:8:

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: 240:25:

Solhint Linter

CLIPCB.sol

```
Compiler version 0.8.22 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path @openzeppelin/contracts/token/ERC20/ERC20.sol
is not allowed. Specify names to import individually or bind all
exports of the module into a name (import "path" as Name)
Pos: 1:3
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:77
Avoid making time-based decisions in your business logic
Pos: 22:100
Error message for require is too long
Pos: 9:193
Error message for require is too long
Pos: 9:215
Avoid making time-based decisions in your business logic
Pos: 13:236
Avoid making time-based decisions in your business logic
Pos: 14:240
Variable "amount" is unused
Pos: 9:259
```

CLIP.sol

```
Compiler version 0.8.22 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path @openzeppelin/contracts/token/ERC20/ERC20.sol
is not allowed. Specify names to import individually or bind all
exports of the module into a name (import "path" as Name)
Pos: 1:3
```

Software analysis result:

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

Email: audit@EtherAuthority.io