Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Secure Stash Token
Platform:	Binance Smart Chain
Website:	https://securestash.in
Language:	Solidity
Date:	October 14th, 2023

Table of contents

Introduction 4

Project Background 4

Audit Scope 4

Claimed Smart Contract Features 5

Audit Summary 7

Technical Quick Stats 8

Business Risk Analysis 9

Code Quality 10

Documentation 10

Use of Dependencies 10

AS-IS overview 11

Severity Definitions 13

Audit Findings 14

Conclusion 17

Our Methodology 18

Disclaimers 20

Appendix

- Code Flow Diagram 21
- Slither Results Log 22
- Solidity static analysis 24
- Solhint Linter 26

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

Email: audit@EtherAuthority.io

`

THIS IS 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.

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

Introduction

EtherAuthority was contracted by the Secure Stash team to perform the Security audit of the Secure Stash 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 October 14th, 2023 .

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

- Secure Stash (SST) is a BEP-20 token on Binance Smart Chain, used as collateral for Collateralized Low Volatile Tokens (C.L.V.T.), a low-volatile token
- The Secure Stash contract inherits ERC20, Ownable, ReentrancyGuard 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 Secure Stash Token Smart Contract**
Platform	**BSC**
File	SecureStash.sol
Github commit hash	1b3499cc7410645b2edd5ec61dc8e8985b7d6202
Updated Github commit hash	d3b6e5ce2a5bff69fc7756b96fc0ad65c5e47dfb
Audit Date	October 14th, 2023
Revised Audit Date	October 17th, 2023

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

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Secure Stash • Symbol: SST • Decimals: 18 • Maximum Supply: 100 million SST tokens • 0.5% tax applied only on DEX trades. • Tokens Release: 30 days	**YES, This is valid.**
Allocation: • Early Bird: 2.5% • Private Sale: 17.5% • Public Sale: 10% • Volatile Control: 50% • Exchange Listing: 10% • Ecosystem Development: 5% • Team: 4% • Reserves: 1%	**YES, This is valid.**
Other Specifications: • Tokens sold will receive an initial 4% allocation during the Token Generation Event (TGE). • The remaining tokens will be vested over a 48-month period and distributed periodically. • Similarly, other allocations will also begin with a 4% initial distribution, followed by monthly vesting for 48 months. • This means that out of 48 million SST tokens, up to 1 million tokens will be released each month over the course of 48 months from the commencement date.	**YES, This is valid.**

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

• New tokens, even though minted by the owner, are sent directly to the ecosystem wallet and not the owner's wallet.	
Ownership control: • Activate a sale. • Allocate tokens for sale. • Stop sale. • Add/Remove address is whitelisted. • Set the tax rates. • Add/Remove a DEX address. • Mint a new token. • Withdraw other vested tokens. • Update fund allocation address. • Locked allocation address.	**YES, This is valid.**

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

Audit Summary

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

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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, 0 high, 0 medium and 1 low and 2 very low level issues.

We confirm that all issues are fixed in the revised smart contract code.

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.

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

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	Passed
	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	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 set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

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

Business Risk Analysis

Category	Result
Buy Tax	0.5%
Sell Tax	0.5%
Can Buy	Passed
Can Sell	Passed
Max Tax	Passed
Modify Tax	Passed
Fee Check	Not Detected
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	Not Detected
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Whitelist?	Passed
Whitelist Check	Passed
Can Mint?	Passed
Is it Proxy?	Not Detected
Can Take Ownership?	Passed
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

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

Code Quality

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

The libraries in Secure Stash 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 Secure Stash Token.

The EtherAuthority 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 well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Secure Stash Token smart contract code in the form of a github web link.

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

Another source of information was its official project URL: https://securestash.in which provided rich information about the project architecture.

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.

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

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

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	Fixed
2	onlyWhenSaleActive	modifier	Passed	No Issue
3	onlyWhitelisted	modifier	Passed	No Issue
4	activateSale	write	access only Owner	No Issue
5	allocateTokensForSale	write	access only Owner	Fixed
6	stopSale	external	access only Owner	No Issue
7	addToWhitelist	external	access only Owner	No Issue
8	removeFromWhitelist	external	access only Owner	No Issue
9	setTaxRates	external	access only Owner	No Issue
10	addDex	write	access only Owner	No Issue
11	removeDex	write	access only Owner	No Issue
12	_updateCirculatingSupply	write	Passed	No Issue
13	transfer	write	Passed	No Issue
14	lockAllocationAddress	write	access only Owner	No Issue
15	updateFundAllocationAddress	write	access only Owner	Fixed
16	mint	write	access only Owner	No Issue
17	burn	write	Passed	No Issue
18	_updateUserSale	internal	Passed	No Issue
19	buySale	write	access only When Sale Active	No Issue
20	withdrawSaleVestedTokens	write	Passed	No Issue
21	withdrawOtherVestedTokens	write	access only Owner	No Issue
22	getOtherAllocationTypeAddress	read	Passed	No Issue
23	getOtherAllocationTypeBalances	read	Passed	No Issue
24	getTotalSupply	read	Passed	No Issue
25	getCirculatingSupply	read	Passed	No Issue
26	getTotalBurns	read	Passed	No Issue
27	onlyOwner	modifier	Passed	No Issue
28	owner	read	Passed	No Issue
29	_checkOwner	internal	Passed	No Issue
30	renounceOwnership	write	access only Owner	No Issue
31	transferOwnership	write	access only Owner	No Issue
32	_transferOwnership	internal	Passed	No Issue
33	totalSupply	read	Passed	No Issue
34	balanceOf	read	Passed	No Issue
35	name	read	Passed	No Issue
36	symbol	read	Passed	No Issue
37	decimals	read	Passed	No Issue
38	transfer	write	Passed	No Issue
39	allowance	read	Passed	No Issue
40	approve	write	Passed	No Issue
41	transferFrom	write	Passed	No Issue

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

42	_transfer	internal	Passed	No Issue
43	_update	internal	Passed	No Issue
44	_mint	internal	Passed	No Issue
45	_burn	internal	Passed	No Issue
46	_approve	internal	Passed	No Issue
47	_approve	internal	Passed	No Issue
48	_spendAllowance	internal	Passed	No Issue
49	nonReentrant	modifier	Passed	No Issue
50	_nonReentrantBefore	write	Passed	No Issue
51	_nonReentrantAfter	write	Passed	No Issue
52	_reentrancyGuardEntered	internal	Passed	No Issue

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

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have 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.

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

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

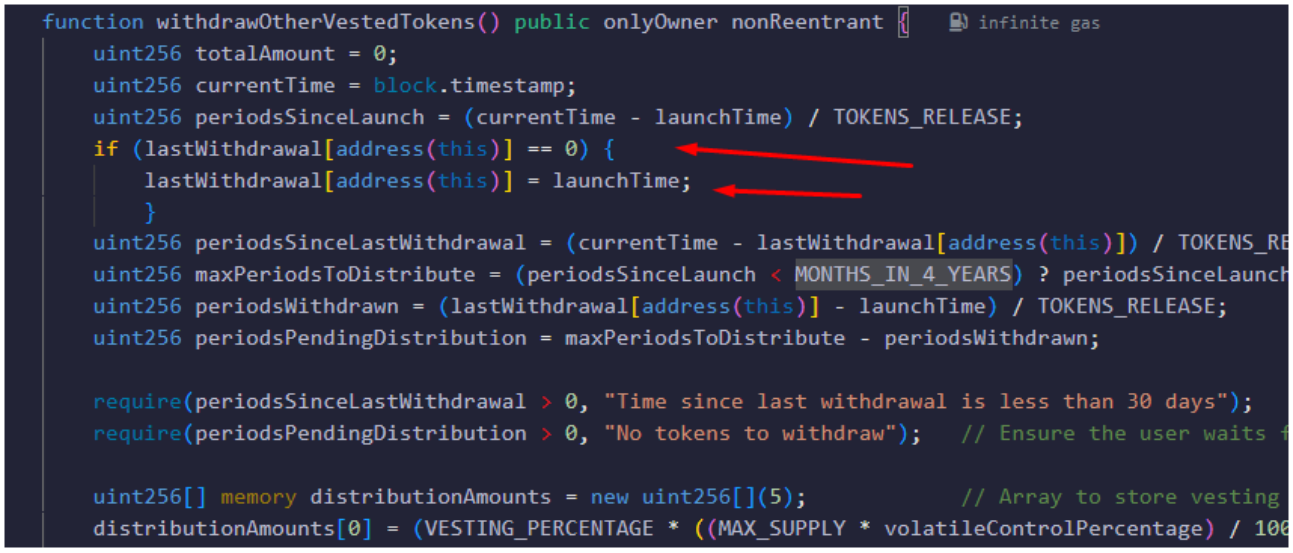
No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) The last withdrawal time:



```
function withdrawOtherVestedTokens() public onlyOwner nonReentrant {    infinite gas
    uint256 totalAmount = 0;
    uint256 currentTime = block.timestamp;
    uint256 periodsSinceLaunch = (currentTime - launchTime) / TOKENS_RELEASE;
    if (lastWithdrawal[address(this)] == 0) {
        lastWithdrawal[address(this)] = launchTime;
        }
    uint256 periodsSinceLastWithdrawal = (currentTime - lastWithdrawal[address(this)]) / TOKENS_RE
    uint256 maxPeriodsToDistribute = (periodsSinceLaunch < MONTHS_IN_4_YEARS) ? periodsSinceLaunch
    uint256 periodsWithdrawn = (lastWithdrawal[address(this)] - launchTime) / TOKENS_RELEASE;
    uint256 periodsPendingDistribution = maxPeriodsToDistribute - periodsWithdrawn;

    require(periodsSinceLastWithdrawal > 0, "Time since last withdrawal is less than 30 days");
    require(periodsPendingDistribution > 0, "No tokens to withdraw");   // Ensure the user waits f

    uint256[] memory distributionAmounts = new uint256[](5);            // Array to store vesting
    distributionAmounts[0] = (VESTING_PERCENTAGE * ((MAX_SUPPLY * volatileControlPercentage) / 100
```

In the constructor, lastWithdrawal time is already set with launchTime. In the withdrawOtherVestedTokens function, if a condition is applied to check lastWithdrawal[address(this)] time, which is not required.

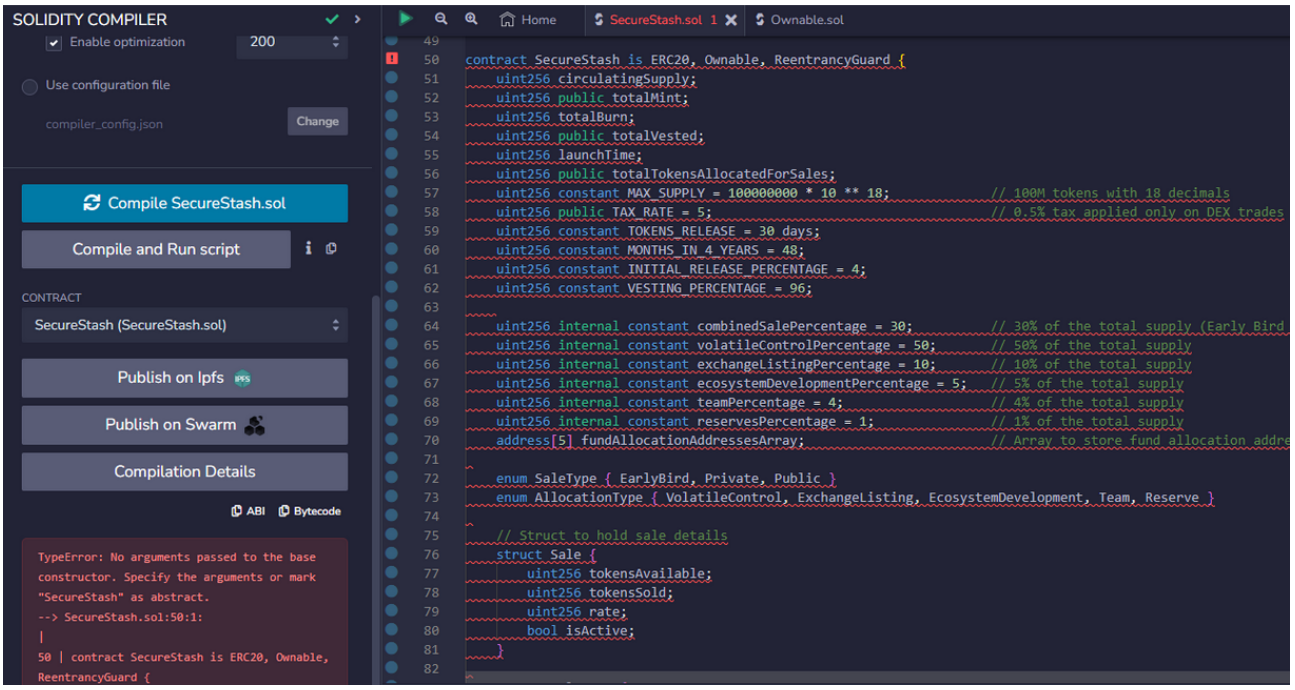
Resolution: We advise removing the if condition (lastWithdrawal[address(this)] == 0) from the withdrawOtherVestedTokens function.

Status: Fixed

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

Very Low / Informational / Best practices:

(1) Compilation Error:



No arguments passed to the base constructor. Specify the arguments or mark "SecureStash" as abstract.
Latest version of the Ownable contract from OpenZeppelin needs a param for initialization.

Resolution: We advise to check the latest OpenZeppelin's ownable contract and supply owner parameter for initialization.
Status: **Fixed**

(2) Critical operation lacks event log:
There are some events which need to be logged.
Events are:

1. updateFundAllocationAddress
2. allocateTokensForSale

Resolution: We suggest adding proper logs for the listed events.

Status: **Fixed**

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

Centralization

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

SecureStash.sol

- activateSale: Activate a sale by the owner.
- allocateTokensForSale: Allocate tokens for sale by the owner.
- stopSale: Stop sale by the owner.
- addToWhitelist: The address is whitelisted by the owner.
- removeFromWhitelist: Remove address from whitelist by the owner.
- setTaxRates: Tax rates can be set by the owner.
- addDex: Add a DEX address provided by the owner.
- removeDex: Remove a DEX address from the owner.
- lockAllocationAddress: An allocation address can be locked by the owner.
- updateFundAllocationAddress: The fund allocation address can be updated by the owner.
- mint: Mint a new token by the owner.
- withdrawOtherVestedTokens: Withdraw other vested tokens by the owner.

Ownable.sol

- renounceOwnership: Leaves the contract without owner. It will not be possible to call `onlyOwner` functions that can only be called by the current owner.
- transferOwnership: Transfers ownership of the contract to a new account can only be called by the current owner.

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

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

Conclusion

We were given a contract code in the form of a github web link. And we have used all possible tests based on given objects as files. We had observed 1 low and 2 informational issues in the smart contract. We confirm that all issues are fixed in the revised smart contract code. So, **it's good to go for the production**.

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 **"Secured".**

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

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of 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.

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

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, 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.

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

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract 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.

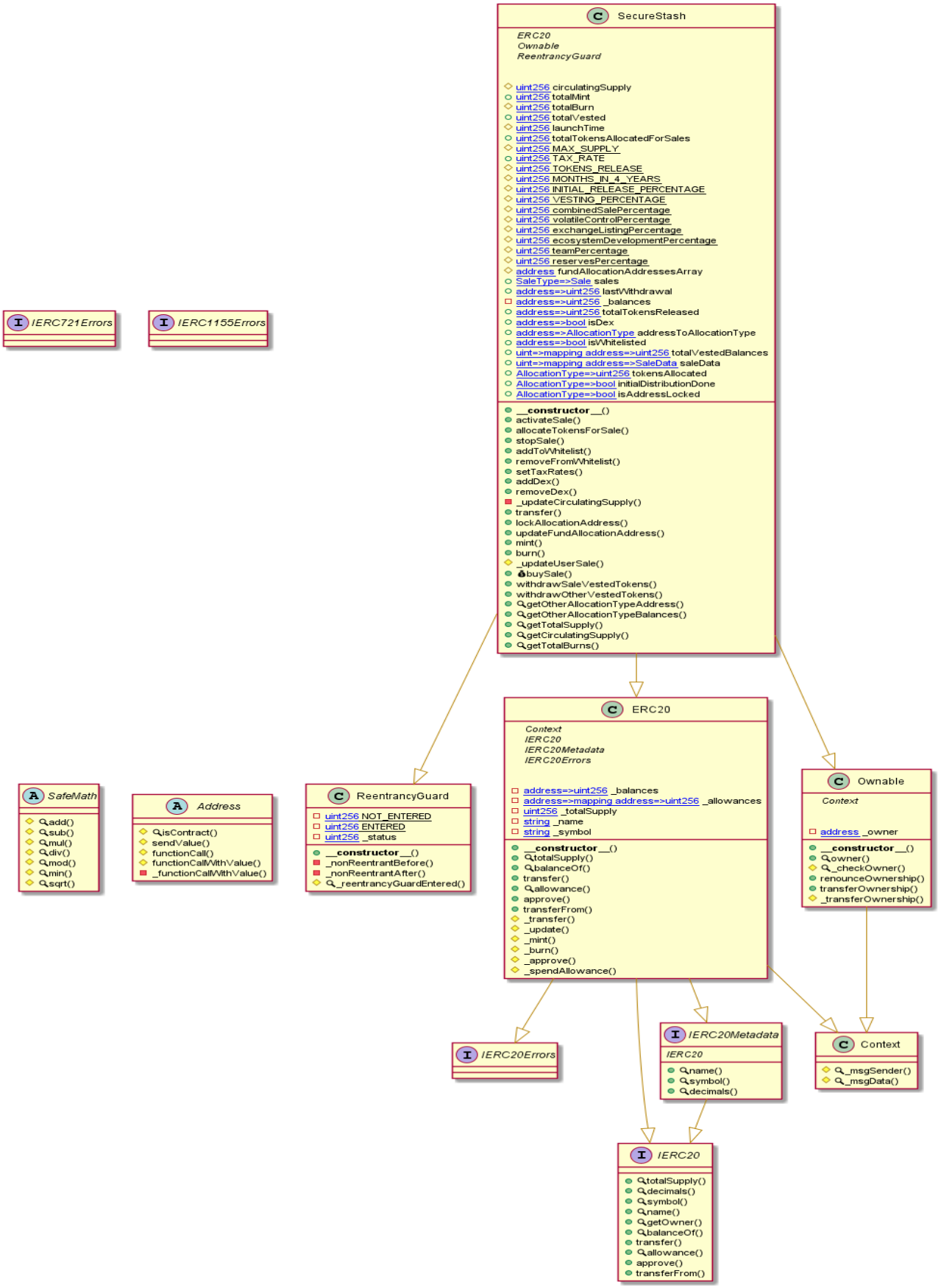
Technical Disclaimer

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.

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

Appendix

Code Flow Diagram - Secure Stash Token



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

Email: audit@EtherAuthority.io

Slither Results Log

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

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

Slither Log >> SecureStash.sol

```
SecureStash.activateSale(SecureStash.SaleType,uint256,uint256) (SecureStash.sol#788-800) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(tokensForSale <= totalTokensAllocatedForSales,Not enough tokens allocated for sale) (SecureStash
.sol#791)
SecureStash.allocateTokensForSale(uint256) (SecureStash.sol#802-814) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(maxAddableTokens > 0,No tokens available) (SecureStash.sol#810)
        - require(bool,string)(amount <= maxAddableTokens,Cannot add more than available tokens) (SecureStash.sol#811)
SecureStash.stopSale(SecureStash.SaleType) (SecureStash.sol#816-823) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(totalTokensAllocatedForSales > 0,No unsold tokens available) (SecureStash.sol#819)
SecureStash.transfer(address,uint256) (SecureStash.sol#853-869) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(balanceOf(msg.sender) >= amount,Insufficient balance) (SecureStash.sol#854)
SecureStash.mint(uint256) (SecureStash.sol#910-918) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((totalSupply() + amount) <= MAX_SUPPLY,Cannot exceed Maximum supply) (SecureStash.sol#911)
SecureStash.withdrawSaleVestedTokens() (SecureStash.sol#964-1007) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(periodsSinceLastWithdrawal > 0,Time since last withdrawal is less than 30 days) (SecureStash.sol
#973)
        - require(bool,string)(pendingRelease <= userSale.vestedTokens,Not enough vested tokens available in user balances) (Se
cureStash.sol#990)
        - require(bool,string)(balanceOf(address(this)) >= totalAmount,Not enough tokens in the contract) (SecureStash.sol#999)
        - require(bool,string)(totalAmount <= totalVested,Not enough tokens available to withdraw) (SecureStash.sol#1000)
        - (periodsSinceLaunch < MONTHS_IN_4_YEARS) (SecureStash.sol#972)
SecureStash.withdrawOtherVestedTokens() (SecureStash.sol#1009-1048) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(periodsSinceLastWithdrawal > 0,Time since last withdrawal is less than 30 days) (SecureStash.sol
#1021)
        - require(bool,string)(periodsPendingDistribution > 0,No tokens to withdraw) (SecureStash.sol#1022)
        - require(bool,string)(balanceOf(address(this)) >= amount,Not enough tokens in the contract) (SecureStash.sol#1036)
        - require(bool,string)(totalAmount <= totalVested,Not enough vested tokens available) (SecureStash.sol#1038)
        - (periodsSinceLaunch < MONTHS_IN_4_YEARS) (SecureStash.sol#1017)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.isContract(address) (SecureStash.sol#155-162) uses assembly
        - INLINE ASM (SecureStash.sol#158-160)
Address._functionCallWithValue(address,bytes,uint256,string) (SecureStash.sol#201-223) uses assembly
        - INLINE ASM (SecureStash.sol#215-218)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

SecureStash.updateFundAllocationAddress(SecureStash.AllocationType,address) (SecureStash.sol#877-908) has costly operations ins
ide a loop:
        - delete addressToAllocationType[fundAllocationAddressesArray[i_scope_0]] (SecureStash.sol#889)
ERC20._update(address,address,uint256) (SecureStash.sol#483-511) has costly operations inside a loop:
        - _totalSupply += value (SecureStash.sol#486)
ERC20._update(address,address,uint256) (SecureStash.sol#483-511) has costly operations inside a loop:
        - _totalSupply -= value (SecureStash.sol#501)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

Address._functionCallWithValue(address,bytes,uint256,string) (SecureStash.sol#201-223) is never used and should be removed
Address.functionCall(address,bytes) (SecureStash.sol#171-173) is never used and should be removed
Address.functionCall(address,bytes,string) (SecureStash.sol#175-181) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (SecureStash.sol#183-189) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (SecureStash.sol#191-199) is never used and should be removed
Address.isContract(address) (SecureStash.sol#155-162) is never used and should be removed
Address.sendValue(address,uint256) (SecureStash.sol#164-169) is never used and should be removed
Context._msgData() (SecureStash.sol#260-263) is never used and should be removed
ReentrancyGuard._reentrancyGuardEntered() (SecureStash.sol#321-323) is never used and should be removed
SafeMath.add(uint256,uint256) (SecureStash.sol#76-81) is never used and should be removed
SafeMath.div(uint256,uint256) (SecureStash.sol#109-111) is never used and should be removed
SafeMath.div(uint256,uint256,string) (SecureStash.sol#113-122) is never used and should be removed
SafeMath.min(uint256,uint256) (SecureStash.sol#137-139) is never used and should be removed
SafeMath.mod(uint256,uint256) (SecureStash.sol#124-126) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (SecureStash.sol#128-135) is never used and should be removed
SafeMath.mul(uint256,uint256) (SecureStash.sol#98-107) is never used and should be removed
SafeMath.sqrt(uint256) (SecureStash.sol#141-152) is never used and should be removed
SafeMath.sub(uint256,uint256) (SecureStash.sol#83-85) is never used and should be removed
```

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

```
SafeMath.sub(uint256,uint256) (SecureStash.sol#83-85) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (SecureStash.sol#87-96) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.21 (SecureStash.sol#45) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.
6/0.8.16
solc-0.8.21 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (SecureStash.sol#164-169):
        - (success) = recipient.call{value: amount}() (SecureStash.sol#167)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (SecureStash.sol#201-223):
        - (success,returndata) = target.call{value: weiValue}(data) (SecureStash.sol#209)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter SecureStash.addToWhitelist(address)._address (SecureStash.sol#826) is not in mixedCase
Parameter SecureStash.removeFromWhitelist(address)._address (SecureStash.sol#830) is not in mixedCase
Parameter SecureStash.setTaxRates(uint256)._taxRate (SecureStash.sol#834) is not in mixedCase
Parameter SecureStash.addDex(address)._dex (SecureStash.sol#840) is not in mixedCase
Parameter SecureStash.removeDex(address)._dex (SecureStash.sol#845) is not in mixedCase
Variable SecureStash.TAX_RATE (SecureStash.sol#691) is not in mixedCase
Constant SecureStash.combinedSalePercentage (SecureStash.sol#697) is not in UPPER_CASE_WITH_UNDERSCORES
Constant SecureStash.volatileControlPercentage (SecureStash.sol#698) is not in UPPER_CASE_WITH_UNDERSCORES
Constant SecureStash.exchangeListingPercentage (SecureStash.sol#699) is not in UPPER_CASE_WITH_UNDERSCORES
Constant SecureStash.ecosystemDevelopmentPercentage (SecureStash.sol#700) is not in UPPER_CASE_WITH_UNDERSCORES
Constant SecureStash.teamPercentage (SecureStash.sol#701) is not in UPPER_CASE_WITH_UNDERSCORES
Constant SecureStash.reservesPercentage (SecureStash.sol#702) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (SecureStash.sol#261)" inContext (SecureStash.sol#255-264)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

```
SecureStash.slitherConstructorConstantVariables() (SecureStash.sol#683-1081) uses literals with too many digits:
        - MAX_SUPPLY = 100000000 * 10 ** 18 (SecureStash.sol#690)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

SecureStash (SecureStash.sol#683-1081) does not implement functions:
        - IERC20Metadata.decimals() (SecureStash.sol#340)
        - IERC20.getOwner() (SecureStash.sol#234)
        - IERC20Metadata.name() (SecureStash.sol#330)
        - IERC20Metadata.symbol() (SecureStash.sol#335)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

SecureStash._balances (SecureStash.sol#731) is never used in SecureStash (SecureStash.sol#683-1081)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

SecureStash.launchTime (SecureStash.sol#688) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
SecureStash.sol analyzed (12 contracts with 84 detectors), 65 result(s) found
```

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

Solidity Static Analysis

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

SecureStash.sol

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: 122:22:

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: 378:30:

Gas costs:

Gas requirement of function SecureStash.withdrawSaleVestedTokens 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: 331:4:

Constant/View/Pure functions:

SecureStash.getOtherAllocationTypeBalances() : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 426:4:

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

Similar variable names:

SecureStash.activateSale(enum SecureStash.SaleType,uint256,uint256) : Variables have very similar names "sale" and "sales". Note: Modifiers are currently not considered by this static analysis.
Pos: 162:8:

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: 186:8:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 256:16:

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: 395:33:

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

Solhint Linter

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

SecureStash.sol

```
Compiler version ^0.8.20 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:44
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: 1:45
global import of path
@openzeppelin/contracts/security/ReentrancyGuard.sol is not allowed.
Specify names to import individually or bind all exports of the
module into a name (import "path" as Name)
Pos: 1:46
global import of path @openzeppelin/contracts/access/Ownable.sol is
not allowed. Specify names to import individually or bind all exports
of the module into a name (import "path" as Name)
Pos: 1:47
Contract has 20 states declarations but allowed no more than 15
Pos: 1:49
Explicitly mark visibility of state
Pos: 5:50
Explicitly mark visibility of state
Pos: 5:52
Explicitly mark visibility of state
Pos: 5:54
Explicitly mark visibility of state
Pos: 5:56
Variable name must be in mixedCase
Pos: 5:57
Explicitly mark visibility of state
Pos: 5:58
Explicitly mark visibility of state
Pos: 5:59
Explicitly mark visibility of state
Pos: 5:60
Explicitly mark visibility of state
Pos: 5:61
Constant name must be in capitalized SNAKE_CASE
Pos: 5:63
Constant name must be in capitalized SNAKE_CASE
Pos: 5:64
Constant name must be in capitalized SNAKE_CASE
Pos: 5:65
Constant name must be in capitalized SNAKE_CASE
```

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

```
Pos: 5:66
Constant name must be in capitalized SNAKE_CASE
Pos: 5:67
Constant name must be in capitalized SNAKE_CASE
Pos: 5:68
Explicitly mark visibility of state
Pos: 5:69
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:118
Avoid making time-based decisions in your business logic
Pos: 23:121
Error message for require is too long
Pos: 9:155
Error message for require is too long
Pos: 9:157
Avoid making time-based decisions in your business logic
Pos: 38:165
Error message for require is too long
Pos: 9:177
Error message for require is too long
Pos: 9:238
Error message for require is too long
Pos: 9:239
Error message for require is too long
Pos: 13:306
Error message for require is too long
Pos: 9:310
Avoid making time-based decisions in your business logic
Pos: 31:332
Error message for require is too long
Pos: 9:339
Error message for require is too long
Pos: 13:356
Error message for require is too long
Pos: 9:364
Error message for require is too long
Pos: 9:365
Error message for require is too long
Pos: 9:366
Avoid making time-based decisions in your business logic
Pos: 31:377
Error message for require is too long
Pos: 9:387
Error message for require is too long
Pos: 13:402
Error message for require is too long
Pos: 13:404
Variable name must be in mixedCase
Pos: 67:416
Variable name must be in mixedCase
Pos: 92:416
Variable name must be in mixedCase
Pos: 117:416
Variable name must be in mixedCase
Pos: 147:416
Variable name must be in mixedCase
Pos: 161:416
Variable name must be in mixedCase
```

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

```
Pos: 68:425
Variable name must be in mixedCase
Pos: 93:425
Variable name must be in mixedCase
Pos: 118:425
Variable name must be in mixedCase
Pos: 148:425
Variable name must be in mixedCase
Pos: 162:425
```

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

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