Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Uniswap Token
Website: uniswap.org
Platform: Ethereum
Language: Solidity
Date: March 14th, 2024

Table of contents

Introduction 4
Project Background 4
Audit Scope 5
Claimed Smart Contract Features 6
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 12
Audit Findings 13
Conclusion 16
Our Methodology 17
Disclaimers 19
Appendix

- Code Flow Diagram 20
- Slither Results Log 21
- Solidity static analysis 22
- Solhint Linter 24

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.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Uniswap Token smart contract from uniswap.org was audited extensively. 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 March 14th, 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

- This Solidity code defines a contract named `Uni`, which represents a token with functionalities like minting, transferring, approvals, and delegation. Let's break down its key components:
    - **SafeMath Library:** This library contains arithmetic functions (`add`, `sub`, `mul`, `div`, `mod`) with overflow/underflow checks to prevent common vulnerabilities like integer overflow/underflow.
    - **Uni Contract:**
        - **Token Metadata:** It defines constants for the token's name, symbol, and decimals.
        - **Total Supply:** The total token supply is set to 1 billion `UNI` tokens.
        - **Minting:** The contract allows minting new tokens by a designated `minter`. Minting is subject to certain conditions like a minimum time between mints, a minting cap, and a delay before minting starts.
        - **Token Balances and Allowances:** Balances and allowances are managed using mappings.
        - **Delegation:** The contract supports the delegation of voting power. Each address can delegate its voting power to another address.
        - **Checkpoint Mechanism:** It maintains a historical record of voting power for each address to support delegation.

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

        - **Functions:** Functions like `transfer`, `transferFrom`, `approve`, `permit`, `delegate`, and `getCurrentVotes` are provided for token transfers, approvals, and delegation.
    - **Modifiers:**
        - **require:** Used throughout the contract to validate conditions, reverting the transaction if the condition is not met.
    - **Events:** Events are emitted to log important contract state changes like transfers, approvals, and delegation changes.
- Overall, this contract provides a standard ERC-20 token functionality along with delegation and permit functionalities, implementing various safety measures to prevent common vulnerabilities.
- The Uniswap Token smart contracts offer functions such as updating minter addresses and minting new tokens.

Audit scope

Name	**Code Review and Security Analysis Report for Uniswap Token Smart Contract**
Platform	**Ethereum**
File	Uni.sol
Smart Contract Code	0x1f9840a85d5af5bf1d1762f925bdaddc4201f984
Audit Date	March 14th, 2024

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: Uniswap • Symbol: UNI • Decimals: 18 • Total Supply: 1 billion Uni tokens • Mint Cap: 2	**YES, This is valid.**
Ownership control: • Change the minter address. • Mint new tokens.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

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.

You are here

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

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

We found 0 critical, 0 high, 0 medium, 0 low, and 4 very low level issues.

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

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	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	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not 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%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
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?	Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	Not Detected
Blacklist Check	No
Can Mint?	Yes
Is it Proxy?	Not Detected
Can Take Ownership?	Not Detected
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 contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Uniswap 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 Uniswap Token.

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

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

Documentation

We were given a Uniswap Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are not well commented on. 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 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	No Issue
2	setMinter	external	Centralization risk, Missing zero address validation	Refer Audit Findings
3	mint	external	Centralization risk	Refer Audit Findings
4	allowance	external	Passed	No Issue
5	approve	external	Passed	No Issue
6	permit	external	Passed	No Issue
7	balanceOf	external	Passed	No Issue
8	transfer	external	Passed	No Issue
9	transferFrom	external	Passed	No Issue
10	delegate	write	Passed	No Issue
11	delegateBySig	write	Passed	No Issue
12	getCurrentVotes	read	Passed	No Issue
13	getPriorVotes	read	Passed	No Issue
14	_delegate	internal	Passed	No Issue
15	_transferTokens	internal	Passed	No Issue
16	_moveDelegates	internal	Passed	No Issue
17	_writeCheckpoint	internal	Passed	No Issue
18	safe32	internal	Passed	No Issue
19	safe96	internal	Passed	No Issue
20	add96	internal	Passed	No Issue
21	sub96	internal	Passed	No Issue
22	getChainId	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

No low-severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Use the latest solidity version:

```
pragma solidity ^0.5.16;
pragma experimental ABIEncoderV2;
```

Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use versions greater than 0.8.7.

(2) Use of Experimental Feature ABIEncoderV2 in Smart Contract:

```
Uni.sol:10:1: Warning: Experimental features are turned on. Do not
use experimental features on live deployments.
pragma experimental ABIEncoderV2;
^-------------------------------^
```

Experimental features are not recommended for live deployments due to their unproven stability and potential security vulnerabilities. This warning indicates a deviation from best

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

practices in smart contract development, which could compromise the reliability and security of the contract when deployed on a live blockchain network.

Resolution: Remove the line pragma experimental ABIEncoderV2; from the smart contract code. This will eliminate the use of the experimental feature, aligning the contract with established best practices.

(3) Missing zero address validation:

```
  function setMinter(address minter_) external {
        require(msg.sender == minter, "Uni::setMinter: only the
minter can change the minter address");
        emit MinterChanged(minter, minter_);
        minter = minter_;
    }
```

Detects missing zero address validation.

Resolution: We suggest first checking that the address is not zero.

(4) Centralization risk:

Some functions of this smart contract are only called by Minter (setMinter(), mint()).

Resolution: We suggest making smart contracts 100% decentralized.

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's private key would be compromised, then it would create trouble. The following are Admin functions:

Uni.sol

- setMinter: The minter address can be updated by the owner.
- mint: Mint the new tokens by the 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 Etherscan web links. And we have used all possible tests based on given objects as files. We observed 4 informational issues in the smart contracts. And those issues are not critical. 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 the maximum possible test cases to scan everything.

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

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

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

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

Documenting Results:

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

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation 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 is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

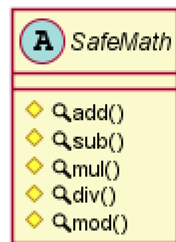
Technical Disclaimer

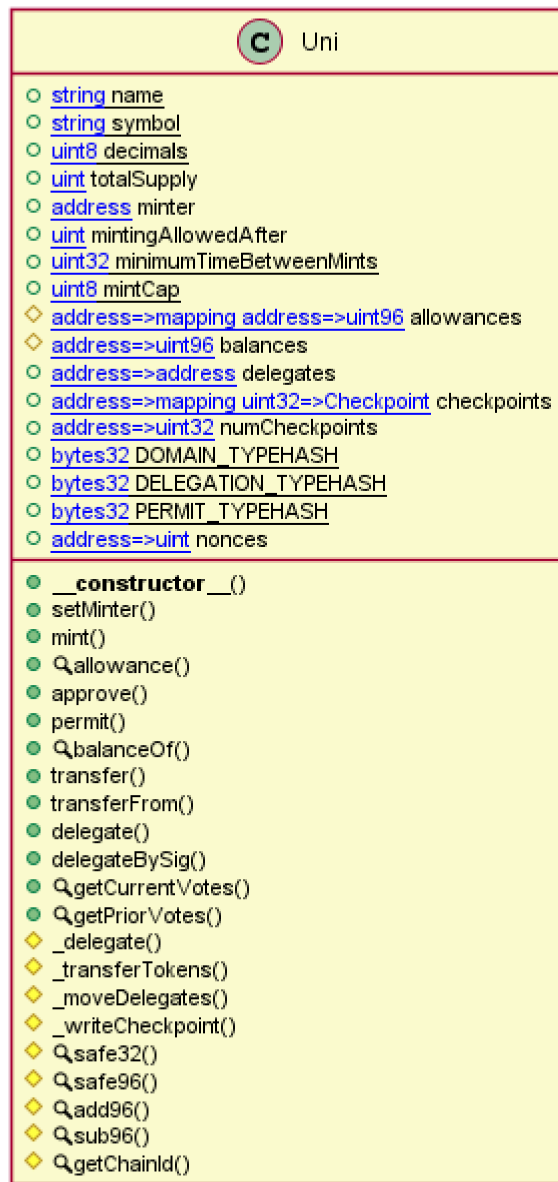
Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

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 - Uniswap 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 >> Uni.sol

```
Uni._writeCheckpoint(address,uint32,uint96,uint96) (Uni.sol#545-556) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (Uni.sol#548)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Uni.constructor(address,address,uint256).minter_ (Uni.sol#274) lacks a zero-check on :
                - minter = minter_ (Uni.sol#279)
Uni.setMinter(address).minter_ (Uni.sol#288) lacks a zero-check on :
                - minter = minter_ (Uni.sol#291)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Uni.constructor(address,address,uint256) (Uni.sol#274-282) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(mintingAllowedAfter_ >= block.timestamp,Uni::constructor: minting can only begin after deployme
nt) (Uni.sol#275)
Uni.mint(address,uint256) (Uni.sol#299-318) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= mintingAllowedAfter,Uni::mint: minting not allowed yet) (Uni.sol#301)
Uni.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (Uni.sol#362-381) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= deadline,Uni::permit: signature expired) (Uni.sol#376)
Uni.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (Uni.sol#444-453) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,Uni::delegateBySig: signature expired) (Uni.sol#451)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Uni.getChainId() (Uni.sol#579-583) uses assembly
        - INLINE ASM (Uni.sol#581)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
Uni.getChainId() (Uni.sol#579-583) uses assembly
        - INLINE ASM (Uni.sol#581)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

SafeMath.add(uint256,uint256,string) (Uni.sol#50-55) is never used and should be removed
SafeMath.mod(uint256,uint256) (Uni.sol#174-176) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Uni.sol#189-192) is never used and should be removed
SafeMath.mul(uint256,uint256,string) (Uni.sol#114-126) is never used and should be removed
SafeMath.sub(uint256,uint256) (Uni.sol#65-67) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (Uni.sol#77-82) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Constant Uni.minimumTimeBetweenMints (Uni.sol#215) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Uni.mintCap (Uni.sol#218) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
Uni.sol analyzed (2 contracts with 84 detectors), 16 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.

Uni.sol

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 583:8:

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: 307:43:

Gas costs:

Gas requirement of function Uni.setMinter 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: 290:4:

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:

Uni.(address,address,uint256) : Variables have very similar names "minter" and "minter_".
Pos: 281:17:

Similar variable names:

Uni.getCurrentVotes(address) : Variables have very similar names "checkpoints" and "nCheckpoints".
Pos: 464:55:

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: 577:8:

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: 495:36:

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.

Uni.sol

```
Error message for require is too long
Pos: 9:102
Constant name must be in capitalized SNAKE_CASE
Pos: 5:198
Constant name must be in capitalized SNAKE_CASE
Pos: 5:216
Constant name must be in capitalized SNAKE_CASE
Pos: 5:219
Error message for require is too long
Pos: 9:276
Avoid making time-based decisions in business logic
Pos: 41:276
Error message for require is too long
Pos: 9:302
Avoid making time-based decisions in business logic
Pos: 17:302
Error message for require is too long
Pos: 9:303
Avoid making time-based decisions in business logic
Pos: 44:306
Avoid making time-based decisions in business logic
Pos: 17:377
Error message for require is too long
Pos: 9:450
Error message for require is too long
Pos: 9:451
Error message for require is too long
Pos: 9:452
Avoid making time-based decisions in business logic
Pos: 17:452
Error message for require is too long
Pos: 9:474
Error message for require is too long
Pos: 9:518
```

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