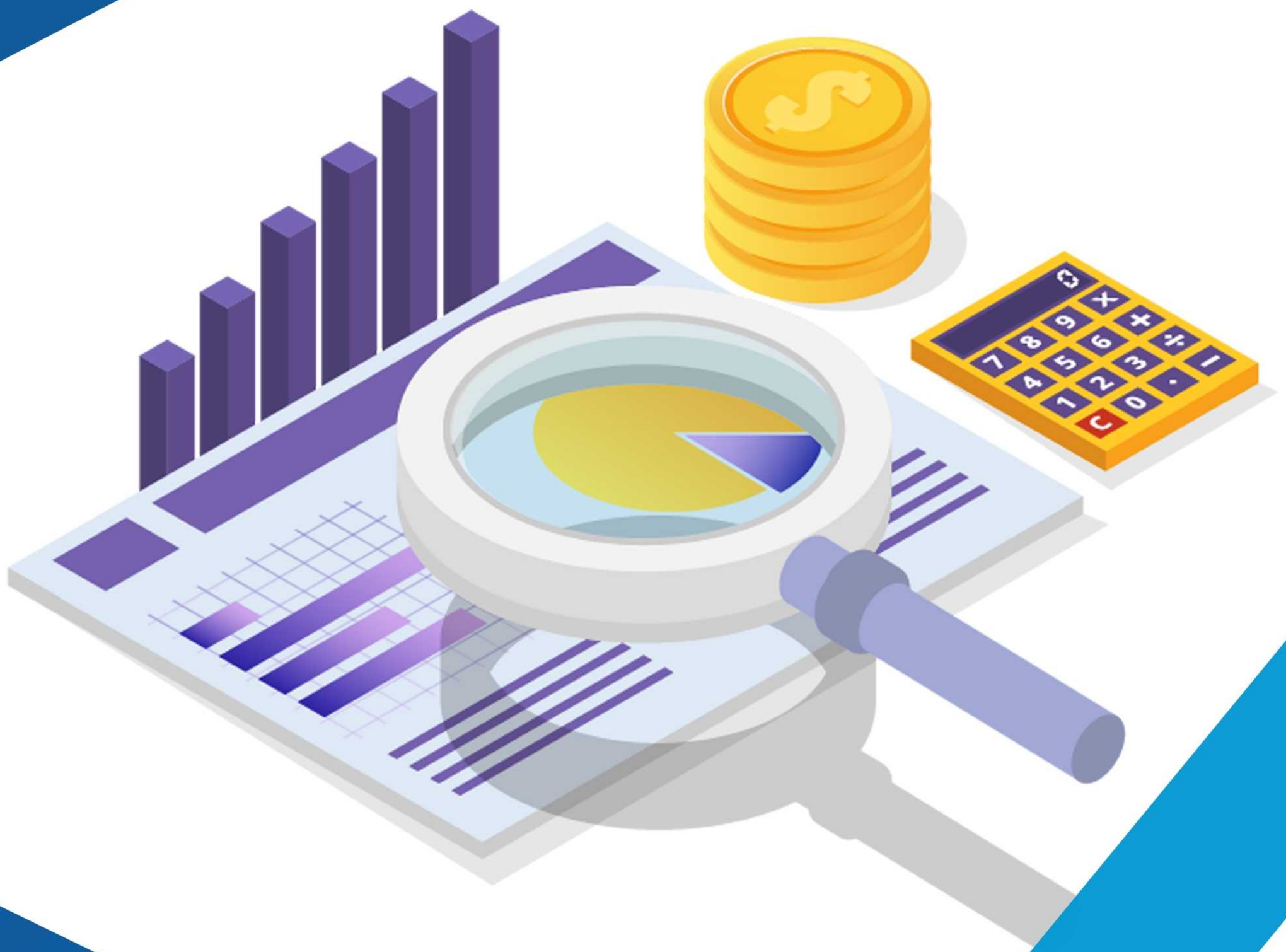
QuillAudits



Audit Report
May, 2021

AIDUS

AIDUS D-FUND PLATFORM

Contents

Scope of Audit

The scope of this audit was to analyze and document Aidus smart contract codebase for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis

Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption

In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	1	1
Closed	0	0	0	0

Introduction

During the period of **MAY 13th, 2021 to May 17th, 2021** - Quillhash Team performed a security audit for Aidus smart contracts.

The code for the audit was taken from following the official link:

https://etherscan.io/token/0xa957045A12D270e2eE0dcA9A3340c340e05d4670

Issues Found – Code Review / Manual Testing

High severity issues

None Found

Medium level severity issues

None Found

Low level severity issues

1. **Incorrect versions of Solidity**

Description:
solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statements.

Recommendation:
Deploy with any of the following Solidity versions:

- **0.5.16 - 0.5.17**
- **0.6.11 - 0.6.12**
- **0.7.5 - 0.7.6**

Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

Informational

Ambiguous error messages in Require statements.

Line	Code
516	require(_frozenAccount[target] != freeze, "Same as current");

522-523	require(!_frozenAccount[from], "error - frozen");
	
	require(!_frozenAccount[to], "error - frozen");

Description:
The error messages in require statements in the Aidus token contract are quite ambiguous.

While this makes it troublesome to detect the reason behind a particular function revert and can lead to functions that are difficult to understand, debug, and maintain, it also reduces the readability of the code.

Recommendation:
Error Messages should be more clear in every require statement in the contract.

The error message in line 516 should be more clear, for example:
require(_frozenAccount[target] != freeze, "This address has been frozen")

Automated Testing

Slither

```
enderphan@enderphan aidius % slither AIDUS.sol
INFO:Detectors:
ERC20Detailed.constructor(string,string,uint8).name (AIDUS.sol#545) shadows:
        - ERC20Detailed.name() (AIDUS.sol#554-556) (function)
ERC20Detailed.constructor(string,string,uint8).symbol (AIDUS.sol#545) shadows:
        - ERC20Detailed.symbol() (AIDUS.sol#561-563) (function)
ERC20Detailed.constructor(string,string,uint8).decimals (AIDUS.sol#545) shadows:
        - ERC20Detailed.decimals() (AIDUS.sol#568-570) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
SafeMath.div(uint256,uint256) (AIDUS.sol#133-140) is never used and should be removed
SafeMath.mod(uint256,uint256) (AIDUS.sol#166-169) is never used and should be removed
SafeMath.mul(uint256,uint256) (AIDUS.sol#116-128) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.5.10 (AIDUS.sol#5) allows old versions
Pragma version^0.5.10 (AIDUS.sol#80) allows old versions
Pragma version^0.5.10 (AIDUS.sol#106) allows old versions
Pragma version^0.5.10 (AIDUS.sol#174) allows old versions
Pragma version^0.5.10 (AIDUS.sol#363) allows old versions
Pragma version^0.5.10 (AIDUS.sol#391) allows old versions
Pragma version^0.5.10 (AIDUS.sol#434) allows old versions
Pragma version^0.5.10 (AIDUS.sol#479) allows old versions
Pragma version^0.5.10 (AIDUS.sol#502) allows old versions
Pragma version^0.5.10 (AIDUS.sol#531) allows old versions
Pragma version^0.5.10 (AIDUS.sol#575) allows old versions
solc-0.5.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter ERC20Frozenable.frozenAccount(address)._address (AIDUS.sol#511) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
owner() should be declared external:
        - Ownable.owner() (AIDUS.sol#29-31)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (AIDUS.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (AIDUS.sol#63-65)
totalSupply() should be declared external:
        - ERC20.totalSupply() (AIDUS.sol#202-204)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (AIDUS.sol#211-213)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (AIDUS.sol#221-223)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (AIDUS.sol#230-233)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (AIDUS.sol#244-250)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (AIDUS.sol#260-265)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (AIDUS.sol#277-283)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (AIDUS.sol#295-301)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (AIDUS.sol#375-377)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (AIDUS.sol#384-386)
addMinter(address) should be declared external:
        - MinterRole.addMinter(address) (AIDUS.sol#458-460)
renounceMinter() should be declared external:
        - MinterRole.renounceMinter() (AIDUS.sol#462-464)
mint(address,uint256) should be declared external:
        - ERC20Mintable.mint(address,uint256) (AIDUS.sol#494-497)
frozenAccount(address) should be declared external:
        - ERC20Frozenable.frozenAccount(address) (AIDUS.sol#511-513)
freezeAccount(address,bool) should be declared external:
        - ERC20Frozenable.freezeAccount(address,bool) (AIDUS.sol#515-519)
name() should be declared external:
        - ERC20Detailed.name() (AIDUS.sol#554-556)
symbol() should be declared external:
        - ERC20Detailed.symbol() (AIDUS.sol#561-563)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:AIDUS.sol analyzed (11 contracts with 75 detectors), 39 result(s) found
```

Solidity Static Analysis

Gas & Economy

Gas costs:

Gas requirement of function AidusToken.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: 230:4:

Gas costs:

Gas requirement of function ERC20.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: 230:4:

Gas costs:

Gas requirement of function ERC20Burnable.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: 230:4:

Gas costs:

Gas requirement of function ERC20Frozenable.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: 230:4:

Gas costs:

Gas requirement of function ERC20Mintable.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: 230:4:

Gas costs:

Gas requirement of function AidusToken.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: 260:4:

Gas costs:

Gas requirement of function ERC20.transferFrom is infinite: If the gas requirement of a function is higher than the block gas

Gas costs:

Gas requirement of function ERC20.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: 260:4:

Gas costs:

Gas requirement of function ERC20Burnable.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: 260:4:

Gas costs:

Gas requirement of function ERC20Frozenable.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: 260:4:

Gas costs:

Gas requirement of function ERC20Mintable.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: 260:4:

Gas costs:

Gas requirement of function AidusToken.increaseAllowance 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: 277:4:

Gas costs:

Gas requirement of function ERC20.increaseAllowance 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: 277:4:

Gas costs:

Gas requirement of function ERC20Burnable.increaseAllowance 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)

Miscellaneous

Constant/View/Pure functions:

ERC20Frozenable._transfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 521:4:

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: 37:8:

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: 72:8:

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: 125:8:

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: 135:8:

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: 146:8:

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.

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: 450:8:

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: 516:8:

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: 522:8:

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: 523:8:

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: 125:16:

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: 136:20:

Mythril

```
enderphan@enderphan aidius % myth a AIDUS.sol
The analysis was completed successfully. No issues were detected.

enderphan@enderphan aidius %
```

Oyente

```
INFO:root:contract aidus/ERC20.sol:ERC20:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                      1.8%
INFO:symExec:     Integer Underflow:                      False
INFO:symExec:     Integer Overflow:                       False
INFO:symExec:     Parity Multisig Bug 2:                  False
INFO:symExec:     Callstack Depth Attack Vulnerability:   False
INFO:symExec:     Transaction-Ordering Dependence (TOD):  False
INFO:symExec:     Timestamp Dependency:                   False
INFO:symExec:     Re-Entrancy Vulnerability:              False
INFO:symExec:   ====== Analysis Completed ======
INFO:root:contract aidus/ERC20Burnable.sol:ERC20Burnable:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                      1.4%
INFO:symExec:     Integer Underflow:                      False
INFO:symExec:     Integer Overflow:                       False
INFO:symExec:     Parity Multisig Bug 2:                  False
INFO:symExec:     Callstack Depth Attack Vulnerability:   False
INFO:symExec:     Transaction-Ordering Dependence (TOD):  False
INFO:symExec:     Timestamp Dependency:                   False
INFO:symExec:     Re-Entrancy Vulnerability:              False
INFO:symExec:   ====== Analysis Completed ======
INFO:root:contract aidus/ERC20Frozenable.sol:ERC20Frozenable:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                      0.8%
INFO:symExec:     Integer Underflow:                      False
INFO:symExec:     Integer Overflow:                       False
INFO:symExec:     Parity Multisig Bug 2:                  False
INFO:symExec:     Callstack Depth Attack Vulnerability:   False
INFO:symExec:     Transaction-Ordering Dependence (TOD):  False
INFO:symExec:     Timestamp Dependency:                   False
INFO:symExec:     Re-Entrancy Vulnerability:              False
INFO:symExec:   ====== Analysis Completed ======
INFO:root:contract aidus/ERC20Mintable.sol:ERC20Mintable:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                      1.2%
INFO:symExec:     Integer Underflow:                      False
INFO:symExec:     Integer Overflow:                       False
INFO:symExec:     Parity Multisig Bug 2:                  False
INFO:symExec:     Callstack Depth Attack Vulnerability:   False
INFO:symExec:     Transaction-Ordering Dependence (TOD):  False
INFO:symExec:     Timestamp Dependency:                   False
INFO:symExec:     Re-Entrancy Vulnerability:              False
INFO:symExec:   ====== Analysis Completed ======
```

Solhint Linter

Linter results:

```
ADIUS.sol:5:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:80:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:106:1: Error: Compiler version ^0.5.0 does not satisfy the r
semver requirement

ADIUS.sol:174:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:363:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:391:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:434:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:479:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:502:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:502:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:531:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:575:1: Error: Compiler version ^0.5.10 does not satisfy the r
semver requirement

ADIUS.sol:582:5: Error: Visibility modifier must be first in list of
modifiers
```

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.
No instances of Re-entrancy or Back-Door Entry were found in the contract.
During the process of the initial audit, a low severity issue was found. It is recommended to kindly go through the above mentioned details and fix the code accordingly.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **AIDUS platform**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Aidus Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

AIDUS

AIDUS D-FUND PLATFORM

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com