



**DIA TOKEN SMART CONTRACT AUDIT  
FOR DIA Data Limited & D.I.A e.V.**

**02.08.2020**

**Made in Germany by Chainsulting.de**



# Smart Contract Audit - DIA Token

1. Disclaimer.....	3
2. About the Project and Company .....	4
2.1 Project Overview:.....	5
3. Vulnerability & Risk Level .....	6
4. Auditing Strategy and Techniques Applied.....	7
4.1 Methodology .....	7
4.2 Tested Contract Files .....	8
4.3 Contract Specifications .....	8
5. Summary of Smart Contract.....	9
5.1 Visualized Dependencies .....	9
5.2 Functions.....	10
5.4 States.....	12
6. Test Suite Results.....	13
6.1 Mythril Classic & MYTHX Security Audit .....	13
6.1.1 A floating pragma is set.....	13
7. SWC Attacks.....	14
8. Gnosis CMM DEX.....	18
8.1 Gnosis CMM & Python implementation .....	19
8.1.1 Wildcard import.....	19
9. Executive Summary.....	20
10. Deployed Smart Contract.....	20



## 1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of D.I.A e.V. . If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description	Author
0.1 (31.07.2020)	Layout	Y. Heinze
0.5 (31.07.2020)	Automated Security Testing Manual Security Testing	Y. Heinze
1.0 (01.08.2020)	Gnosis CMM Python Script Testing	Y. Heinze
1.1 (02.08.2020)	Summary and Recommendation	Y. Heinze

## 2. About the Project and Company

### Company address:

President of the board: Michael Weber

Members of the board: Paul Claudius, Martin Hobler

DIA Data Limited  
63/66 Hatton Garden  
London, EC1N 8LE  
United Kingdom  
Company number 12308863



D.I.A. e.V. (Association)  
Baarerstrasse 10  
6300 Zug  
Switzerland  
Association Register No.: CHE-447.804.203



## 2.1 Project Overview:

DIA (Decentralized Information Asset) is an open-source, financial information platform that utilizes crypto economic incentives to source and validate data. Market actors can supply, share and use financial and digital asset data.

As a Swiss-based non-profit association, it is DIA's mission to democratize financial data, similar to what Wikipedia has done in the broader information space with regard to central encyclopedias.

DIA data sources and methodologies are transparent and publicly accessible to everyone. DIA uses crypto-economic incentives for its stakeholders to validate data sources when be added and throughout their usage.

The DIA platform is an ecosystem that employs a governance token. DIA is managed by a decentralized community of DIA token-holders and their delegates. DIA governance tokens can be used to drive the collection of data, validate the data, vote on association relevant decisions and incentivize the building of the DIA platform itself.

### 3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

## 4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

### 4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
  - i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
  - ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
  - iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
  - i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
  - ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.



## 4.2 Tested Contract Files

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (SHA256)
DIAToken.sol	A49DF241E7D403925732175EA0ACCE1A918CC370B6E323F2587B4EF735D95130

Source:

<https://raw.githubusercontent.com/chainsulting/Smart-Contract-Security-Audits/master/DIA%20Token/DIAToken.sol>

## 4.3 Contract Specifications

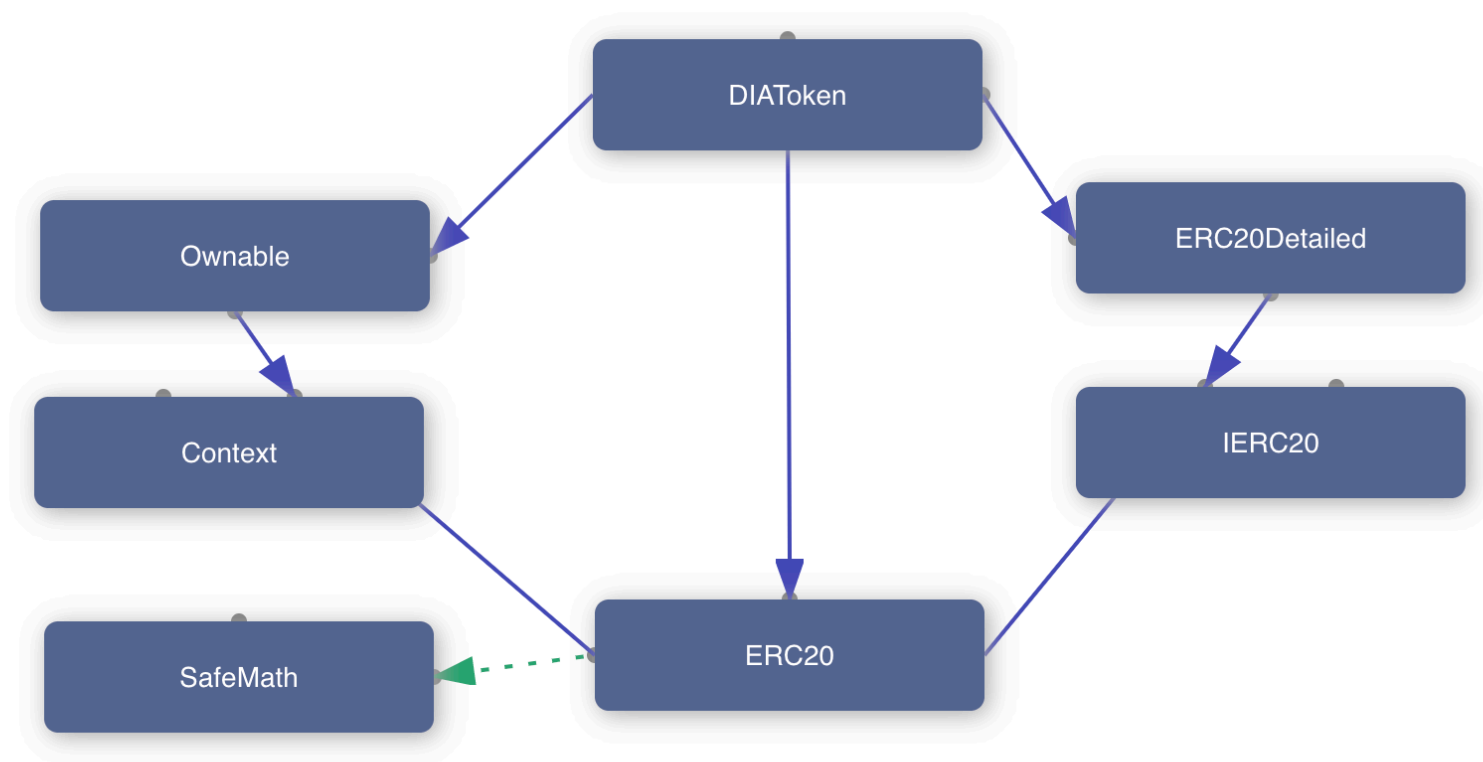
Language	Solidity
Token Standard	ERC20
Most Used Framework	OpenZeppelin
Compiler Version	0.5.0
Burn Function	Yes
Mint Function	No (Fixed total supply)
Lock Mechanism	No
Vesting Function	No
Ticker Symbol	DIA
Total Supply	200 000 000
Decimals	18





## 5. Summary of Smart Contract

### 5.1 Visualized Dependencies



## 5.2 Functions

contract	func	visibility	modifiers	stateMutability
Context	"constructor"	internal		
Context	_msgSender	internal		view
Context	_msgData	internal		view
IERC20	totalSupply	external		view
IERC20	balanceOf	external		view
IERC20	transfer	external		
IERC20	allowance	external		view
IERC20	approve	external		
IERC20	transferFrom	external		
SafeMath	add	internal		pure
SafeMath	sub	internal		pure
SafeMath	sub	internal		pure
SafeMath	mul	internal		pure
SafeMath	div	internal		pure
SafeMath	div	internal		pure
SafeMath	mod	internal		pure
SafeMath	mod	internal		pure
ERC20	totalSupply	public		view
ERC20	balanceOf	public		view
ERC20	transfer	public		
ERC20	allowance	public		view

ERC20	approve	public		
ERC20	transferFrom	public		
ERC20	increaseAllowance	public		
ERC20	decreaseAllowance	public		
ERC20	_transfer	internal		
ERC20	_mint	internal		
ERC20	_burn	internal		
ERC20	_approve	internal		
ERC20	_burnFrom	internal		
ERC20Detailed	"constructor"	public		
ERC20Detailed	name	public		view
ERC20Detailed	symbol	public		view
ERC20Detailed	decimals	public		view
Ownable	"constructor"	internal		
Ownable	owner	public		view
Ownable	isOwner	public		view
Ownable	renounceOwnership	public	onlyOwner	
Ownable	transferOwnership	public	onlyOwner	
Ownable	_transferOwnership	internal		
DIAToken	"constructor"	public	ERC20Detailed	
DIAToken	addAuditHash	public	onlyOwner	

## 5.3 Modifiers

contract	modifier
Ownable	onlyOwner

## 5.4 States

contract	state	type	visibility	isConst
ERC20	_balances	mapping	private	false
ERC20	_allowances	mapping	private	false
ERC20	_totalSupply	uint256	private	false
ERC20Detailed	_name	string	private	false
ERC20Detailed	_symbol	string	private	false
ERC20Detailed	_decimals	uint8	private	false
Ownable	_owner	address	private	false
DIAToken	auditHashes	mapping	public	false

## 6. Test Suite Results

The DIA Token is part of the DIA Smart Contract and this one was audited. All the functions and state variables are well commented using the natspec documentation for the functions which is good to understand quickly how everything is supposed to work.

### 6.1 Mythril Classic & MYTHX Security Audit

Mythril Classic is an open-source security analysis tool for Ethereum smart contracts. It uses concolic analysis, taint analysis and control flow checking to detect a variety of security vulnerabilities.

#### Detected Vulnerabilities

Informational: 0

Low: 1

Medium: 0

High: 0

Critical: 0

#### 6.1.1 A floating pragma is set.

Severity: LOW

Code: SWC-103

File(s) affected: DIAToken.sol


Attack / Description	Code Snippet	Result/Recommendation
The current pragma Solidity directive is "^0.5.0". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is	Line: 1 pragma solidity ^0.5.0;	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.  Pragma solidity 0.5.0

especially important if you rely on bytecode-level verification of the code.		
--	--	--

**Result:** The analysis was completed successfully. No major issues were detected.

## 7. SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✓
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✓
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✓
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✓
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✓
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✓

ID	Title	Relationships	Test Result
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	

ID	Title	Relationships	Test Result
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✓
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✓
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✓
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✓
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✓
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✓
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✓
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✓



ID	Title	Relationships	Test Result
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✓
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✗
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✓
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✓
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓

Sources:

<https://smartcontractsecurity.github.io/SWC-registry>

<https://dasp.co>

<https://github.com/chainsulting/Smart-Contract-Security-Audits>

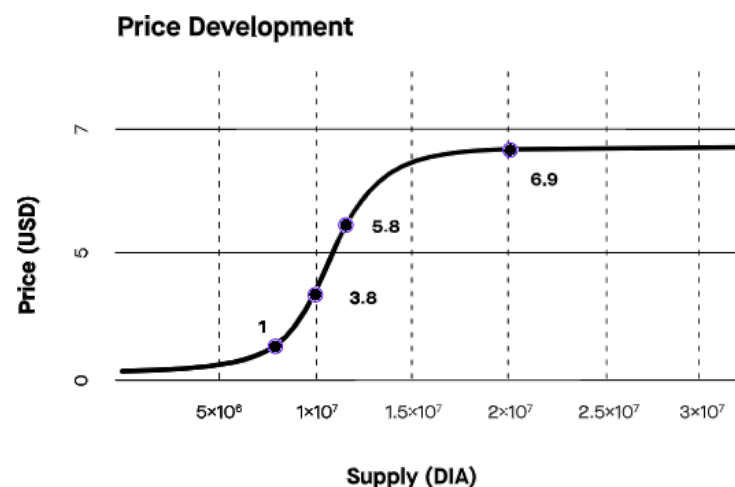
[https://consensys.github.io/smart-contract-best-practices/known\\_attacks](https://consensys.github.io/smart-contract-best-practices/known_attacks)

## 8. Gnosis CMM DEX

### Approach of DIA Token Sale

The intention of the token sale is a price curve, a so called "bonding curve". In general, a bonding curve is a function  $y = f(x)$  where  $x$  is the "number" of the token and  $y$  the price. The curve follows a sigmoid shape, so that the price starts at (\$0.05) and ends up at (\$6.94). For the sale the DIA Team is utilizing the Gnosis DEX and Custom Market Maker (CMM) to deploy an order book and following that curve in steps.

The custom market maker (CMM) allows to set multiple limit orders at custom price brackets and passively provide liquidity on Gnosis Protocol. In summary, the CMM works by placing multiple limit orders between a token pair. These limit orders occur at different price ranges referred to as brackets. When token price moves through the price points specified by the brackets, the CMM automatically provides liquidity to the countertrade.



Sources:

<https://docs.gnosis.io/protocol/docs/tutorial-cmm/>

<https://docs.gnosis.io/protocol/docs/intro-cmm/>

<https://github.com/gnosis/dex-liquidity-provision>

## 8.1 Gnosis CMM & Python implementation

As part of the DIA Token sale, we done an audit of the Gnosis CMM implementation. Therefor we audit the python script implementation, which will coordinate the "bonding curve". All the functions and variables are well commented, which is good to understand quickly how everything is supposed to work. We used Pylint to do a code analysis and checked the logic of the script manually.

### 8.1.1 Wildcard import

Severity: LOW

Code: W0401

File(s) affected: dia-trades.py

Attack / Description	Code Snippet	Result/Recommendation
Occurs when <code>from</code> module <code>import *</code> is detected.	Line: 3  <code>from decimal import *</code>	When an import statement in the pattern of <code>from MODULE import *</code> is used it may become difficult for a Python validator to detect undefined names in the program that imported the module. Furthermore, as a general best practice, import statements should be as specific as possible and should only import what they need.

Source:

<https://raw.githubusercontent.com/chainsulting/Smart-Contract-Security-Audits/master/DIA%20Token/dia-trades.py>

## 9. Executive Summary

A majority of the code was standard and copied from widely-used and reviewed contracts and as a result, a lot of the code was reviewed before. It correctly implemented widely-used and reviewed contracts for safe mathematical operations. The audit identified no major security vulnerabilities, at the moment of audit. We noted that a majority of the functions were self-explanatory, and standard documentation tags (such as `@dev`, `@param`, and `@returns`) were included.

Regards the gnosis custom market maker modification via python script, to coordinate the bonding curve, no logic mistakes been detected, even no known vulnerabilities.

## 10. Deployed Smart Contract

<https://etherscan.io/token/0x84cA8bc7997272c7CfB4D0Cd3D55cd942B3c9419>

