



**DIA**

**Oracle v2 on Polygon**

**SMART CONTRACT AUDIT**

**01.02.2022**

**Made in Germany by Chainsulting.de**



## Table of contents

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 Metrics / CallGraph.....	9
4.4 Metrics / Source Lines & Risk.....	10
4.5 Metrics / Capabilities .....	11
5. Scope of Work .....	13
5.1 Manual and Automated Vulnerability Test.....	14
5.1.1 Missing natspec documentation.....	15
5.2. SWC Attacks .....	16
5.3 Verify claims .....	20
6. Executive Summary.....	21
7. Deployed Smart Contract .....	21

## 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
0.1 (29.01.2022)	Layout
0.4 (30.01.2022)	Automated Security Testing Manual Security Testing
0.5 (31.01.2022)	Verify Claims and Test Deployment
0.6 (01.01.2022)	Testing SWC Checks
0.9 (01.01.2022)	Summary and Recommendation
1.0 (01.01.2022)	Final document
1.1 (01.01.2022)	Added deployed contract



## 2. About the Project and Company

### Company address:

D.I.A. e.V. (Association)  
Baarerstrasse 10  
6300 Zug  
Switzerland

**Website:** <https://diadata.org>

**Twitter:** [https://twitter.com/diadata\\_org](https://twitter.com/diadata_org)

**Medium:** [https://medium.com/@diadata\\_org](https://medium.com/@diadata_org)

**Telegram:** [https://t.me/DIAdata\\_org](https://t.me/DIAdata_org)

**LinkedIn:** <https://www.linkedin.com/company/diadata-org>

**GitHub:** <https://github.com/diadata-org/diadata>

**Reddit:** <https://www.reddit.com/user/DIAdata>

**YouTube:** [https://www.youtube.com/c/DIAdata\\_org](https://www.youtube.com/c/DIAdata_org)

## 2.1 Project Overview

DIA (Decentralised Information Asset) is an open-source oracle platform that enables market actors to source, supply and share trustable data. DIA aims to be an ecosystem for open financial data in a financial smart contract ecosystem, to bring together data analysts, data providers and data users. In general, DIA provides a reliable and verifiable bridge between off-chain data from various sources and on-chain smart contracts that can be used to build a variety of financial DApps. DIA is the governance token of the platform. It is currently based on ERC-20 Ethereum protocol. The project was founded in 2018, while the token supply was made available to the public during the bonding curve sale from Aug. 3 through Aug. 17, 2020, where 10.2 million tokens were sold.

### Who Are the Founders of DIA?

The DIA association was co-founded by a group of a dozen people, though Paul Claudius, Michael Weber and Samuel Brack are the leaders. Claudius is the face of the project and its lead advocate, sometimes also mentioned as a CBO. He has a masters degree in international management from ESCP Europe and a bachelors in business and economics from Passau University. Apart from working on DIA, he is also a co-founder and CEO of BlockState AG and c ventures. Before crypto, he had worked as director for a nutrition company called nu3. Weber is the project's CEO. He holds a masters in management from ESCP Business School and an equivalent to a bachelors in economics and physics from University of Cologne. He has worked in several banks and financial institutions before turning to crypto, where he founded such projects as Goodcoin, myLucy and BlockState. Samuel Brack serves DIA in the role of CTO. Like both Claudius and Weber, he shares the same position at BlockState. He has a masters degree in computer science from Humboldt University of Berlin, where as of January 2020, he is still studying for his PhD.

### What Makes DIA Unique?

DIA aims to become the Wikipedia of financial data. It specifically addresses the problem of dated/unverified/hard to access data in the world of finance and crypto, especially DeFi, while proposing to solve it via system of financial incentives for users to keep the flow of open-source, validated data streams to the oracles up and running. The current design of oracles, DIA argues, is non-transparent, difficult to scale and vulnerable to attack. The DIA governance token will be used to fund data collection, data validation, voting on governance decisions and to incentivize the development of the platform. Users can stake DIA tokens to incentivise new data to appear on the platform, but access to historical data though DIA is free.

### 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.

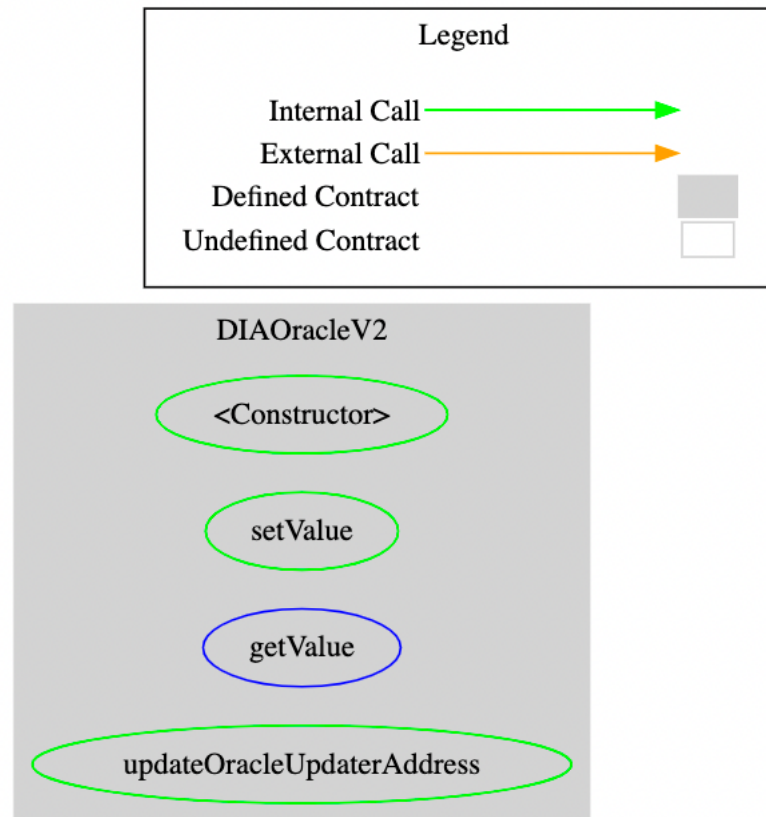
## 4.2 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 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

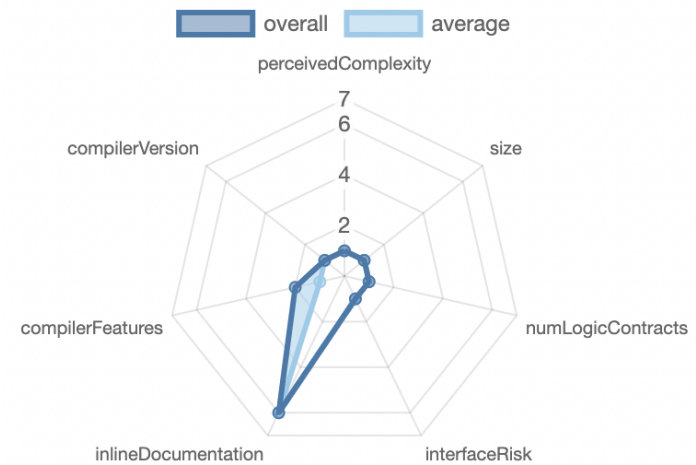
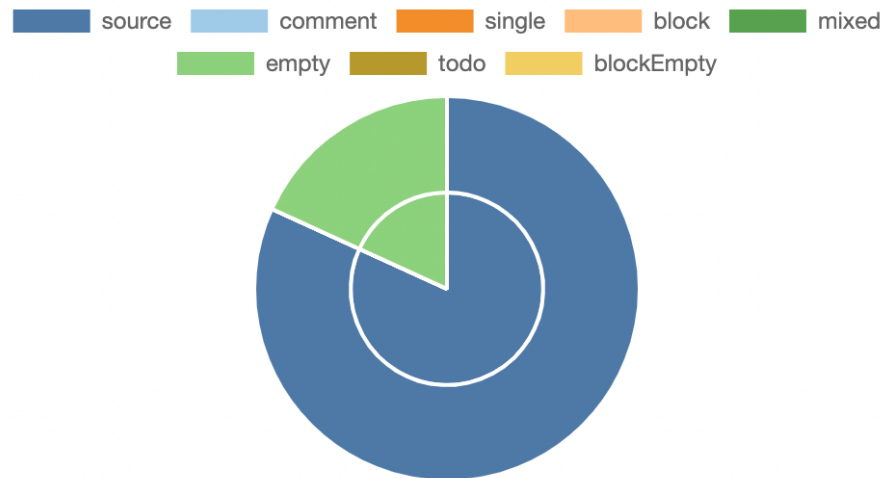
File	Fingerprint (MD5)
./DIAOracleV2.sol	9ea9ddbc37c32c5f125d8de61210a9d2










## 4.3 Metrics / CallGraph



## 4.4 Metrics / Source Lines & Risk





## 4.5 Metrics / Capabilities


Solidity Versions observed		 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
<div>0.7.4</div>					
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECRrecover	 New/Create/Create2

### Exposed Functions



This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

 <b>Public</b>	 <b>Payable</b>				
3	0				
<b>External</b>	<b>Internal</b>	<b>Private</b>	<b>Pure</b>	<b>View</b>	
1	3	0	0	1	

### StateVariables

<b>Total</b>	 <b>Public</b>
2	1

## 4.6 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complexity Score	Capabilities
	DIAOracleV2.sol	1		33	33	27		16	
	Totals	1		33	33	27	0	16	

Legend: [ ]

- **Lines:** total lines of the source unit
- **nLines:** normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC:** normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines:** lines containing single or block comments
- **Complexity Score:** a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

## 5. Scope of Work

The DIA Data Team provided us with the file that needs to be tested. The scope of the audit is the Oracle v2 contract.

The team put forward the following assumptions regarding the security, usage of the contracts:

- Only the Oracle Updater can update key value
- Oracle Updater can be changed by the old oracle updater only
- The smart contract is coded according to the newest standards and in a secure way

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

## 5.1 Manual and Automated Vulnerability Test

### CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

### HIGH ISSUES

During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

### MEDIUM ISSUES

During the audit, Chainsulting's experts found **no Medium issues** in the code of the smart contract

### LOW ISSUES

During the audit, Chainsulting's experts found **no Low issues** in the code of the smart contract

## INFORMATIONAL ISSUES

### 5.1.1 Missing natspec documentation

Severity: INFORMATIONAL

File(s) affected: ALL

Status: ACKNOWLEDGED

Attack / Description	Code Snippet	Result/Recommendation
Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).	NA	It is recommended to include natspec documentation and follow the doxygen style including @author, @title, @notice, @dev, @param, @return and make it easier to review and understand your smart contract.

## 5.2. SWC Attacks

ID	Title	Relationships	Test Result
<a href="#">SWC-131</a>	Presence of unused variables	<a href="#">CWE-1164: Irrelevant Code</a>	✓
<a href="#">SWC-130</a>	Right-To-Left-Override control character (U+202E)	<a href="#">CWE-451: User Interface (UI) Misrepresentation of Critical Information</a>	✓
<a href="#">SWC-129</a>	Typographical Error	<a href="#">CWE-480: Use of Incorrect Operator</a>	✓
<a href="#">SWC-128</a>	DoS With Block Gas Limit	<a href="#">CWE-400: Uncontrolled Resource Consumption</a>	✓
<a href="#">SWC-127</a>	Arbitrary Jump with Function Type Variable	<a href="#">CWE-695: Use of Low-Level Functionality</a>	✓
<a href="#">SWC-125</a>	Incorrect Inheritance Order	<a href="#">CWE-696: Incorrect Behavior Order</a>	✓
<a href="#">SWC-124</a>	Write to Arbitrary Storage Location	<a href="#">CWE-123: Write-what-where Condition</a>	✓
<a href="#">SWC-123</a>	Requirement Violation	<a href="#">CWE-573: Improper Following of Specification by Caller</a>	✓




ID	Title	Relationships	Test Result
<a href="#">SWC-122</a>	Lack of Proper Signature Verification	<a href="#">CWE-345: Insufficient Verification of Data Authenticity</a>	✓
<a href="#">SWC-121</a>	Missing Protection against Signature Replay Attacks	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-120</a>	Weak Sources of Randomness from Chain Attributes	<a href="#">CWE-330: Use of Insufficiently Random Values</a>	✓
<a href="#">SWC-119</a>	Shadowing State Variables	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-118</a>	Incorrect Constructor Name	<a href="#">CWE-665: Improper Initialization</a>	✓
<a href="#">SWC-117</a>	Signature Malleability	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-116</a>	Timestamp Dependence	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-115</a>	Authorization through tx.origin	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-114</a>	Transaction Order Dependence	<a href="#">CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-113</a>	DoS with Failed Call	<a href="#">CWE-703: Improper Check or Handling of Exceptional Conditions</a>	✓
<a href="#">SWC-112</a>	Delegatecall to Untrusted Callee	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-111</a>	Use of Deprecated Solidity Functions	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-110</a>	Assert Violation	<a href="#">CWE-670: Always-Incorrect Control Flow Implementation</a>	✓
<a href="#">SWC-109</a>	Uninitialized Storage Pointer	<a href="#">CWE-824: Access of Uninitialized Pointer</a>	✓
<a href="#">SWC-108</a>	State Variable Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-107</a>	Reentrancy	<a href="#">CWE-841: Improper Enforcement of Behavioral Workflow</a>	✓
<a href="#">SWC-106</a>	Unprotected SELFDESTRUCT Instruction	<a href="#">CWE-284: Improper Access Control</a>	✓
<a href="#">SWC-105</a>	Unprotected Ether Withdrawal	<a href="#">CWE-284: Improper Access Control</a>	✓
<a href="#">SWC-104</a>	Unchecked Call Return Value	<a href="#">CWE-252: Unchecked Return Value</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-103</a>	Floating Pragma	<a href="#">CWE-664: Improper Control of a Resource Through its Lifetime</a>	✓
<a href="#">SWC-102</a>	Outdated Compiler Version	<a href="#">CWE-937: Using Components with Known Vulnerabilities</a>	✓
<a href="#">SWC-101</a>	Integer Overflow and Underflow	<a href="#">CWE-682: Incorrect Calculation</a>	✓
<a href="#">SWC-100</a>	Function Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓


## 5.3 Verify claims

### 5.3.1 Only the oracle updater can update key value

**Status:** tested and verified 


```
18 function setValue(string memory key, uint128 value, uint128 timestamp) public {  
19     require(msg.sender == oracleUpdater);  
20     uint256 cValue = (((uint256)(value)) << 128) + timestamp;  
21     values[key] = cValue;  
22     emit OracleUpdate(key, value, timestamp);  
23 }
```

### 5.3.2 Oracle Updater can be changed by the old oracle updater only

**Status:** tested and verified 

```
32 function updateOracleUpdaterAddress(address newOracleUpdaterAddress) public {  
33     require(msg.sender == oracleUpdater);  
34     oracleUpdater = newOracleUpdaterAddress;  
35     emit UpdaterAddressChange(newOracleUpdaterAddress);  
36 }
```

### 5.3.3 The smart contract is coded according to the newest standards and in a secure way

**Status:** tested and verified 

## 6. Executive Summary

Our Chainsulting expert performed an audit of the smart contract codebase. The final debriefs took place on the January 30, 2022.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the claims inside the scope of work. During the audit, no critical issues were found after the manual and automated security testing.

## 7. Deployed Smart Contract

VERIFIED

<https://polygonscan.com/address/0xf44b3c104f39209cd8420a1d3ca4338818aa72ab#code>

