DIA Yield SMART CONTRACT AUDIT

FOR DIA Data Limited & D.I.A e.V.

12.08.2020

Made in Germany by Chainsulting.de

Smart Contract Audit - DIA Yield

1. Disclaimer

The audit makes no statements or warrantees 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.

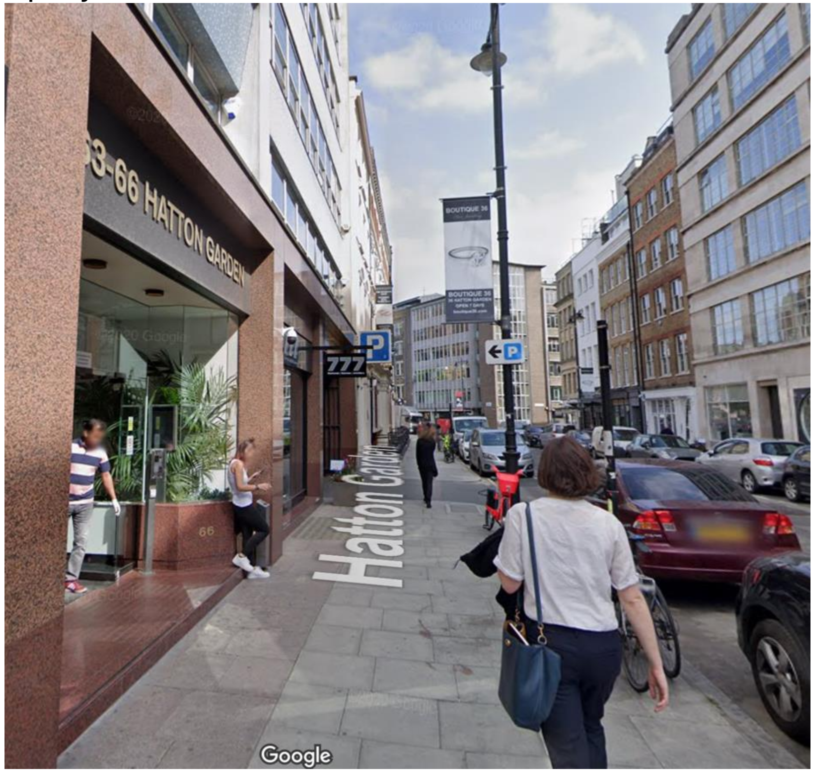
The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of D.I.A e.V. . If you are not the intended receptor of this document, remember that any disclosure, copying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

Major Versions / Date	Description	Author
0.1 (07.08.2020)	Layout	Y. Heinze
0.5 (08.08.2020)	Automated Security Testing Manual Security Testing	Y. Heinze
1.0 (09.08.2020)	Summary and Recommendation	Y. Heinze
1.1 (10.08.2020)	Finalization	Y. Heinze
1.2 (11.08.2020)	Added deployed contract address etc.	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)	Version
yield.sol	CEFCA34BB95ABC97F061ED1095AB278344734C9BDE45AD170C429F1BAC7E24BF	1.0
yield.sol	42C61F27EF4A2A0535B10FBDB3B04B97315C068D60A2F8C28A98CDE8AE0A6874	1.5

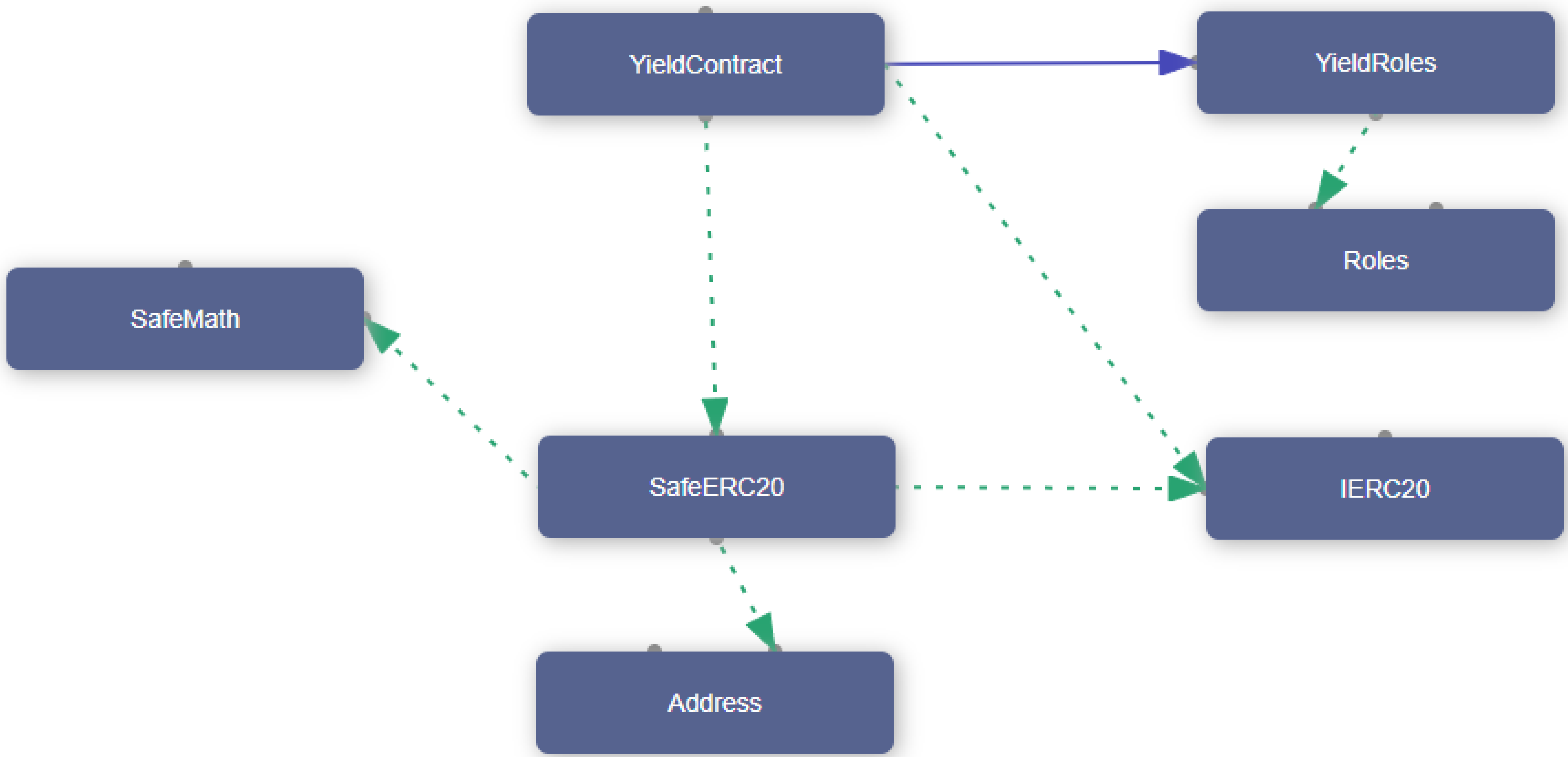
Source:
https://raw.githubusercontent.com/chainsulting/Smart-Contract-Security-Audits/master/DIA%20Token/yield.sol
https://raw.githubusercontent.com/chainsulting/Smart-Contract-Security-Audits/master/DIA%20Token/yield_public.sol

4.3 Context

Token holder will be able to lock DIA Tokens into three timeframes, which are 3, 6 or 12 months long. Over the locking period they receive dividends in form of DIA Tokens. The withdrawal timer is starting after deposit of funds and is stopping after chosen locking period.

5. Summary of Smart Contract

5.1 Visualized Dependencies



5.2 Functions

contract	func	visibility	modifiers	stateMutability
Roles	add	internal		
Roles	remove	internal		
Roles	has	internal		view
YieldRoles	"constructor"	internal		
YieldRoles	isOwner	public		view
YieldRoles	addOwner	public	onlyOwner	
YieldRoles	renounceOwner	public		
YieldRoles	_addOwner	internal		
YieldRoles	_removeOwner	internal		
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
Address	isContract	internal		view
Address	sendValue	internal		
Address	functionCall	internal		
Address	functionCall	internal		
Address	functionCallWithValue	internal		
Address	functionCallWithValue	internal		
Address	_functionCallWithValue	private		
SafeERC20	safeTransfer	internal		
SafeERC20	safeTransferFrom	internal		
SafeERC20	safeApprove	internal		
SafeERC20	safeIncreaseAllowance	internal		
SafeERC20	safeDecreaseAllowance	internal		
SafeERC20	_callOptionalReturn	private		
YieldContract	"constructor"	public		
YieldContract	getLockBoxBeneficiary	public		view
YieldContract	deposit3m	external		
YieldContract	deposit6m	external		
YieldContract	deposit12m	external		
YieldContract	deposit	internal		
YieldContract	updateBeneficiary	public		
YieldContract	withdraw	public		
YieldContract	triggerWithdrawAll	public		
YieldContract	updateEndDepositTime	public	onlyOwner	

YieldContract	updateYieldWallet	public	onlyOwner	
YieldContract	updateYields	public	onlyOwner	
YieldContract	updateMaxTokens	public	onlyOwner	

5.3 Modifiers

contract	modifier
YieldRoles	onlyOwner

5.4 States

contract	state	type	visibility	isConst
YieldRoles	_owners	Roles.Role	private	false
YieldContract	token	IERC20	default	false
YieldContract	beginDepositTime	uint256	public	false
YieldContract	endDepositTime	uint256	public	false
YieldContract	maxTokens	uint256	internal	false
YieldContract	yieldWallet	address	internal	false
YieldContract	threeMonthPercentage	uint256	public	false
YieldContract	sixMonthPercentage	uint256	public	false
YieldContract	twelveMonthPercentage	uint256	public	false
YieldContract	lockBoxStructs	array	public	false

6. Test Suite Results

The DIA Yield is a separate smart contract, which utilize locking functions and pays out dividends. 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.

Testnet
https://ropsten.etherscan.io/address/0x55291d45ce619fa88df6ca584da20acf068dc7f9#code

6.1 MYTHX

Mythril Classic & Mythx are 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: 2
Medium: 1
High: 0
Critical: 0

6.1.1 A floating pragma is set

Severity: LOW (FIXED)
Code: SWC-103
File(s) affected: yield.sol

Attack / Description	Code Snippet	Result/Recommendation
A floating pragma is set.	Line: 1 `pragma solidity ^0.5.12;`	The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code. Pragma solidity 0.5.12

6.1.2 State variable visibility is not set.

Severity: LOW (FIXED)
Code: SWC-108
File(s) affected: yield.sol

Attack / Description	Code Snippet	Result/Recommendation

State variable visibility is not set.	Line: 531
	`uint256 beginDepositTime;`
	Line: 532
	`uint256 endDepositTime;`
	Line: 535
	`address yieldWallet;`
	Line: 538
	`address kickerWallet;`
	Line: 539
	`uint256 kickerDeadline;`
	Line: 540
	`uint256 kickerPromille;`
		It is best practice to set the visibility of state variables explicitly. The default visibility for non-set state variables is internal. Other possible visibility settings are public and private.
		
		JavaScript front end for a DApp, can read any data in the blockchain, but marking a state variable as public makes it considerably easier to access.
		
		https://programtheblockchain.com/posts/2018/01/02/making-smart-contracts-with-public-variables/

6.1.3 Loop over unbounded data structure

Severity: MEDIUM (FIXED)
Code: SWC-128
File(s) affected: yield.sol

Attack / Description	Code Snippet	Result/Recommendation
Loop over unbounded data structure	Line: 638 `for (uint256 i = 0; i < lockBoxStructs.length; ++i) {`	Gas consumption in function "triggerWithdrawAll" in contract "DIACompanyLockup" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

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

6.2 Manually Security Testing

Detected Vulnerabilities
Informational: 1
Low: 1
Medium: 0
High: 0
Critical: 0

6.2.1 Private modifier

Severity: INFORMATIONAL (FIXED)
Code: None
File(s) affected: yield.sol

Attack / Description	Code Snippet	Result/Recommendation
Contrary to a popular misconception, the private modifier does not make a variable invisible. Miners have access to all contracts' code and data. Developers must account for the lack of privacy in Ethereum.	Line: 51 `Roles.Role private _owners;`	Keep in mind that the private modifier does not make a variable invisible. https://solidity.readthedocs.io/en/develop/contracts.html#visibility-and-getters

6.2.2 Prefer external to public visibility level (Gas Optimization)

Severity: LOW (FIXED)
Code: None
File(s) affected: yield.sol

Attack / Description	Code Snippet	Result/Recommendation

A function with public visibility modifier that is not called internally. Changing visibility level to external increases code readability. Moreover, in many cases functions with external visibility modifier spend less gas comparing to functions with public visibility modifier.	Line: 570 - 585
	`function deposit3m(address beneficiary, uint256 amount) public {`
	`        deposit(beneficiary, amount, 12 weeks);`
	`    }`
		As for best practices, you should use external if you expect that the function will only ever be called externally, and use public if you need to call the function internally.
		
		https://medium.com/@gus_tavo_guim/public-vs-external-functions-in-solidity-b46bcf0ba3ac

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	✓
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	✓

ID	Title	Relationships	Test Result
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	✅
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	✅

ID	Title	Relationships	Test Result
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	✅
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	✅

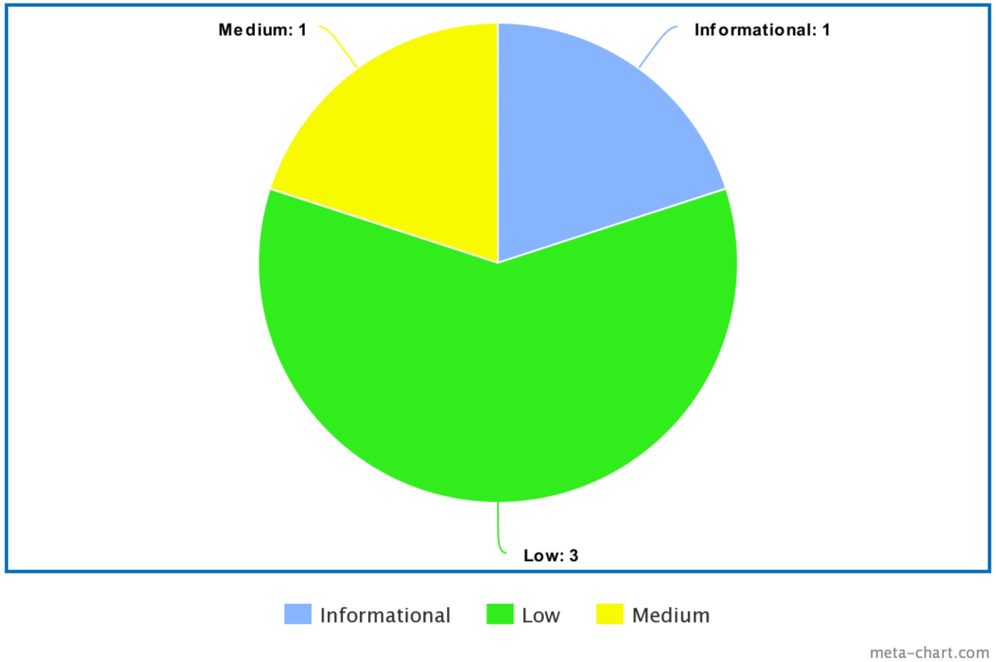
ID	Title	Relationships	Test Result
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✅
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✅
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅

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

8. 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.

The used libraries are based on OpenZepplin codebase, such as Roles, IERC20, SafeMath, SafeERC20, TokenTimelock. We suggest to use solidity version 0.6.0 and utilizing the newest OpenZepplin libraries. Also consider the mentioned state variables marking as public makes it considerably easier to access.



9. Deployed Smart Contract & Summary

https://etherscan.io/address/0x59490052ea18d4fd974657d0aa7c8424d456d353#code

The DIA Yield smart contract have been analysed under different aspects. We have been able to identify several security issues. DIA Teams response has been fast and professional, they were able to fix most of the issues quickly through code changes.