WaveLength

Protocol

SMART CONTRACT AUDIT

26.10.2022

Made in Germany by Chainsulting.de

Table of contents

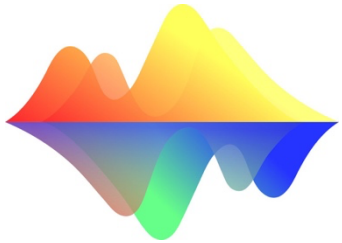
1. Disclaimer

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 NEOFINTECH LDA. 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 (05.09.2022)	Layout
0.4 (08.09.2022)	Automated Security Testing Manual Security Testing
0.5 (14.09.2022)	Verify Claims and Test Deployment
0.6 (16.09.2022)	Testing SWC Checks
0.9 (16.09.2022)	Summary and Recommendation
1.0 (17.09.2022)	Final document
1.1 (22.10.2022)	Re-check
1.2 (23.10.2022)	Deployment to Mainnet

2. About the Project and Company



Company address:

Wavelength Labs Inc.
Intershore Chambers
P.O. Box 4342
Road Town, Tortola VG1110
British Virgin Islands

Website: https://www.wavelength.exchange

Twitter: https://twitter.com/wavelength_dao

Telegram: https://t.me/WavelengthDAO

Reddit: https://www.reddit.com/r/Wavelength_DAO

2.1 Project Overview

WaveLength is built on Balancer V2, and it is expected to become a one-stop shop and premier liquidity hub for Decentralized Financial Services on Velas.

They are developing further on the novel creations brought to DeFi by the Balancer team to leverage their technology and generate automated index funds that yield returns through trading fees from traders, who will re-balance the user's portfolios when following arbitrage opportunities.
They also provide features to maximize capital efficiency for traders, allowing a low slippage trading experience and effective stableswaps, essentially revolutionizing the DeFi trading experience on Velas, making room for higher on-chain volume, more cost-effective trading, and a possibility for custom AMMs.

Trading on WaveLength is the next level of liquidity usage. A traditional AMM will simply execute a trade through the deepest liquidity pool for that specific exchange. The WaveLength Smart Order Router will calculate the most efficient path and break up the trades through all of the WaveLength Pools, ensuring the most optimal trading experience by utilizing all of the AMM's available liquidity by improving capital optimization.

Wavelength Pools are smart contracts that define how traders can swap between tokens on Wavelength Protocol. Their limitless flexibility makes Wavelength Pools unique from those of other protocols.

- **Index Fund Pools** - Weighted Pools are highly versatile and configurable pools. They are ideal for general cases and enable users to build pools with different token counts and weightings, such as pools with 80/20 or 50/50 weightings.
- **Stable Pools** - Stable Pools are optimal for assets expected to consistently trade at near parity, such as different varieties of stablecoins or synthetics.
- **Custom Pools -** Any user can create their own custom pools with any tokens and weights they so choose, as well as choose the pool fees.

Wavelength's native WAVE token is known as a governance token. WAVE holders will have an opportunity to vote on proposals relevant to the Protocol, such as new features and new directions that Wavelength Protocol could follow or not. These proposals will range from Protocol fees to how WAVE tokens should be distributed, for example, the allocation of WAVE tokens towards the Balancer Liquidity Mining program.

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.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 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)
WeightedPool2TokensFactory.sol	9f310f93637474df8d0ff3fc4ce053a3
WAVEToken.sol	6b1cd99a08ee8145b7f49fbadc217e30
Authorizer.sol	b4bb85f98bf907803ae86461a353ad3c
WAVEMasterChef.sol	550acf21de4ae412479473e27c827784
AaveLinearPoolFactory.sol	2b2338929d7dccb66ea4377c1d4831b9
WeightedPoolFactory.sol	8974d0a4cbbe710eb0db5fa319a1d1b5
Timelock.sol	9b3bb5526c247585417d9ddce0d0201b
Vault.sol	6361cb4af92b9747c179f03e7dcdcabc
MasterChefOperator.sol	8bb6d2a5453e407b34be74ea548bfa8b
NoProtocolFeeLiquidityBootstrappingPoolFactory.sol	35a983c670cc20b78785652804b59231
StablePoolFactory.sol	e3899654d096ec4ab05a7a50d3c75559

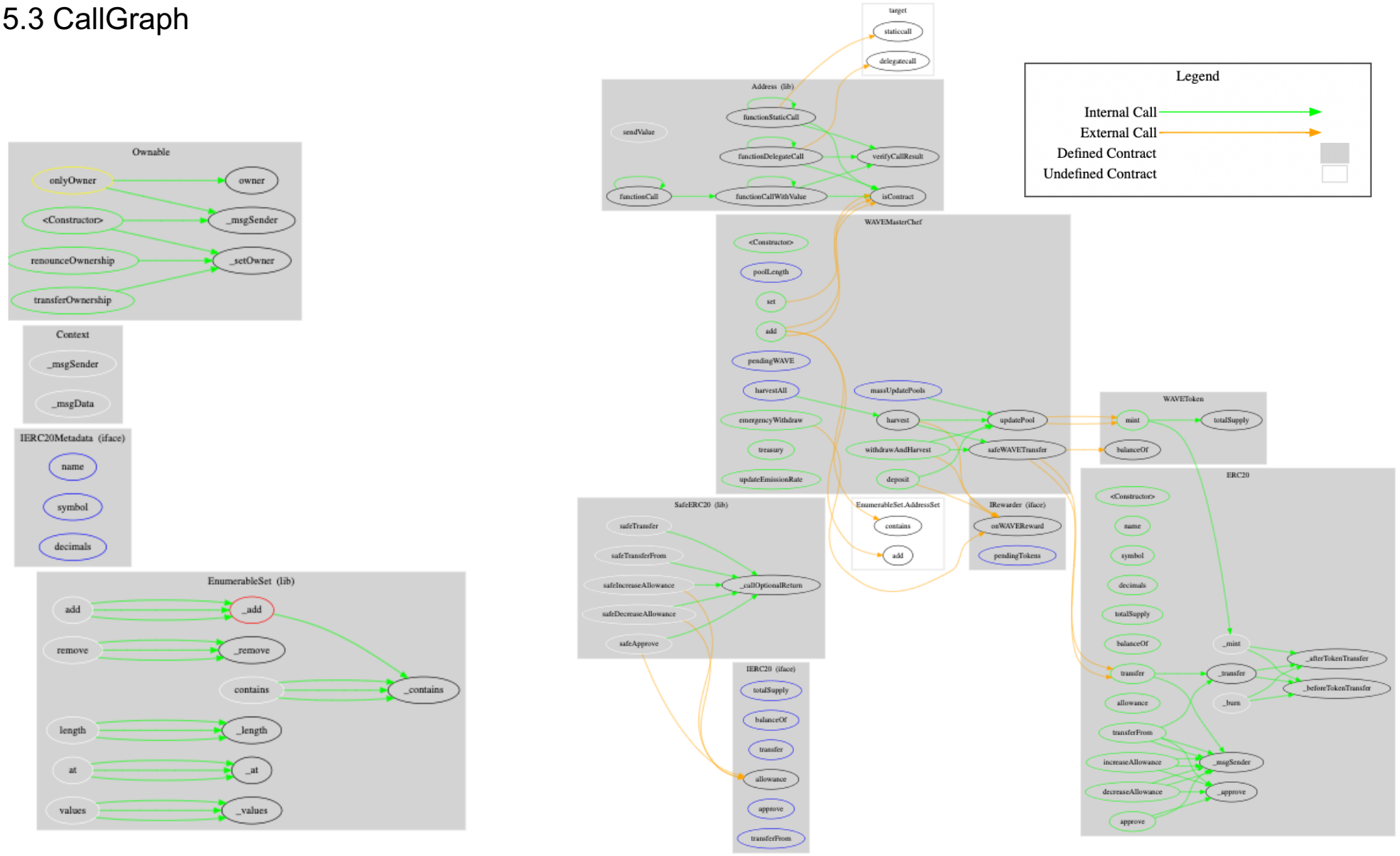
5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
AaveLinearPoolFactory.sol	https://github.com/balancer-labs/metastable-rate-providers
AaveLinearPoolFactory.sol, Authorizer.sol Vault.sol, Timelock.sol, StablePoolFactory.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol	https://github.com/balancer-labs/balancer-v2-monorepo/tree/master/pkg
MasterChefOperator.sol	https://github.com/beethovenxfi/beethovenx-token/blob/main/contracts/governance/MasterChefOperator.sol
Vault.sol	https://github.com/Mozaic-fi/balancer-pool-utils
WAVEMasterChef.sol	https://github.com/beethovenxfi/beethovenx-token/blob/main/contracts/token/BeethovenxMasterChef.sol
WAVEToken.sol	https://github.com/beethovenxfi/beethovenx-token/blob/main/contracts/token/BeethovenxToken.sol
AaveLinearPoolFactory.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, StablePoolFactory.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol (library SafeMath)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0-solc-0.7/contracts/math/SafeMath.sol
AaveLinearPoolFactory.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, StablePoolFactory.sol, Vault.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol (library Math)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0-solc-0.7/contracts/math/Math.sol
AaveLinearPoolFactory.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol,	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0-solc-0.7/contracts/token/ERC20/ERC20.sol

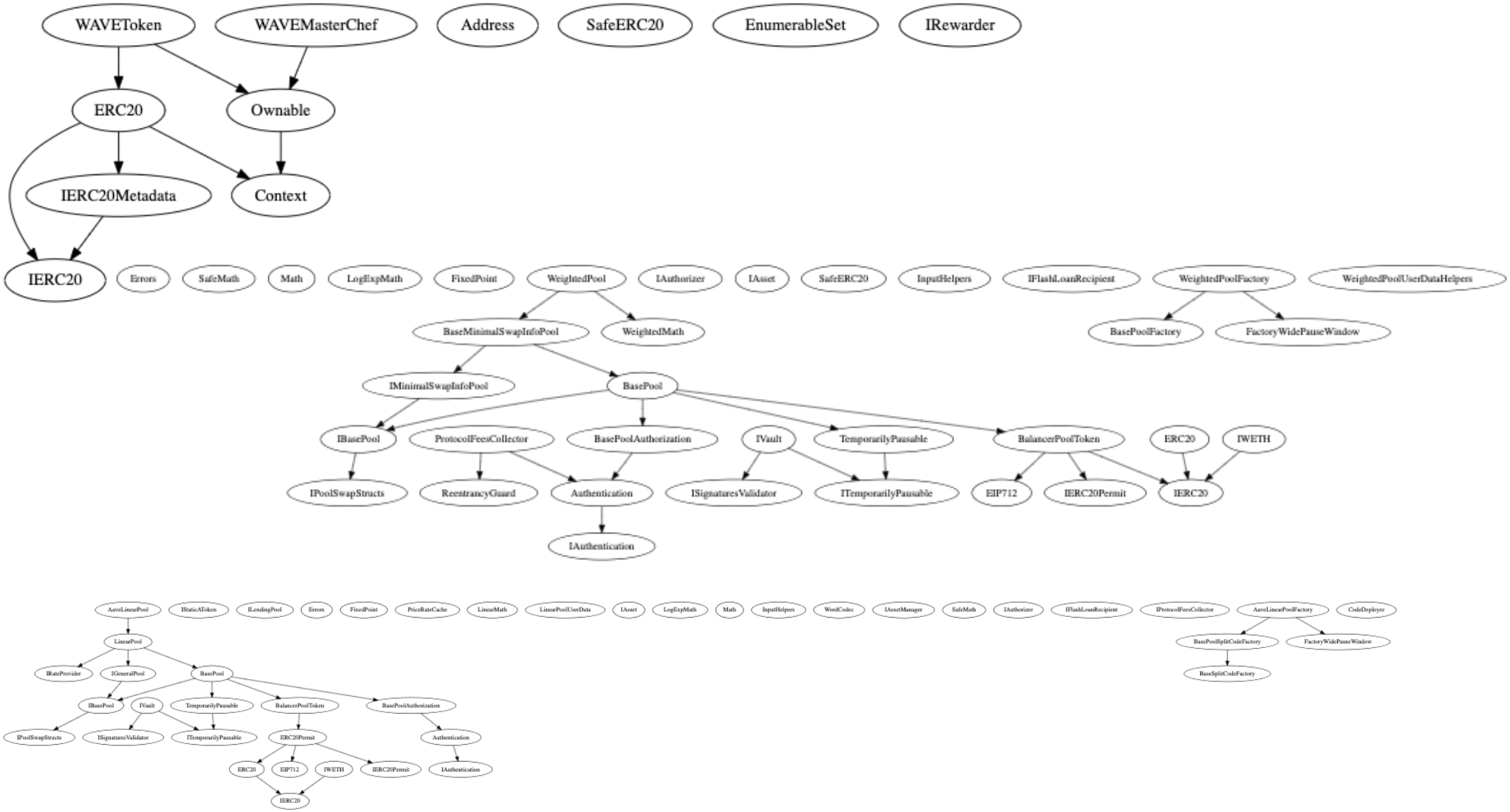
Dependency / Import Path	Source
StablePoolFactory.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol, WAVEMasterChef.sol, WAVEToken.sol (library ERC20)	
AaveLinearPoolFactory.sol, Authorizer.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, StablePoolFactory.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol, MasterChefOperator.sol, WAVEMasterChef.sol (library, WAVEToken.sol IERC20)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0-solc-0.7/contracts/token/ERC20/IERC20.sol
Authorizer.sol, Vault.sol, MasterChefOperator.sol, WAVEMasterChef.sol (library EnumerableSet)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0-solc-0.7/contracts/utils/EnumerableSet.sol
NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, Vault.sol, WeightedPool2TokensFactory.sol (library ReentrancyGuard)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0-solc-0.7/contracts/utils/ReentrancyGuard.sol
Vault.sol, WAVEMasterChef.sol (library Address)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0-solc-0.7/contracts/utils/Address.sol
Vault.sol (library SafeCast)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0-solc-0.7/contracts/utils/SafeCast.sol
Vault.sol (library EnumerableMap)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0-solc-0.7/contracts/utils/EnumerableMap.sol
Vault.sol, WeightedPoolFactory.sol, WAVEMasterChef.sol (library SafeERC20)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0-solc-0.7/contracts/token/ERC20/SafeERC20.sol
AaveLinearPoolFactory.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, StablePoolFactory.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol (library IERC20Permit)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/token/ERC20/extensions/draft-IERC20Permit.sol

Dependency / Import Path	Source
AaveLinearPoolFactory.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, StablePoolFactory.sol, Vault.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol (library EIP712)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/utils/cryptography/draft-EIP712.sol
MasterChefOperator.sol (library IERC165)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.7.0/contracts/interfaces/IERC165.sol
MasterChefOperator.sol, WAVEMasterChef.sol, WAVEToken.sol (library Context)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.7.0/contracts/utils/Context.sol
MasterChefOperator.sol, WAVEMasterChef.sol, WAVEToken.sol (library IER20Metadata)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.7.0/contracts/interfaces/IERC20Metadata.sol
WAVEMasterChef.sol, WAVEToken.sol (library Ownable)	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.7.0/contracts/access/Ownable.sol

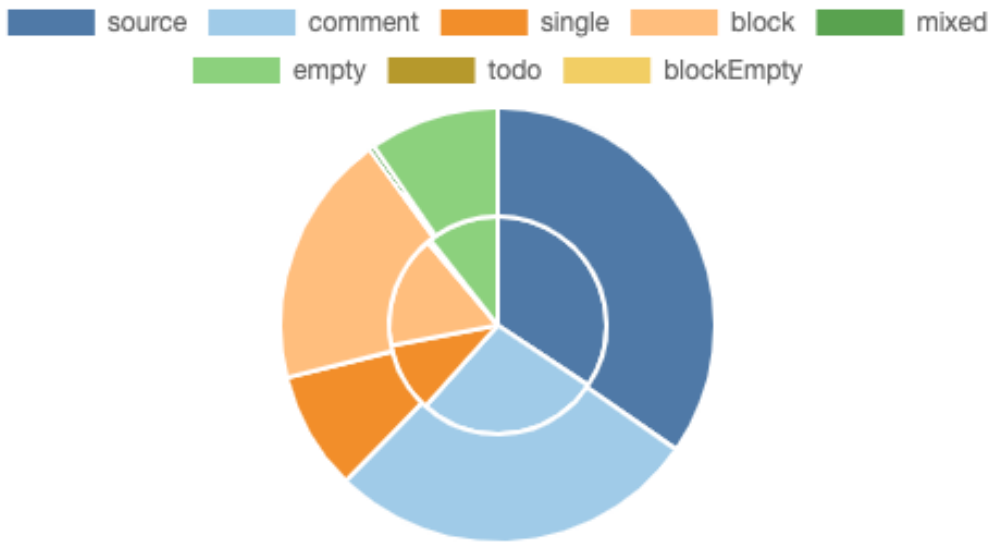
5.3 CallGraph

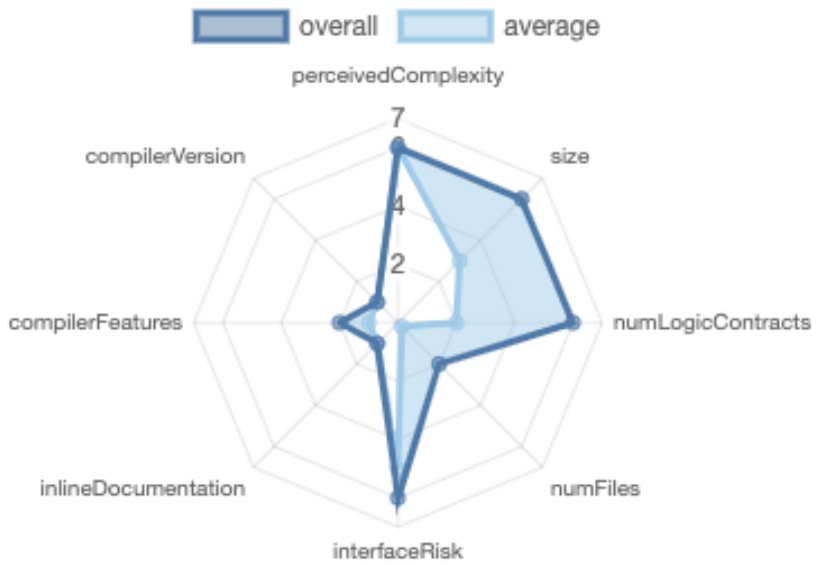


5.4 Inheritance Graph



5.5 Source Lines & Risk





5.6 Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts

`^0.7.0`
`0.8.7`
`^0.8.0`
	`ABIEncoderV2`	`yes`	`yes`
			(77 asm blocks)	

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
`yes`		`yes`	`yes`	`yes`	`yes` → `NewContract:StablePool` → `AssemblyCall:Name:create` → `NewContract:ProtocolFeesCollector` → `NewContract:WeightedPool` → `NewContract:WeightedPool2Tokens`

TryCatch	Unchecked
	`yes`

Contracts	Libraries	Interfaces	Abstract
60	72	106	74

Exposed Functions

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

🌐Public	💰Payable
830	40

External	Internal	Private	Pure	View
588	1634	226	557	787

StateVariables

Total	🌐Public
1570	41

5.7 Source Unites in Scope

Source: https://github.com/wavelength-velas/WaveLength-Contracts/tree/main/contracts/Audit_Contracts
Last commit: 7fb66f98ace78a554b25e68113134196927f1c52

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/StablePoolFactory.sol	24	16	5563	4313	2017	2186	1665	
	contracts/NoProtocolFeeLiquidityBootstrappingPoolFactory.sol	47	18	8830	7139	3610	3067	2999	
	contracts/MasterChefOperator.sol	7	5	1389	1198	679	516	389	
	contracts/Vault.sol	32	13	6077	5014	2271	2496	1838	
	contracts/Timelock.sol	1		236	218	163	21	73	
	contracts/WeightedPoolFactory.sol	23	13	5012	3995	1898	1986	1522	
	contracts/AaveLinearPoolFactory.sol	25	18	5806	4593	2137	2318	1727	
	contracts/WAVEMasterChef.sol	8	3	1729	1513	702	745	401	

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍🎨	contracts/Authorizer.sol	5	3	759	680	300	359	280	🖥️🔆
📝🔍🎨	contracts/WAVEToken.sol	4	2	616	481	188	306	138	🔆Σ
📝📚🔍🎨	contracts/WeightedPool2TokensFactory.sol	30	15	7187	5868	2798	2694	2246	🖥️🧪💰🧱🪚🌀🔆
📝📚🔍🎨	**Totals**	**206**	**106**	**43204**	**35012**	**16763**	**16694**	**13278**	🖥️🧪💰🔌👥🧱🪚🌀🔆Σ

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, ...)

6. Scope of Work

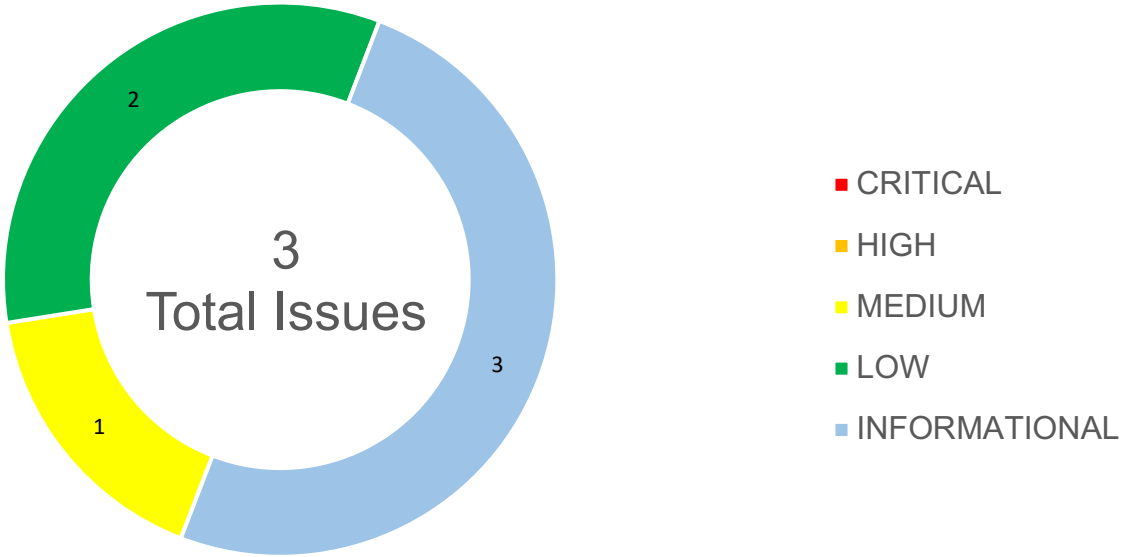
The WaveLength Team provided us with the files that needs to be tested. The scope of the audit are the protocol contracts.

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

- The Balancer contracts are correctly implemented
- The Vault is correctly implemented and working as expected
- Timelock is correctly implemented and working as expected
- MasterChefs are correctly implemented and working as expected
- 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.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Signature Malleability	MEDIUM	ACKNOWLEDGED
6.2.2	Old And Non-Constant Solc Pragma ^0.7.0	LOW	ACKNOWLEDGED
6.2.3	Custom Errors To Save Gas	LOW	ACKNOWLEDGED
6.2.4	Floating Pragma Version Identified	INFORMATIONAL	ACKNOWLEDGED
6.2.5	Flatted Files	INFORMATIONAL	ACKNOWLEDGED
6.2.6	Use Of SafeMath Library	INFORMATIONAL	ACKNOWLEDGED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES
During the audit, Chainsulting‘s experts found **0 Critical issues** in the code of the smart contract.

HIGH ISSUES
During the audit, Chainsulting’s experts found **0 High issues** in the code of the smart contract.

MEDIUM ISSUES
During the audit, Chainsulting’s experts found **1 Medium issue** in the code of the smart contract.

6.2.1 Signature Malleability
Severity: MEDIUM
Status: ACKNOWLEDGED
Code: NA
File(s) affected: AaveLinearPoolFactory.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, StablePoolFactory.sol, Vault.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol

Attack / Description	The function ecrecover allows you to convert a valid signature into a different valid signature without requiring knowledge of the private key. It is usually not a problem unless you use signatures to identify items or require them to be uniquely recognizable. Therefore, depending on the function of the code, this may lead to discrepancies and faulty logic.
Code	e.g. Line 880 (AaveLinearPoolFactory.sol) `address signer = ecrecover(hash, v, r, s);`

Result/Recommendation	It is recommended to use OpenZeppelin's ECDSA library that has a wrapper around ecrecover that mitigates this issue. The data signer can be recovered using ECDSA.recover, and its address can be compared to verify the signature.

LOW ISSUES

During the audit, Chainsulting's experts found **2 Low issues** in the code of the smart contract.

6.2.2 Old And Non-Constant Solc Pragma ^0.7.0

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: AaveLinearPoolFactory.sol, Authorizer.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, StablePoolFactory.sol, Vault.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol

Attack / Description	The solc pragma ^0.7.0 is old and non-constant. Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.
Code	e.g. Line 1 `pragma solidity ^0.7.0;`
Result/Recommendation	We recommend monitoring the known solc bugs to ensure nothing in the 0.7.x range applies to the codebase and consider using the solidity version 0.8.7, which patches most solidity vulnerabilities. The recommendation is not associated with a specific vulnerability; however, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

6.2.3 Custom Errors To Save Gas

Severity: LOW

Status: ACKNOWLEDGED
Code: NA
File(s) affected: AaveLinearPoolFactory.sol, Authorizer.sol, MasterChefOperator.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, StablePoolFactory.sol, Timelock.sol, Vault.sol, WAVEMasterChef, WAVEToken.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol

Attack / Description	The contract was found to be using revert() statements. Since Solidity v0.8.4, custom errors have been introduced which are a better alternative to the revert. This allows the developers to pass custom errors with dynamic data while reverting the transaction and also making the whole implementation a bit cheaper than using reverts.
Code	e.g. Line 304 (WAVEMasterChef.sol) `revert(errorMessage);`
Result/Recommendation	It is recommended to replace all the instances of revert() statements with error() to save gas, if you consider to upgrade the compiler version

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **3 Informational issues** in the code of the smart contract.

6.2.4 Floating Pragma Version Identified
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: SWC-103
File(s) affected: ALL

Attack / Description	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.

Code	e.g. Line 2
	`pragma solidity ^0.7.0;`
	`pragma solidity ^0.8.0;`

Result/Recommendation	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. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version. i.e. Pragma solidity 0.7.0 or 0.8.0

6.2.5 Flatted Files

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: ALL

Attack / Description	The solidity project files have been flatted which makes the review, modification, updates in the future quite hard.
Code	NA
Result/Recommendation	It is recommended to have a clean repository project structure, with folders for interfaces, libraries and correctly imported 3th party code, such as OpenZeppelin Framework or Balancer core contracts.

6.2.6 Use Of SafeMath Library

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA

File(s) affected: AaveLinearPoolFactory.sol, NoProtocolFeeLiquidityBootstrappingPoolFactory.sol, StablePoolFactory.sol, Vault.sol, WAVEMasterChef.sol, WeightedPool2TokensFactory.sol, WeightedPoolFactory.sol

Attack / Description	SafeMath library is found to be used in the contract. This increases gas consumption than traditional methods and validations if done manually.

Code	`pragma solidity ^0.7.0;`
	
	`/**`
	` * @dev Wrappers over Solidity's arithmetic operations with added overflow`
	` * checks.`
	` *`
	` * Arithmetic operations in Solidity wrap on overflow. This can easily result`
	` * in bugs, because programmers usually assume that an overflow raises an`
	` * error, which is the standard behavior in high level programming languages.`
	` * `SafeMath` restores this intuition by reverting the transaction when an`
	` * operation overflows.`
	` *`
	` * Using this library instead of the unchecked operations eliminates an entire`
	` * class of bugs, so it's recommended to use it always.`
	` */`
	`library SafeMath {`

Result/Recommendation	It is recommended to upgrade compiler version to solidity 0.8, as it includes checked arithmetic operations by default, and this renders SafeMath unnecessary.

6.3 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')	✅

ID	Title	Relationships	Test Result
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	✅
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✅

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	X
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	✅

6.4 Verify Claims

6.4.1 The Balancer contracts are correctly implemented
Status: tested and verified ✅

6.4.2 The Vault is correctly implemented and working as expected
Status: tested and verified ✅

6.4.3 Timelock is correctly implemented and working as expected
Status: tested and verified ✅

6.4.4 MasterChefs are correctly implemented and working as expected
Status: tested and verified ✅

6.4.5 The smart contract is coded according to the newest standards and in a secure way.
Status: tested and verified ✅

6.5 Audits By Forked Codebases

Source: https://github.com/balancer-labs/balancer-v2-monorepo

Provider	Date	Link	Check
ABDK	15.04.2022	https://github.com/balancer-labs/balancer-v2-monorepo/blob/master/audits/abdk/2022-04-15.pdf	✅
Certora	19.04.2021	https://github.com/balancer-labs/balancer-v2-monorepo/blob/master/audits/certora/2021-04-19.pdf	✅
OpenZeppelin	15.03.2021	https://github.com/balancer-labs/balancer-v2-monorepo/blob/master/audits/openzeppelin/2021-03-15.pdf	✅
OpenZeppelin	09.10.2021	https://github.com/balancer-labs/balancer-v2-monorepo/blob/master/audits/openzeppelin/2021-10-09.pdf	✅
Trail of Bits	05.04.2021	https://github.com/balancer-labs/balancer-v2-monorepo/blob/master/audits/trail-of-bits/2021-04-05.pdf	✅
Trail of Bits	22.12.2021	https://github.com/balancer-labs/balancer-v2-monorepo/blob/master/audits/trail-of-bits/2021-12-22.pdf	✅

Provider	Date	Link	Check
Trail of Bits	22.06.2022	https://github.com/balancer-labs/balancer-v2-monorepo/blob/master/audits/trail-of-bits/2022-06-22.pdf	✅

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase.

The main goal of the audit was to verify the claims regarding the security and functions of the smart contract. During the audit, no critical, no high, 1 medium, 2 low and 3 informational issues have been found, after the manual and automated security testing.

We advise the WaveLength team to implement the recommendations contained in all 6 of our findings, to further enhance the overall security and readability.

8. Mainnet Contract

VERIFIED

WaveToken: https://explorer.velas.com/address/V2bVUyzHDtTnrmwbpwR2xP8RNrRKUmQTay
Authorizer: https://explorer.velas.com/address/VBcQL75BXJtjcjvKJmNTHwHzKCQbxAhdQD
Vault: https://explorer.velas.com/address/VG1YwvB7dz7xzrbPhrmnB59VenZfMugNp8
WeightedPoolFactory: https://explorer.velas.com/address/VGksYiJeeReeqk2BhxWhhQoAF2NJSF2286
WeightedPool2TokensFactory: https://explorer.velas.com/address/VGswC9M2r4uPBvpYXppjfaPELy5RTh3go5

StablePoolFactory: https://explorer.velas.com/address/V9W9fCHbPCtCKKxaRaMTAXAPTypwfuZ83v
StablePhantomPoolFactory: https://explorer.velas.com/address/V4Yngvgsv8KVeqk2b9xr8kQexysrvr4Hkd
ProtocolFeesCollector: https://explorer.velas.com/address/V15sb2QvCMtmw2QCKzAd31ZmivqrMaggpU
NoProtocolFeeLiquidityBootstrappingPoolFactory: https://explorer.velas.com/address/VekEvHhDrkoQR7Z2dDjKVh9w7CsWv9E59
Multicall: https://explorer.velas.com/address/V9GU5YtzmQ1ZenL87odLuAWESooqpQ8vpw
BatchRelayerLibrary: https://explorer.velas.com/address/VKp9XnWHsfq5eemBtfuZ9FRyw3FLfBvRG6
BalancerHelpers: https://explorer.velas.com/address/VKECn8xgSgfo5LVEeycw8kDsch9rvsnQv5
WaveMasterChef: https://explorer.velas.com/address/VPCDu78Bb9TmQdG4yKHPVnnpGnKJfT1ZHx
WaveBar: https://explorer.velas.com/address/VM8993WKp3UKYbRSdErT9o3JvcodfrDmCh
TREASURY: https://explorer.velas.com/address/VK8us26mDErfnb6wBeaB189R35cvGLwXTz
DEV: https://explorer.velas.com/address/V5NdXfhjvV27SfZbLomKcxxpwY3hpxVJPh

9. About the Auditor

Chainsulting is a professional software development firm, founded in 2017 and based in Germany. They show ways, opportunities, risks and offer comprehensive web3 solutions. Their services include web3 development, security and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Solana, Tezos, Ethereum, Binance Smart Chain, and Polygon to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of 1Inch, POA Network, Unicrypt, LUKSO among numerous other top DeFi projects.

Chainsulting currently secures $100 billion in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the web3 sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

Check our website for further information: https://chainsulting.de