HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: UWULEND
Date: 21 Sep, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for UWULEND
Approved By	Oleksii Zaiats | SC Audits Head at Hacken OÜ
Tags	Dex; Vesting; Yield Farming;
Platform	EVM
Language	Solidity
Methodology	Link
Website	http://wagmi.com/
Changelog	18.08.2023 - Initial Review 13.09.2023 - Second Review 21.09.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by UWULEND (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

Wagmi is a staking protocol with the following contracts:

- Dispatcher.sol - contract acts as a central coordinator, interfacing between users, multipools, and strategies. It allows users to deposit and withdraw from multipools, and it ensures rewards are correctly distributed.
- Factory.sol - is responsible for creating new multipool instances, ensuring each new pool adheres to specific standards and configurations.
- Multipool.sol - handles the core functionality of liquidity pools, such as allowing users to add or remove liquidity, processing swaps between tokens, and internally balancing the pool's assets. The rebalancing feature ensures that the ratio of assets remains optimal over time.
- MultipoolToken.sol - When users deposit assets into a multipool, they receive multipool tokens in return, representing their share of the pool. This contract manages the minting, burning, and general ERC20 operations for these tokens.
- MultiStrategy.sol - defines and executes various investment strategies to maximize returns.
- PlatformFeesVault.sol - is a treasury for the platform. All platform fees, whether from swaps, withdrawals, or other operations, are sent to this contract.
- WagmiVesting.sol - is a mechanism to release tokens to users over a specified time period, often used to incentivize long-term participation and loyalty.
- GMI.sol - a primary token or management mechanism for the GMI platform, overseeing pools, users, rewards, and related interactions.
- ErrLib.sol - a library for error handling.
- IMultipool.sol - an interface for Multipool contract.
- IDispatcher.sol - an interface for Dispatcher contract.
- IFactory.sol - an interface for Factory contract
- IMultiStrategy.sol - an interface for MultiStrategy contract.
- IPlatformFeesVault.sol - an interface for PlatformFeesVault contract.

Privileged roles

- WagmiVesting.sol:
  - Owner:
    - getBackWagmi: Retrieves a specified amount of WAGMI tokens from the contract and transfers them to the specified address.
- Gmi.sol:
  - Owner:
    - setEarnBeforeMint: Allows the owner to configure a setting related to earning before minting.
    - setEarnOperator: Allows the owner to set or modify the earn operator.
    - getMultiPoolIndex: Allow the owner to retrieve the index of a multipool.
    - _setOracle: Allows the owner to set or modify the oracle.
    - setPremiumInfo: Allow the owner to set or modify premium information.
- PlatformFeesVault.sol:
  - Owner:
    - setGmiAddress: This function allows the owner to set or modify the GMI address.
- MultiStrategy.sol:
  - Owner:
    - setStrategy: This function allows the owner to set or modify a strategy.
- Multipool.sol:
  - Operator:
    - rebalaceAll: Allows the operator to rebalance the pools with specific parameters.
  - Owner:
    - setQuotePoolAddress: Allows the owner to set the address of the quote pool.
    - addUnderlyingPool: Allow the owner to add an underlying pool.
    - setParam: Allows the owner to set certain parameters for the contract.
    - manageSwapTarget: Set/restrict permission to the aggregator's router to swap through.
- Factory.sol:
  - Owner:
    - attachWagmiTokenAddress: Allows the owner to attach the address of the Wagmi token.

        - createMultipool: Allows the owner to create a new multipool.
- Dispatcher.sol:
    - Owner:
        - setWagmiTokenAddress: Allows the owner to set the address of the Wagmi token.
        - add: Add new multipool to dispatcher.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **7** out of **10**.

- Functional requirements are partially missed:
  - Use of oracles is explained.
  - The system implies third-party constant control, the mechanism of this interaction is not explained.
- Technical description is robust:
  - Run instructions are provided.
  - Technical specification is provided.
  - NatSpec is sufficient.

Code quality

The total Code Quality score is **9** out of **10**.

- Best practice violations.

Test coverage

Code coverage of the project is **65%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is presented.

Security score

As a result of the audit, the code contains **no** issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.14**. The system users should acknowledge all the risks summed up in the risks section of the report.

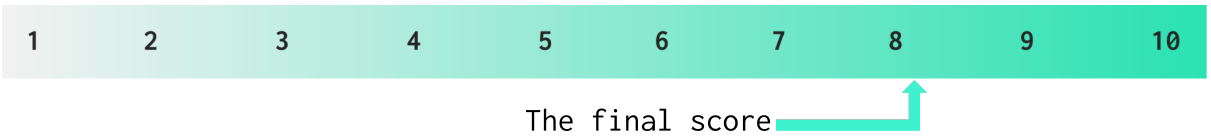

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
18 August 2023	3	3	0	0
13 September 2023	3	0	0	0
21 September 2023	0	0	0	0

Risks

- The logic of the Admin EOA or contract is out of scope.
- The GMI smart contract uses oracles to get prices for tokens. The code only describes that the oracle smart contract must implement the IOracle interface. The documentation is missing the details on which kind of oracles will be used. The implementation of functions for getting prices from the oracle is also unknown. Oracles' implementation is a significant part of the system, which is out of the audit scope and was not checked.
- The system highly depends on the Admin actions, Admin is capable to commit the next actions:
    - Set fees for the Strategies;
    - Set Uniswap pool address for specific tokens pair;
    - Increase vesting period up to 6900 hours;
    - Specify arbitrary Oracle for Wagmi-USD price within vesting contract.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Passed	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Passed	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. Contracts shouldn't rely on values that can be changed in the same transaction.	Passed	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	
Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed	
Style Guide Violation	Style guides and best practices should be followed.	Failed	I02, I04
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Passed	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

M01. Denial Of Service

Impact	Medium
Likelihood	Medium

The *GMI* contract allows burning *GMI* tokens and withdrawing *Wagmi* tokens to the vesting contract after at least 5 (*MINIMUM_LOCK_PERIOD_IN_BLOCKS*) blocks since the last mintage of the *GMI* tokens. However, it is possible to mint (*mintGmi*) a minimum amount of tokens to arbitrary *recipient* in order to block the ability to withdraw (*burnGmi*) tokens to the vesting.

This might lead to temporary inability of the entity to withdraw the token to the vesting.

Path: ./liquidity-commitment-contracts/contracts/GMI.sol : mintGmi()

Recommendation: Rework the function to disallow minting tokens to arbitrary addresses.

Found in: 7fda42f

Status: Fixed (Revised commit: 5c7e271)

M02. Denial Of Service

Impact	High
Likelihood	Low

The system of the contracts has Gas demanding computations due to the multiple loop iterations over the arrays. The list of the pools within the ecosystem is a permanently growing array. The system has no functionality to remove elements from the array, the maximum number of pools is not specified.

This might lead to DoS due to the growing Gas consumption caused by the pools array growth.

Path: ./liquidity-commitment-contracts/contracts/GMI.sol : : _earnAll(), _precalculateMint(), _withdrawPlatformShares(),

Recommendation: Add limitation on the number of pools in the system, or rework the system logic to avoid iterations over the full list of pools.

Found in: 7fda42f

Status: Fixed (Revised commit: 5c7e271)

M03. CEI Pattern Violation

Impact	Low
Likelihood	High

It is considered following best practices to avoid unclear situations and prevent common attack vectors.

The Checks-Effects-Interactions pattern is violated. During the function, some state variables are updated after the external calls.

This may lead to reentrancies, race conditions, and denial of service vulnerabilities during implementation of new functionality.

Paths: ./concentrator/contracts/Dispatcher.sol : deposit(), withdraw()

Recommendation: Follow common best practices and implement the function according to the Checks-Effects-Interactions pattern.

Found in: 3071cf4

Status: Mitigated (Revised commit: 5c7e271)

Low

L01. Data Inconsistency

Impact	Medium
Likelihood	Low

The admin might specify the maximum supply of the Multipool tokens, but the specified amount might exceed the existing supply.

This might lead to the confusions during the interactions with the system.

Path: ./concentrator/contracts/MultipoolToken.sol : setMaxTotalSupply()

Recommendation: Add input data validation to verify that the new supply limit does not contradict the existing supply value.

Found in: 3071cf4

Status: Mitigated (Revised commit: 241e81e)

L02. Missing Zero Address Validation

Impact	Low
Likelihood	Low

Additional checks against the *0x0* address should be included in the reported functions to avoid unexpected results.

Paths: ./concentrator/contracts/Factory.sol : constructor(), ./concentrator/contracts/Multipool.sol : constructor(), setQuotePoolAddress(), claimProtocolFees(), addUnderlyingPool(), ./concentrator/contracts/MultiStrategy.sol : setMultipool(),, ./concentrator/contracts/Dispatcher.sol : setWagmiTokenAddress(), add(); ./concentrator/contracts/PlatformFeesVault.sol : setGmiAddress(); ./concentrator/contracts/MultipoolToken.sol : setMaxTotalSupply(); ./liquidity-commitment-contracts/contracts/GMI.sol : constructor(); ./liquidity-commitment-contracts/contracts/WagmiVesting.sol : constructor();

Recommendation: It is recommended to add zero address checks.

Found in: 3071cf4, 7fda42f

Status: Mitigated (Revised commit: 241e81e)

L03. Missing Events

Impact	Low
Likelihood	Medium

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Paths:

./concentrator/contracts/Factory.sol : attachWagmiTokenAddress(); ./concentrator/contracts/Multipool.sol : constructor(), setQuotePoolAddress(), addUnderlyingPool(); ./concentrator/contracts/MultiStrategy.sol : setMultipool(),, ./concentrator/contracts/Dispatcher.sol : setWagmiTokenAddress(), , ./concentrator/contracts/PlatformFeesVault.sol : setGmiAddress(); ./concentrator/contracts/MultipoolToken.sol : setMaxTotalSupply(), ./liquidity-commitment-contracts/contracts/GMI.sol : setEarnBeforeMint(), earn(); ./liquidity-commitment-contracts/contracts/WagmiVesting.sol : startVesting();

Recommendation: Consider emitting events in the specified functions.

Found in: 3071cf4, 7fda42f

Status: Mitigated (Revised commit: 241e81e)

Informational

I01. Missing Variable Explicit Visibility

A variable does not have the visibility written explicitly in the code.

That leads to readability issues.

Path: ./concentrator/contracts/Multipool.sol : operator

Recommendation: Describe the variable visibilities explicitly in the code.

Found in: 3071cf4

Status: Fixed (Revised commit: 241e81e)

I02. Style Guide Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be ordered and grouped by their visibility as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)
- External functions
- Public functions
- Internal functions
- Private functions

Within each grouping, view and pure functions should be placed at the end.

Furthermore, following the Solidity naming convention and adding NatSpec annotations for all functions are strongly recommended. These measures aid in the comprehension of code and enhance overall code quality.

Paths: ./concentrator/contracts/Factory.sol, ./concentrator/contracts/Multipool.sol, ./concentrator/contracts/MultiStrategy.sol, ./concentrator/contracts/Dispatcher.sol, ./concentrator/contracts/PlatformFeesVault.sol, ./concentrator/contracts/DispatcherCode.sol, ./concentrator/contracts/MultipoolCode.sol, ./liquidity-commitment-contracts/contracts/GMI.sol, ./liquidity-commitment-contracts/contracts/WagmiVesting.sol

Recommendation:

Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts. Providing comprehensive NatSpec annotations for functions and following Solidity's naming conventions further enrich the quality of the code. Change order of layout to fit Official Style Guide.

Found in: 3071cf4, 7fda42f

Status: Reported (Revised commit: 5c7e271)

I03. Functions That Should Be External

Public functions that are not called from inside the contract should be declared external to save Gas.

Paths:

./concentrator/contracts/PlatformFeesVault.sol : setGmiAddress(); ./liquidity-commitment-contracts/contracts/GMI.sol : wagmiLockedOf(), totalLockedWagmi(), platformOwedLiquidity(), gmiPrice();

Recommendation: Consider changing the function visibility to external.

Found in: 3071cf4, 7fda42f

Status: Fixed (Revised commit: 241e81e)

I04. Missing Event Indexes

Use indexed events to keep track of a smart contract's activity after it is deployed, which is helpful in reducing overall Gas.

Path:
./concentrator/contracts/Factory.sol : CreateMultipool;
./concentrator/contracts/Multipool.sol : Deposit, Withdraw, Rebalance, SwapTargetApproved, ParamChanged, TrustedPoolAdded, FeesGrowth,
./concentrator/contracts/MultiStrategy.sol : SetNewStrategy, SetMultipool;
./concentrator/contracts/Dispatcher.sol : AddNewPool, Deposit, Withdraw, WagmiLossCompensation;
./liquidity-commitment-contracts/contracts/GMI.sol : AddPoolToGmi, MintGmi, BurnGmi, Earn, NotEnoughWagmiBalance;
./liquidity-commitment-contracts/contracts/WagmiVesting.sol : UpVestingPeriod, Exit, GetBackWagmi;

Recommendation: Add missed *indexed* keywords to easier tracking smart contract information.

Found in: 3071cf4, 7fda42f

Status: Reported (Revised commit: 5c7e271)

I05. Non-Explicit Variable Unit Sizes

Variable types uint are used without explicitly setting their size.

Path:
./concentrator/contracts/Multipool.sol : RebalanceParams{amountIn}

Recommendation: Set variable size explicitly for uint.

Found in: 3071cf4

Status: Fixed (Revised commit: 9e4f725)

I06. State Variables Can Be Declared Constant

Compared to regular state variables, the gas costs of constant variables are much lower.

Path:
./concentrator/contracts/Multipool.sol : protocolFeeWeightMax, protocolFeeWeight;

Recommendation: Declare mentioned variables as constant.

Found in: 3071cf4

Status: Fixed (Revised commit: 9e4f725)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/RealWagmi/concentrator
Commit	3071cf4
Whitepaper	https://docs.popsicle.finance/v/wagmi-1/
Requirements	https://docs.popsicle.finance/v/wagmi-1/
Technical Requirements	https://github.com/RealWagmi/concentrator/blob/main/README.md
Contracts	File: ./contracts/Factory.sol SHA3: c75994e07dba30c8ab769101bdd66907aee50750ef13441488bc11b0695c6a01 File: ./contracts/Multipool.sol SHA3: 68a09ef9284a2fe8a76071aa363a7ee365a54aef17d33f717939a0687c887227 File: ./contracts/MultiStrategy.sol SHA3: 6ae775ba104c065acea91c620b035753ddb1fd0237da1d68b66a73dfdcd0aa6f File: ./contracts/Dispatcher.sol SHA3: e8472bfb049956488d6595b919c6b55618e065f3053d5600399868c382dc8965 File: ./contracts/PlatformFeesVault.sol SHA3: 3d953c437eaf5ba16b1b830b9a5c9e2ea11a97532f0c6023292eb0aa35ad4c83 File: ./contracts/DispatcherCode.sol SHA3: 85df47b3fd87a165a13bfdf1325d3a9176823eadcf387832b8d7b59a1448255e File: ./contracts/MultipoolCode.sol SHA3: 5d0dfc5e1eea872606e1bbf52071bc1022ae3a1fcff140a268c514e9c4ffa1f4 File: ./contracts/libraries/ErrLib.sol SHA3: a0a1f72fcace6a10bf49dacbccda84874e3f3266f789318285ed7abe71340b99 File: ./contracts/interfaces/IMultipool.sol SHA3: 47e369dd4c6649752c0e380aed33b328b9a7c1bfc76129b0b406485cc6b16a5b File: ./contracts/interfaces/IDispatcher.sol SHA3: 93bdfc8cbf735026436d8ebe6f16653bff2964e4aec28a37d83173f4ce10ca7d File: ./contracts/interfaces/IFactory.sol SHA3: 021902a857706c4c054023452bdba25c0f54c6af6e5aec84e0118a1c52975f16 File: ./contracts/interfaces/IMultiStrategy.sol SHA3: 923978a31dd83b1f54a8e052673faa20c719d83fd29095daf251996e33ffcc0d File: ./contracts/interfaces/IPlatformFeesVault.sol SHA3: 58c50a98336c50cfcbd4aee6a0b62c988a3bbec0a2422babff85e9d72805ab90 File: ./contracts/interfaces/ICode.sol SHA3: e888a69f3eb6845a79802140dd5fc222d5d57edb8a5e5ecbe58b2c2719915fa0

Repository	https://github.com/ReakWagmi/liquidity-commitment-contracts
Commit	7fda42f
Whitepaper	https://docs.popsicle.finance/v/wagmi-1/
Requirements	https://docs.popsicle.finance/v/wagmi-1/
Technical Requirements	https://github.com/RealWagmi/liquidity-commitment-contracts/blob/main/README.md
Contracts	File: ./contracts/GMI.sol SHA3: d936dc47fe27d2ec7dcc451911ad27ba9188d58d377ffff9b802a37032e00ab9 File: ./contracts/WagmiVesting.sol SHA3: 5e5bc50a81e4b7ef23667a38cd0eb6388356b0f3d5a8a333201ab0935dbebf53 File: ./contracts/interfaces/IOracle.sol SHA3: 7d13aad1127afc6a049a5eb090b4ba2d7ece6957237fdc593561cf6b4fb1dd50 File: ./contracts/interfaces/IWagmiVesting.sol SHA3: 6b1bc898bd0566eb6ebcf55a45fdfe074d5267ce51b118ce8759cd2760c71096

Second review scope

Repository	https://github.com/RealWagmi/concentrator
Commit	81fb51a
Whitepaper	https://docs.popsicle.finance/v/wagmi-1/
Requirements	https://docs.popsicle.finance/v/wagmi-1/
Technical Requirements	https://github.com/RealWagmi/concentrator/blob/main/README.md
Contracts	File: ./contracts/Dispatcher.sol SHA3: a5221bcfb6a8cc7d7f386b7c3585603f8fde059f134ce6268a8a380cf429db22 File: ./contracts/DispatcherCode.sol SHA3: 85df47b3fd87a165a13bfdf1325d3a9176823eadcf387832b8d7b59a1448255e File: ./contracts/Factory.sol SHA3: c75994e07dba30c8ab769101bdd66907aee50750ef13441488bc11b0695c6a01 File: ./contracts/Multipool.sol SHA3: dcb55624ed875b8f195fcde46bf97acf5fe42df9964fa32c02e321e903feea37 File: ./contracts/MultipoolCode.sol SHA3: 5d0dfc5e1eea872606e1bbf52071bc1022ae3a1fcff140a268c514e9c4ffa1f4 File: ./contracts/MultipoolToken.sol SHA3: 24344050948f78f7eeb08859c997df630656aa918b087681d0969fcb09847f5f File: ./contracts/MultiStrategy.sol SHA3: c7ebd67b7db3f6039436055836f72ad9c0fe0c9f81eae2c9a3832277a3232a55

	File: ./contracts/PlatformFeesVault.sol SHA3: 9dfaebdcc955435bbb4500e080c38a77dae4a45cdab9ed71897870383af39cb2 File: ./contracts/interfaces/ICode.sol SHA3: e888a69f3eb6845a79802140dd5fc222d5d57edb8a5e5ecbe58b2c2719915fa0 File: ./contracts/interfaces/IDispatcher.sol SHA3: e48cbb93ca218161b2656d8693adb12f27f743fbd69f93e39c1c4670a4fae245 File: ./contracts/interfaces/IFactory.sol SHA3: 021902a857706c4c054023452bdba25c0f54c6af6e5aec84e0118a1c52975f16 File: ./contracts/interfaces/IMultipool.sol SHA3: 1a238abfbb149f52c9b1c09ec71ada38c1a4d9e9358dd20e703c850f8c84cba4 File: ./contracts/interfaces/IMultipoolToken.sol SHA3: e6e30bc081f822aa1d94cfcf173952422bda0325f5ca8561c728de07c94ff312 File: ./contracts/interfaces/IMultiStrategy.sol SHA3: 9400d2a4a683b4752492df72be6fd9988c563ac8df2e22ce505cf77d82b71fd3 File: ./contracts/interfaces/IPlatformFeesVault.sol SHA3: 58c50a98336c50cfcbd4aee6a0b62c988a3bbec0a2422babff85e9d72805ab90 File: ./contracts/libraries/ErrLib.sol SHA3: a0a1f72fcace6a10bf49dacbccda84874e3f3266f789318285ed7abe71340b99

Repository	https://github.com/ReakWagmi/liquidity-commitment-contracts
Commit	5c7e271
Whitepaper	https://docs.popsicle.finance/v/wagmi-1/
Requirements	https://docs.popsicle.finance/v/wagmi-1/
Technical Requirements	https://github.com/RealWagmi/liquidity-commitment-contracts/blob/main/README.md
Contracts	File: ./contracts/GMI.sol SHA3: c84b6c39c551d380ea1f38b5eacfaa32cd6190beb3332e36c5f0355a318a55c1 File: ./contracts/WagmiVesting.sol SHA3: ca939418d9d5634bd7ccc8813e6e2d4a0755844c32b595bdea4659628f621017 File: ./contracts/interfaces/IOracle.sol SHA3: a7bb79907e9e1cee069428d162800bd129dfccd65e9c7062e55258d077668265 File: ./contracts/interfaces/IWagmiVesting.sol SHA3: 6b1bc898bd0566eb6ebcf55a45fdfe074d5267ce51b118ce8759cd2760c71096

Third review scope

Repository	https://github.com/RealWagmi/concentrator
Commit	9e4f725

Whitepaper	https://docs.popsicle.finance/v/wagmi-1/
Requirements	https://docs.popsicle.finance/v/wagmi-1/
Technical Requirements	https://github.com/RealWagmi/concentrator/blob/main/README.md
Contracts	File: ./contracts/Dispatcher.sol SHA3: a5221bcfb6a8cc7d7f386b7c3585603f8fde059f134ce6268a8a380cf429db22 File: ./contracts/DispatcherCode.sol SHA3: 85df47b3fd87a165a13bfdf1325d3a9176823eadcf387832b8d7b59a1448255e File: ./contracts/Factory.sol SHA3: c75994e07dba30c8ab769101bdd66907aee50750ef13441488bc11b0695c6a01 File: ./contracts/Multipool.sol SHA3: 94371ee59dd6569000d79c55000cf1c2cff13447ad2fe20fc262f86c78dfb14b File: ./contracts/MultipoolCode.sol SHA3: 5d0dfc5e1eea872606e1bbf52071bc1022ae3a1fcff140a268c514e9c4ffa1f4 File: ./contracts/MultipoolToken.sol SHA3: 9496a3617e7f4155c12ea9d17dd8183d8702cc132ad55dd7add5701aa1e40cfe File: ./contracts/MultiStrategy.sol SHA3: c7ebd67b7db3f6039436055836f72ad9c0fe0c9f81eae2c9a3832277a3232a55 File: ./contracts/PlatformFeesVault.sol SHA3: 214f0c476eb4d77f9f8d14462a6626a5d5c259b5d9c4fc2b02fb99b2e245cc96 File: ./contracts/interfaces/ICode.sol SHA3: e888a69f3eb6845a79802140dd5fc222d5d57edb8a5e5ecbe58b2c2719915fa0 File: ./contracts/interfaces/IDispatcher.sol SHA3: e48cbb93ca218161b2656d8693adb12f27f743fbd69f93e39c1c4670a4fae245 File: ./contracts/interfaces/IFactory.sol SHA3: 021902a857706c4c054023452bdba25c0f54c6af6e5aec84e0118a1c52975f16 File: ./contracts/interfaces/IMultipool.sol SHA3: 1a238abfbb149f52c9b1c09ec71ada38c1a4d9e9358dd20e703c850f8c84cba4 File: ./contracts/interfaces/IMultipoolToken.sol SHA3: e6e30bc081f822aa1d94cfcf173952422bda0325f5ca8561c728de07c94ff312 File: ./contracts/interfaces/IMultiStrategy.sol SHA3: 9400d2a4a683b4752492df72be6fd9988c563ac8df2e22ce505cf77d82b71fd3 File: ./contracts/interfaces/IPlatformFeesVault.sol SHA3: 58c50a98336c50cfcbd4aee6a0b62c988a3bbec0a2422babff85e9d72805ab90 File: ./contracts/libraries/ErrLib.sol SHA3: a0a1f72fcace6a10bf49dacbccda84874e3f3266f789318285ed7abe71340b99

Repository	https://github.com/ReakWagmi/liquidity-commitment-contracts
Commit	241e81e

Whitepaper	https://docs.popsicle.finance/v/wagmi-1/
Requirements	https://docs.popsicle.finance/v/wagmi-1/
Technical Requirements	https://github.com/RealWagmi/liquidity-commitment-contracts/blob/main/README.md
Contracts	File: ./contracts/GMI.sol SHA3: 5ac297ef215c11ccc36b8c1c8c2d1cea2602e85085f9480f674470ff259b941e File: ./contracts/WagmiVesting.sol SHA3: ab702379405133b09da2618c40d5943e87187b6170289d501ee624e8faf0eb5e File: ./contracts/interfaces/IOracle.sol SHA3: a7bb79907e9e1cee069428d162800bd129dfccd65e9c7062e55258d077668265 File: ./contracts/interfaces/IWagmiVesting.sol SHA3: 6b1bc898bd0566eb6ebcf55a45fdfe074d5267ce51b118ce8759cd2760c71096