1inch Mooniswap v2

Security Assessment

December 16th, 2020

For :
1inch Mooniswap v2

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Camden Smallwood @ CertiK
camden.smallwood@certik.org

Disclaimer

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	1inch Mooniswap v2
Description	An upgrade to the Mooniswap v2 implementation introducing governance, a new slippage fee model and a new mechanism for the referral fee distribution.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. ca55668d1fe0abe7d9368a4ac3ac09affdf04aca 2. e9c6a03ab28f1c486ed73c92a30d5a283aed2014

Audit Summary

Delivery Date	December 16th, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	December 2nd, 2020 - December 16th, 2020

Vulnerability Summary

Total Issues	36
Total Critical	0
Total Major	0
Total Medium	3
Total Minor	5
Total Informational	28

Executive Summary

The 1inch team approached us to conduct an audit of their upgraded Mooniswap implementation signaling a v2 release. The contracts in question compose a DEX system whereby liquidity pools are created via a single point of entry, a factory, and users add liquidity to the pools and trade the supported pairs. The system boasts a governance system via which changes to the various parameters of the protocol are proposed and voted on in a decentralized fashion.

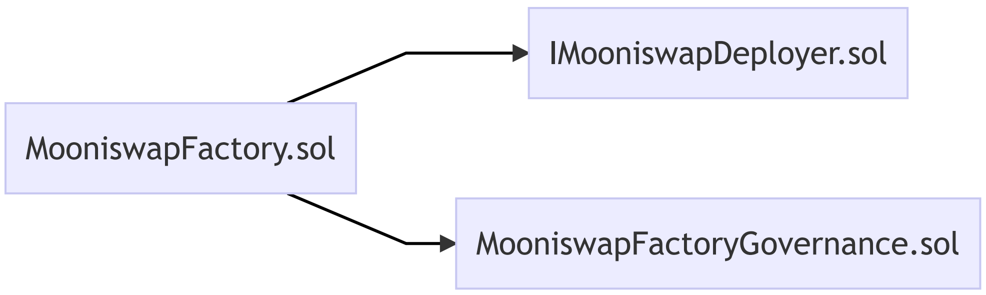
The codebase contains extensive test cases and has been developed conforming to the latest Solidity standards. During the audit, we applied static analysis tools and focused more on our manual audit process to identify any potential flaws or attack vectors the system may contain. Additionally, we analysed the system from an optimizational standpoint to ensure that the gas footprint of the overall system is reduced to the lowest possible.

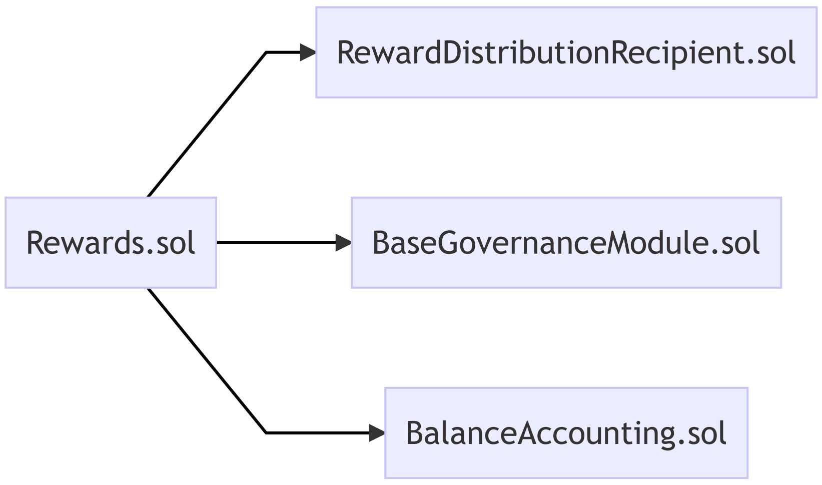
Over the course of the audit we identified 3 medium-level findings that affected the integrity of the governance and referral system which were promptly alleviated or responded to by the 1inch team. Additionally, we were able to identify 5 more misbehaviours of the system which were also swiftly dealt with. All informational exhibits identified within the report were safe to ignore as they mostly related to either optimizations or coding style and due to time constraints some of them were chosen to not be applied and were simply noted by the 1inch team.

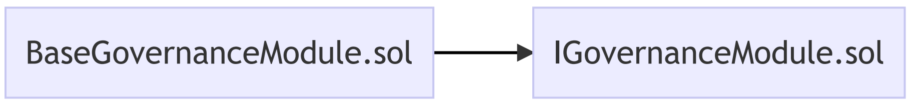
Files In Scope

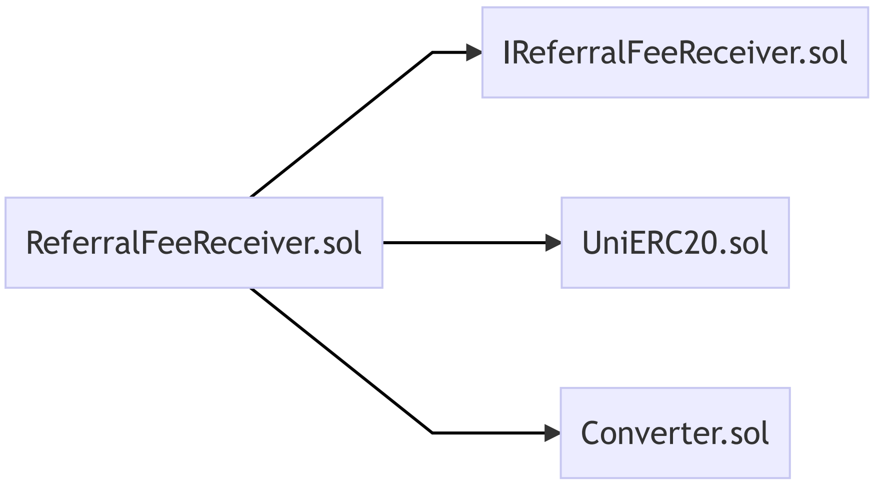
ID	Contract	Location
BAG	BalanceAccounting.sol	contracts/utils/BalanceAccounting.sol
BGM	BaseGovernanceModule.sol	contracts/governance/BaseGovernanceModule.sol
CON	Converter.sol	contracts/utils/Converter.sol
GMP	GovernanceMothership.sol	contracts/inch/GovernanceMothership.sol
GFR	GovernanceFeeReceiver.sol	contracts/governance/GovernanceFeeReceiver.sol
IGM	IGovernanceModule.sol	contracts/interfaces/IGovernanceModule.sol
IMF	IMooniswapFactory.sol	contracts/interfaces/IMooniswapFactory.sol
IMD	IMooniswapDeployer.sol	contracts/interfaces/IMooniswapDeployer.sol
IRF	IReferralFeeReceiver.sol	contracts/interfaces/IReferralFeeReceiver.sol
IMG	IMooniswapFactoryGovernance.sol	contracts/interfaces/IMooniswapFactoryGovernance.sol
LVG	LiquidVoting.sol	contracts/libraries/LiquidVoting.sol
MOO	Mooniswap.sol	contracts/Mooniswap.sol
MFY	MooniswapFactory.sol	contracts/MooniswapFactory.sol
MDR	MooniswapDeployer.sol	contracts/MooniswapDeployer.sol
MCS	MooniswapConstants.sol	contracts/MooniswapConstants.sol
MGE	MooniswapGovernance.sol	contracts/governance/MooniswapGovernance.sol
MFG	MooniswapFactoryGovernance.sol	contracts/governance/MooniswapFactoryGovernance.sol
REW	Rewards.sol	contracts/governance/rewards/Rewards.sol
RFR	ReferralFeeReceiver.sol	contracts/ReferralFeeReceiver.sol
RDR	RewardDistributionRecipient.sol	contracts/governance/rewards/RewardDistributionRecipient.sol
SQR	Sqrt.sol	contracts/libraries/Sqrt.sol
UER	UniERC20.sol	contracts/libraries/UniERC20.sol
VOT	Vote.sol	contracts/libraries/Vote.sol
VBE	VirtualBalance.sol	contracts/libraries/VirtualBalance.sol

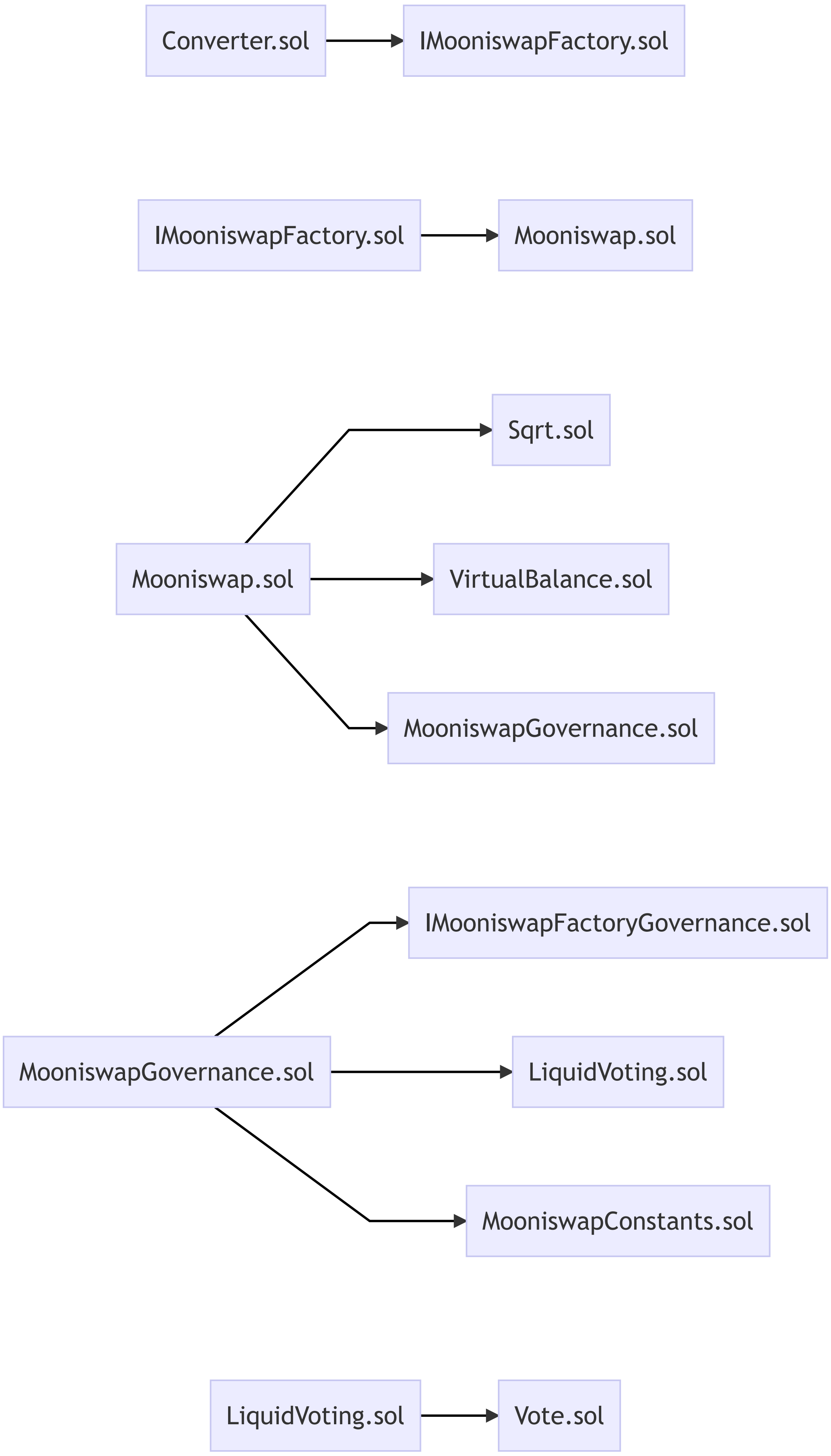
File Dependency Graph (BETA)



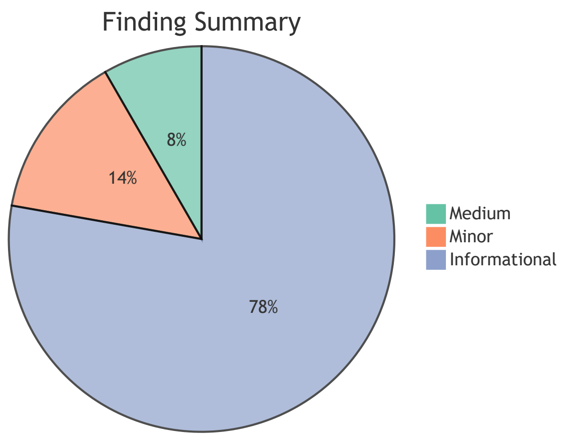







Converter.sol
IMooniswapFactory.sol
IMooniswapFactory.sol
Mooniswap.sol
Sqrt.sol
Mooniswap.sol
VirtualBalance.sol
MooniswapGovernance.sol
IMooniswapFactoryGovernance.sol
MooniswapGovernance.sol
LiquidVoting.sol
MooniswapConstants.sol
LiquidVoting.sol
Vote.sol

Findings



ID	Title	Type	Severity	Resolved
BAG-01	User-Defined Getters	Gas Optimization	Informational	✓
BAG-02	`SafeMath` Redundancy	Gas Optimization	Informational	✓
GMP-01	Incorrect Stake Notification Implementation	Logical Issue	Medium	✓
GFR-01	Misleading Error Message	Inconsistency	Informational	✓
GFR-02	Inexistent Access Control	Control Flow	Minor	✓
LVG-01	`SafeMath` Redundancy	Gas Optimization	Informational	✓
SQR-01	Babylonian Method Optimization	Gas Optimization	Informational	ⓘ
UER-01	`SafeMath` Redundancy	Gas Optimization	Informational	✓
UER-02	Incorrect `len` Boundry Check	Logical Issue	Informational	✓
UER-03	Assembly-based Optimization	Gas Optimization	Informational	ⓘ
UER-04	Redundant Function Implementation	Gas Optimization	Informational	ⓘ
UER-05	Potential Code Redundancy	Gas Optimization	Informational	✓
VBE-01	`SafeMath` Redundancy	Gas Optimization	Informational	✓
VOT-01	Redundant Nesting	Coding Style	Informational	ⓘ
MFG-01	`SafeMath` Redundancy	Gas Optimization	Informational	✓
MFG-02	Variable Typos	Coding Style	Informational	✓
MFG-03	`require` to `modifier`	Gas Optimization	Informational	ⓘ
MGE-01	Incorrect `to` Conditional	Logical Issue	Minor	✓

ID	Title	Type	Severity	Resolved
MGE-02	Redundant `isDefault` Invocation	Gas Optimization	Informational	✓
REW-01	`SafeMath` Redundancy	Gas Optimization	Informational	✓
MOO-01	Event Indexing	Inconsistency	Informational	✓
MOO-02	Disproportionate Initial Minting	Logical Issue	Minor	✓
MOO-03	Incorrect Transfer Evaluation	Logical Issue	Minor	✓
MOO-04	Midway Condition Evaluation	Gas Optimization	Informational	!
MOO-05	Loop Optimization	Gas Optimization	Informational	!
MOO-06	Function Visibility Optimization	Gas Optimization	Informational	!
MOO-07	Incorrect Implementation / Naming Convention	Gas Optimization	Informational	✓
MOO-08	`require` Consistency	Inconsistency	Informational	✓
MOO-09	Inexistent Input Sanitization	Logical Issue	Medium	✓
MOO-10	Documentation Consistency	Inconsistency	Informational	✓
MOO-11	Redundant Use of Dynamic Arrays	Inconsistency	Informational	✓
RFR-01	`SafeMath` Redundancy	Gas Optimization	Informational	✓
RFR-02	Unfair Proportionate Calculation	Mathematical Operations	Informational	✓
RFR-03	Prohibition of Ether Transfers	Logical Issue	Informational	✓
RFR-04	Inexistent Input Sanitization of Mooniswap Addresses	Logical Issue	Medium	✓

ID	Title	Type	Severity	Resolved
CON-01	Whitelist Evaluation	Control Flow	Minor	✓

BAG-01: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	BalanceAccounting.sol L11, L14-L16

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The 1inch team decided to leave the variable as is to properly depict that it is meant to be altered only by the `internal` functions of the `BalanceAccounting` contract.

BAG-02: `SafeMath` Redundancy

Type	Severity	Location
Gas Optimization	Informational	BalanceAccounting.sol L29

Description:

The linked statement conducts a `SafeMath` subtraction operation on the `_totalSupply` variable based on the amount burned from the `_balances` mapping. As the contract level variables of the contract are `private`, they cannot be altered by derivative implementations and as such, the `_totalSupply` is kept in perfect sync with the amount of total balance held in the `_balances` mapping. As a result, if a `SafeMath` subtraction operation succeeds on the `_balances` mapping it is guaranteed to succeed in the `_totalSupply` variable as well, rendering its use redundant gas-wise.

Recommendation:

We advise that the `SafeMath` utilization from this point is omitted. For the sake of clarity, a comment may be added to aid in future auditing endeavors.

Alleviation:

The 1inch team preferred to leave the redundancy as is for the sake of readability.

GMP-01: Incorrect Stake Notification Implementation

Type	Severity	Location
Logical Issue	Medium	GovernanceMothership.sol L49

Description:

The `notifyFor` function is meant to be used to allow any address to force a status update of another `address` arbitrarily to ensure the stakes states are kept in sync. However, the second argument passed to the `_notifyFor` function is the balance of the `msg.sender` instead of the account to-be-notified, causing an incorrect balance to be reported for it and potentially for as many accounts as a particular `msg.sender` wants.

Recommendation:

We advise that the second argument is instead changed to the `balanceOf` of the actual account in the same way the `batchNotifyFor` implementation functions.

Alleviation:

The `balanceOf` measurement was properly fixed in this commit hash as per our recommendation.

GFR-01: Misleading Error Message

Type	Severity	Location
Inconsistency	Informational	GovernanceFeeReceiver.sol L14

Description:

The error message alludes that ETH transfers are completely forbidden to the contract whereas transfers from contracts are allowed.

Recommendation:

We advise that the error message is revised to properly reflect the check's purpose.

Alleviation:

The issue was acknowledged but no action was taken to alter the error message as it is believed to be sufficiently descriptive. To note, the contracts have been revised in the latest commit hash to instead rely on a `receive` implementation in the `Converter` contract.

GFR-02: Inexistent Access Control

Type	Severity	Location
Control Flow	Minor	GovernanceFeeReceiver.sol L23-L25, L27-L31

Description:

The linked functions are meant to conduct sensitive operations on the contract and utilize its full balance to conduct a swap. Even though the target of a given `swap` operation will always be the `rewards` address specified during construction time, a malicious path can lead to multiple pairs being exchanged artificially increasing volume and diminishing the final output sent to the `rewards` address.

Recommendation:

We advise that both the `unwrapLPTokens` and `swap` functions are guarded via a check to ensure an authorized member is conducting those operations either via a governance or ownership system.

Alleviation:

The 1inch team reported that sufficient ACL checks are imposed in the form of valid conversion checks within the `Converter.sol` implementation whereby a `path` of fixed length and whitelisted addresses is guaranteed, a slippage check is imposed as well as a check that at most 1% of available liquidity within the pool is swapped. These are considered sufficient measures against malicious trades.

LVG-01: `SafeMath` Redundancy

Type	Severity	Location
Gas Optimization	Informational	LiquidVoting.sol L30, L31, L34

Description:

The linked mathematical operations utilize their `SafeMath` counterpart implementation whereas it is completely reudndant for the operations of L31 and L34 as well as potentially redundant for the operation of L30.

Recommendation:

We advise that its usage is omitted for the guaranteed operations and evaluated for the potential operation to optimize gas costs.

Alleviation:

The 1inch team preferred to leave the redundancy as is for the sake of readability.

SQR-01: Babylonian Method Optimization

Type	Severity	Location
Gas Optimization	Informational	Sqrt.sol L8-L22

Description:

The linked Babylonian Square Root implementation can be further optimized by omitting the final `else` block as Solidity returns zeroed out values by default regardless of whether the return variable has been explicitly named or not. Additionally, we highly advise that the new, more optimized Babylonian Method from ABDK Consulting is evaluated as an optimized alternative which has already been integrated in Uniswap.

Recommendation:

Included in the description.

Alleviation:

The 1inch Mooniswap v2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

UER-01: `SafeMath` Redundancy

Type	Severity	Location
Gas Optimization	Informational	UniERC20.sol L42

Description:

The linked mathematical statement conducts a `SafeMath` operation whilst its safety is guaranteed by the preceding `if` clause.

Recommendation:

We advise that the `SafeMath` usage is omitted from the codebase to optimize gas cost.

Alleviation:

The 1inch team preferred to leave the redundancy as is for the sake of readability.

UER-02: Incorrect `len` Boundry Check

Type	Severity	Location
Logical Issue	Informational	UniERC20.sol L66

Description:

The boundary check of the `len` variable is meant to ensure that the `len` variable can fit within a single 8-bit slot by ensuring its within its valid range. However, the upper bound specified is incorrect.

Recommendation:

We advise that the upper bound is adjusted to a strict less-than (`<`) comparison with 256 as a byte cannot retain the number `256` since its valid range is between `0-255` in its unsigned representation.

Alleviation:

The 1inch team responded by stating that the check is not meant to ensure the length fits within 1 `byte` but rather that the length is simply within the specified bound as they decided to not support tokens with a name larger than `256` bytes.

UER-03: Assembly-based Optimization

Type	Severity	Location
Gas Optimization	Informational	UniERC20.sol L78-L82

Description:

The linked code block creates a new in-memory `bytes` array and assigns each byte sequentially to it via a loop from the `data` array.

Recommendation:

We advise that this code block is refactored in `assembly` to be heavily optimized. The EVM is meant to operate on 32-byte datasets and as such, the array could be copied 32-bytes at a time at a much cheaper gas cost than it currently is being done so.

Alleviation:

The 1inch Mooniswap v2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

UER-04: Redundant Function Implementation

Type	Severity	Location
Gas Optimization	Informational	UniERC20.sol L86, L89-L91

Description:

The function `_toHex` is implemented twice utilizing function overloading, however the middleware implementation accepting an `address` argument is only used once.

Recommendation:

We advise that the middleware implementation of `_toHex` is omitted and that L86 invokes the actual implementation directly to avoid the superfluous function call.

Alleviation:

The 1inch Mooniswap v2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

UER-05: Potential Code Redundancy

Type	Severity	Location
Gas Optimization	Informational	UniERC20.sol L86

Description:

The linked line performs an on-chain convertion from the `address` low-level representation to its human-readable hexadecimal representation.

Recommendation:

As the conversion between raw bytes and hexadecimal does not affect its usability as a unique key, we advise that its actual need is evaluated as the binary to hex conversion could potentially happen completely off-chain to optimize the gas costs necessitated by on-chain operations.

Alleviation:

The 1inch team stated that the purpose of the on-chain conversion is to aid in the readability of token names in blockchain explorers like Etherscan, thus nullifying the validity of this exhibit.

VBE-01: `SafeMath` Redundancy

Type	Severity	Location
Gas Optimization	Informational	VirtualBalance.sol L32, L33, L36

Description:

The linked statements conduct `SafeMath` operations when their safety may be guaranteed by the system for the first and last linked statements and is fully guaranteed for the middle statement.

Recommendation:

We advise that the `SafeMath` utilizations are evaluated and omitted where possible to optimize gas costs.

Alleviation:

The 1inch team preferred to leave the redundancy as is for the sake of readability.

VOT-01: Redundant Nesting

Type	Severity	Location
Coding Style	Informational	Vote.sol L7-L9

Description:

The `Vote` library is meant to be utilized on `Data` structs which contain only a single `uint256` value.

Recommendation:

We advise that the library is instead applied directly on the `uint256` data type to decrease code ambiguity when considering nested statements such as L22-L24.

Alleviation:

The 1inch Mooniswap v2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

MFG-01: `SafeMath` Redundancy

Type	Severity	Location
Gas Optimization	Informational	MooniswapFactoryGovernance.sol L147, L149

Description:

The linked statements both contain redundant operations as they are nested within an `if-else` block that guarantees their validity.

Recommendation:

We advise that these `SafeMath` statements are omitted.

Alleviation:

The 1inch team preferred to leave the redundancy as is for the sake of readability.

MFG-02: Variable Typos

Type	Severity	Location
Coding Style	Informational	MooniswapFactoryGovernance.sol L162, L163, L166, L167

Description:

The linked code lines contain a mispelling of the `default` word as `defaul`, leading to mistyped variable names.

Recommendation:

We advise that these variable names are corrected.

Alleviation:

The variable names were corrected according to our recommendation in the linked commit hash.

MFG-03: `require` to `modifier`

Type	Severity	Location
Gas Optimization	Informational	MooniswapFactoryGovernance.sol L98, L107, L116, L117, L126, L127, L136

Description:

The linked `require` statements could instead be coded in a small `internal` or `private` function that is invoked by a corresponding `modifier` to reduce the gas footprint of the contract.

Recommendation:

We advise that the suggestion explained in the description is implemented.

Alleviation:

The 1inch Mooniswap v2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

MGE-01: Incorrect `to` Conditional

Type	Severity	Location
Logical Issue	Minor	MooniswapGovernance.sol L102

Description:

The linked `_beforeTokenTransfer` hook incorrectly sanitizes burn operations by evaluating whether the `from` instead of the `to` variable is different than the zero address for the ternary operator used on the `balanceTo` assignment.

Recommendation:

We advise that the ternary evaluation instead utilizes the `to` variable.

Alleviation:

The ternary operator was properly fixed according to our recommendation in the linked commit hash.

MGE-02: Redundant `isDefault` Invocation

Type	Severity	Location
Gas Optimization	Informational	MooniswapGovernance.sol L142, L143

Description:

The `if` conditional that precedes those two statements already evaluates whether the vote is the default one, meaning a literal can be passed here instead.

Recommendation:

We advise that a literal is instead passed here to optimize the gas cost of those statements.

Alleviation:

The optimization we advised was applied in this commit hash.

REW-01: `SafeMath` Redundancy

Type	Severity	Location
Gas Optimization	Informational	Rewards.sol L75, L77, L97, L99, L101

Description:

The linked statements all conduct `SafeMath` operations whilst the safety of their operations is guaranteed by the system either via `if-else` clauses or `constant` variable utilizations.

Recommendation:

We advise that the `SafeMath` utilizations from these points are omitted. For the sake of clarity, comments may be added to aid in future auditing endeavors.

Alleviation:

The 1inch team preferred to leave the redundancy as is for the sake of readability.

MOO-01: Event Indexing

Type	Severity	Location
Inconsistency	Informational	Mooniswap.sol L57

Description:

The `Swapped` event is indexing the source token, however the destination token remains unindexed.

Recommendation:

We advise that either both or neither of the two tokens involved in the swap are indexed to ensure consistency in the codebase.

Alleviation:

After conversing with the 1inch team, we identified that our initial suggestion was incorrect as it is not possible to add more than 3 `indexed` fields to an event and as such, this exhibit is considered void.

MOO-02: Disproportionate Initial Minting

Type	Severity	Location
Logical Issue	Minor	Mooniswap.sol L140-L153

Description:

The initial minting process of a Mooniswap appears to be disproportionate in the sense that it does not rely on the initial deposit, meaning that a user can simply deposit only a single unit and acquire a mint equal to `_BASE_SUPPLY.mul(99)` as only a `Math.max` operation is used on L145.

Recommendation:

We advise that the validity of such a mint is evaluated and the corresponding code segments are refactored if it is deemed unfair.

Alleviation:

The 1inch team responded by stating that the initial mint donates 1% of the minted amount to the pool to ensure that underfunded pools cannot be redeemed i.e. a deposit of a single unit for the creation of the pool will render that unit unredeemable. Additionally, consequent mints are properly evaluated to be proportionate thus diminishing the percentage of the original mint.

MOO-03: Incorrect Transfer Evaluation

Type	Severity	Location
Logical Issue	Minor	Mooniswap.sol L150-L151

Description:

The linked code invokes `uniTransferFrom` for an amount equal to `maxAmounts[i]` and consequently assigns that value to `receivedAmounts[i]` even if a token imposes fees on the transaction.

Recommendation:

We advise that the `uniBalanceOf` evaluation paradigm that is already utilized in L174 is also utilized here to ensure compatibility with tokens that carry a transfer fee.

Alleviation:

The 1inch team stated that the actual received amount can be safely ignored for the first deposit as it is not weighted for the minting process.

MOO-04: Midway Condition Evaluation

Type	Severity	Location
Gas Optimization	Informational	Mooniswap.sol L147, L169

Description:

The linked `require` statements ensure that the `maxAmounts` provided to the contract at the start are greater-than (`>`) zero, yet they are evaluated mid-way through.

Recommendation:

We advise that both elements of the `maxAmounts` array are evaluated at the very start to ensure no gas is wasted in the statements that precede them if the function is going to fail regardless.

Alleviation:

The 1inch Mooniswap v2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

MOO-05: Loop Optimization

Type	Severity	Location
Gas Optimization	Informational	Mooniswap.sol L155-L182

Description:

The linked code block contains four `for` loops to ultimately conduct all operations necessary on the 2-member `maxAmounts` and `realBalances` arrays.

Recommendation:

We advise that the block is refactored to utilize a single loop as we believe some or all of the `for` loops can be grouped into one and significantly reduce gas cost.

Alleviation:

The 1inch Mooniswap v2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

MOO-06: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	Informational	Mooniswap.sol L130, L134, L191, L195

Description:

The linked functions are declared as `external` or `public`, contain array function arguments and are meant to be mostly invoked by external parties.

Recommendation:

We advise that the functions' visibility specifiers are set to `public` or `external` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas costs of the functions.

Alleviation:

The 1inch Mooniswap v2 development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

MOO-07: Incorrect Implementation / Naming Convention

Type	Severity	Location
Gas Optimization	Informational	Mooniswap.sol L322-L327

Description:

The naming convention utilizing `tax` alludes to a tax being imposed on the balances, however L323 assigns the addition instead of the subtraction of the tax from the `srcBalance`.

Recommendation:

We advise that either the naming convention or the implementation are adjusted to properly reflect what they are meant to achieve.

Alleviation:

The 1inch team revised the codebase in the linked commit hash to contain more legible variable naming.

MOO-08: `require` Consistency

Type	Severity	Location
Inconsistency	Informational	Mooniswap.sol L209

Description:

The project imposes `require` checks on multiple functions, like L255 of `_doTransfers`, to ensure that zero value swaps etc. are prohibited. The `withdrawFor` function fails to impose such a check as it merely ensures that the `value` is greater-than-or-equal to `minReturns[i]` which itself could be `0`.

Recommendation:

We advise that an additional check akin to `_doTransfers` is imposed here as well to ensure consistency.

Alleviation:

After consulting with the 1inch team, we concluded that the `require` check imposed in the linked lines is actually carried out properly as tokens with a small number of decimal places can indeed yield a zero amount whereas their counterpart within the pool may yield a non-zero amount. Consequently, this exhibit is nullified.

MOO-09: Inexistent Input Sanitization

Type	Severity	Location
Logical Issue	Medium	Mooniswap.sol L222

Description:

The `src` and `dst` arguments of the `swapFor` function are not gauranteed to be equal to the supported tokens of the exchange. While `_doTransfers` will simply yield 0 for the `confirmed` variable on unsupported tokens, this can lead to incorrect reward minting on `_mintRewards` as the confirmed balance is simply utilized during reward calculation.

Recommendation:

We advise that the `if` conditional of `_doTransfers` is instead changed to a `require` check to ensure only the supported tokens are ever swapped on the Mooniswap exchange.

Alleviation:

The 1inch team stated that a check is actually imposed within `_doTransfers` on the result of the invocation of `_getReturn` which will yield `0` for unsupported tokens, thus preventing unsupported tokens from being accepted by the `_doTransfers` fucntion.

MOO-10: Documentation Consistency

Type	Severity	Location
Inconsistency	Informational	Mooniswap.sol L304-L316

Description:

The linked documentation block is meant to explain the calculations carried out in `_getReturn`, however the variable naming conventions of the function block do not conform to what is laid out in the documentation and the documentation itself appears inconsistent with the statements of the function.

Recommendation:

We advise that either the documentation or the variable naming conventions are updated correspondingly to increase the legibility of this particular code block.

Alleviation:

The 1inch team stated that the accompanying comments are meant to paint the greater picture with regards to the formula the Mooniswap system is utilizing whereas the last segment, `(ret = dx * y / (x + dx) * (x + dx - slip_fee * dx) / (x + dx))`, is what is being implemented by the formula of the function.

MOO-11: Redundant Use of Dynamic Arrays

Type	Severity	Location
Inconsistency	Informational	Mooniswap.sol L191, L195

Description:

The functions of the Mooniswap contract accept statically sized arrays of 2 members whereas the `withdraw` prefixed functions accept dynamically sized arrays whose size is ensured to be `2` within their respective code block.

Recommendation:

We advise that they too are adjusted to be statically sized arrays to reduce the gas cost of invoking the functions and ensuring consistency in the codebase.

Alleviation:

The 1inch team stated that the arrays are dynamic to allow users to optionally specify `minReturns`. However, for the sake of gas optimization, we still believe the system can be revised as a value of `0` on both members would render them optional since the `require` conditional `value >= minReturns[i]` would always yield `true` due to the usage of `uint256`.

RFR-01: `SafeMath` Redundancy

Type	Severity	Location
Gas Optimization	Informational	ReferralFeeReceiver.sol L57, L72

Description:

The linked statement conducts a `SafeMath` subtraction operation on the `_totalSupply` variable based on the amount burned from the `_balances` mapping. As the contract level variables of the contract are `private`, they cannot be altered by derivative implementations and as such, the `_totalSupply` is kept in perfect sync with the amount of total balance held in the `_balances` mapping. As a result, if a `SafeMath` subtraction operation succeeds on the `_balances` mapping it is guaranteed to succeed in the `_totalSupply` variable as well, rendering its use redundant gas-wise.

Recommendation:

We advise that the `SafeMath` utilization from this point is omitted. For the sake of clarity, a comment may be added to aid in future auditing endeavors.

Alleviation:

The 1inch team preferred to leave the redundancy as is for the sake of readability.

RFR-02: Unfair Proportionate Calculation

Type	Severity	Location
Mathematical Operations	Informational	ReferralFeeReceiver.sol L185-L195

Description:

The linked code block is meant to collect an epoch's share by calculating the percentage of shares a user has proportionate to the total supply of shares of a given epoch and multiply the resulting percentage with the total inch balance of the epoch.

Recommendation:

The calculation carried out is unfair because the share is subtracted from the total supply after it has been claimed, resulting in disproportionate percentages due to rounding down of the calculation on L190. For a minimal example:

State A:

- Share Supply: 100
- 1inch Balance: 50
- User A: 20 shares equivalent to `20 * 50 / 100` -> `10`
- User B: 5 shares equivalent to `5 * 50 / 100` -> `2.5` -> `2`

User B claims

State B:

- Share Supply: 95
- 1inch Balance: 48
- User A: 20 shares equivalent to `20 * 48 / 95` -> `10.105~`

The differences should be negligible in most cases as they too are rounded down, however for larger amounts or after compounding they may become significant. We advise that the impact of this is evaluated and potentially ignored if identified to be completely negligible in the context of Mooniswap.

Alleviation:

After discussing with the 1inch team, we concluded taht this issue is negligible as the `1inch` token has `18` decimal places.

RFR-03: Prohibition of Ether Transfers

Type	Severity	Location
Logical Issue	Informational	ReferralFeeReceiver.sol L10

Description:

The contract does not permit Ethereum transfers from contracts in contrast to `GovernanceFeeReceiver`.

Recommendation:

We advise that support for Ether transfers from contracts is added as is the case with `GovernanceFeeReceiver.sol`.

Alleviation:

The codebase was revised to instead relocate the prevention of Ether transfers from `GovernanceFeeReceiver` to `Converter` thus avoiding code duplication and ensuring consistency.

RFR-04: Inexistent Input Sanitization of Mooniswap Addresses

Type	Severity	Location
Logical Issue	Medium	ReferralFeeReceiver.sol L38, L52, L66, L113, L129, L141, L166, L174, L200

Description:

The linked code blocks link to function implementations within `ReferralFeeReceiver` that do not conduct any sanitization on the input `mooniswap` variable and invoke functions on it.

Recommendation:

We advise that a sanitization check is imposed whereby the `address` is verified to be existent within the Mooniswap Factory mapping via its `isPool` exposed function.

Alleviation:

The 1inch team implemented a `validPool` modifier within `Converter` to ensure that the pools are properly validated to have been deployed by the official Mooniswap factory. Additionally, the team added reentrancy guards to ensure that even in the event of valid pools malicious tokens do not attempt to re-enter the contract and attempt to exploit it.

CON-01: Whitelist Evaluation

Type	Severity	Location
Control Flow	Minor	Converter.sol L100

Description:

The linked `for` loop ensures that the elements in the `path` are whitelisted, however the first element is assumed to be whitelisted whilst it is not guaranteed so in the `require` checks of the function.

Recommendation:

We advise that the `for` loop also evaluates the first element in the array, as functions like `swap` in `GovernanceFeeReceiver` do not guarantee the first element.

Alleviation:

The 1inch team stated that the first element of the `path` is actually meant to be an arbitrary token not necessarily whitelisted, thus nullifying the validity of this exhibit.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.