SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Messier

Audit

Security Assessment
05. July, 2022

For

M87

MESSIER

SolidProof_io

@solidproof_io

Disclaimer

SolidProof.io reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc’...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug- free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof’s position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	05. July 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network

Ethereum (ERC20)

Website

https://blackhole.messier.app/

Telegram

https://t.me/MessierM87Community

Twitter

https://twitter.com/MessierM87

Github

https://github.com/MessierM87

Medium

https://medium.com/@MessierM87

Description

TBA

Project Engagement

During the 3rd of July 2022, **Messier Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Link

v1.0

- MessierAnonymity
  - https://etherscan.io/address/0x1963737af12D3649F6f319fB64af2aA29f7256B7#code
  - https://etherscan.io/address/0xb3b3206203109Ff1dFF8118386Ea90aC834d6C77#code
  - https://etherscan.io/address/0xDFE8FB8faA4b4bb9F8d0A8135A1706104b3681c6#code
  - https://etherscan.io/address/0xeB8878C6455f2c979E11FaB5DC92768aFC13C462#code
- MessierWrapper
  - https://etherscan.io/address/0x615A2Ba5e4494a2958CD377A97490BBb66d70919#code
  - https://etherscan.io/address/0xb2b01b38caa4004b17d880cbc6bd8a092ea17408#code
  - https://etherscan.io/address/0xeb3C2460143c0D1101318FAa81546e4b3FA17122#code
  - https://etherscan.io/address/0x85C2EeBaB92Ef47289F9f05EE29b273e5fA122C0#code

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 aspossible.
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

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.

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 SolidProof 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 SolidProof describe.

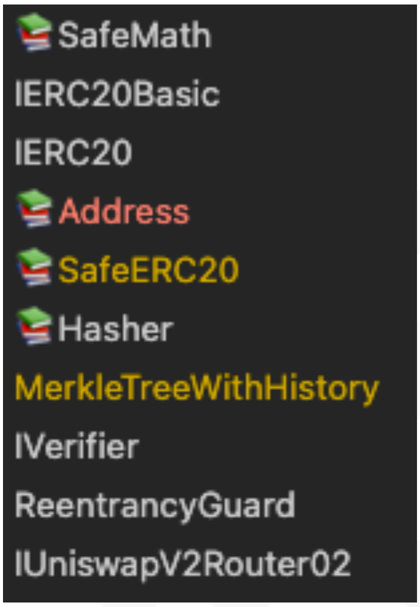
2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analysing a program 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.

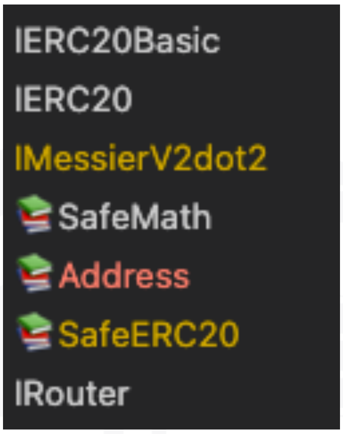
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

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

Imported packages:



SafeMath
IERC20Basic
IERC20
Address
SafeERC20
Hasher
MerkleTreeWithHistory
IVerifier
ReentrancyGuard
IUniswapV2Router02



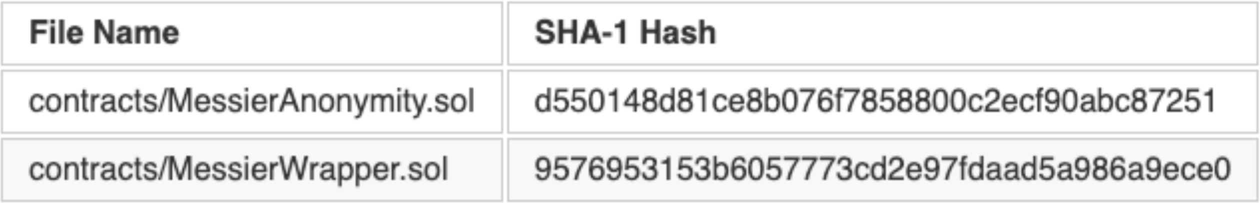
IERC20Basic
IERC20
IMessierV2dot2
SafeMath
Address
SafeERC20
IRouter

Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

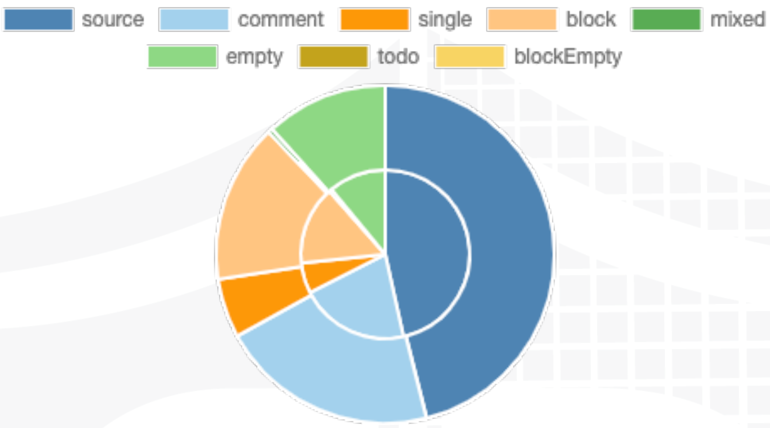


File Name	SHA-1 Hash
contracts/MessierAnonymity.sol	d550148d81ce8b076f7858800c2ecf90abc87251
contracts/MessierWrapper.sol	9576953153b6057773cd2e97fdaad5a986a9ece0

Metrics

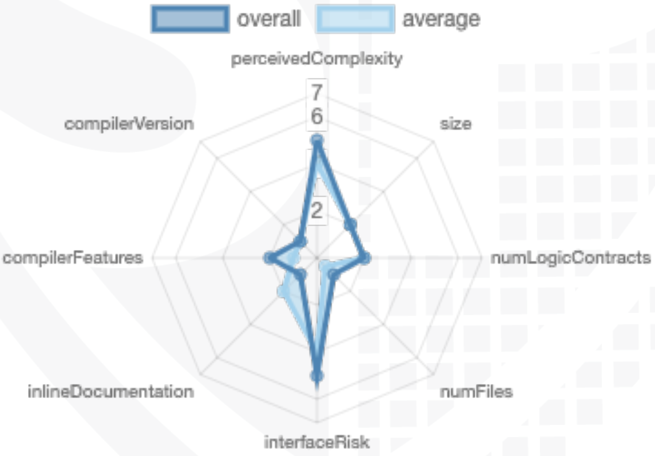
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	9	7	3	0

Exposed Functions

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

Version	Public	Payable
1.0	59	8

Version	External	Internal	Private	Pure	View
1.0	27	66	4	12	25

State Variables

Version	Total	Public
1.0	40	36

Capabilities

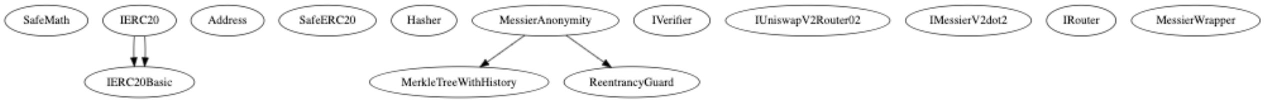
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts

1.0	`<0.6 >=0.4.21`
	`^0.5.0`
	`<0.6 >=0.4.24`
	`>=0.5.0 <0.8.0`
			`yes`	`yes`
				(2 asm blocks)	

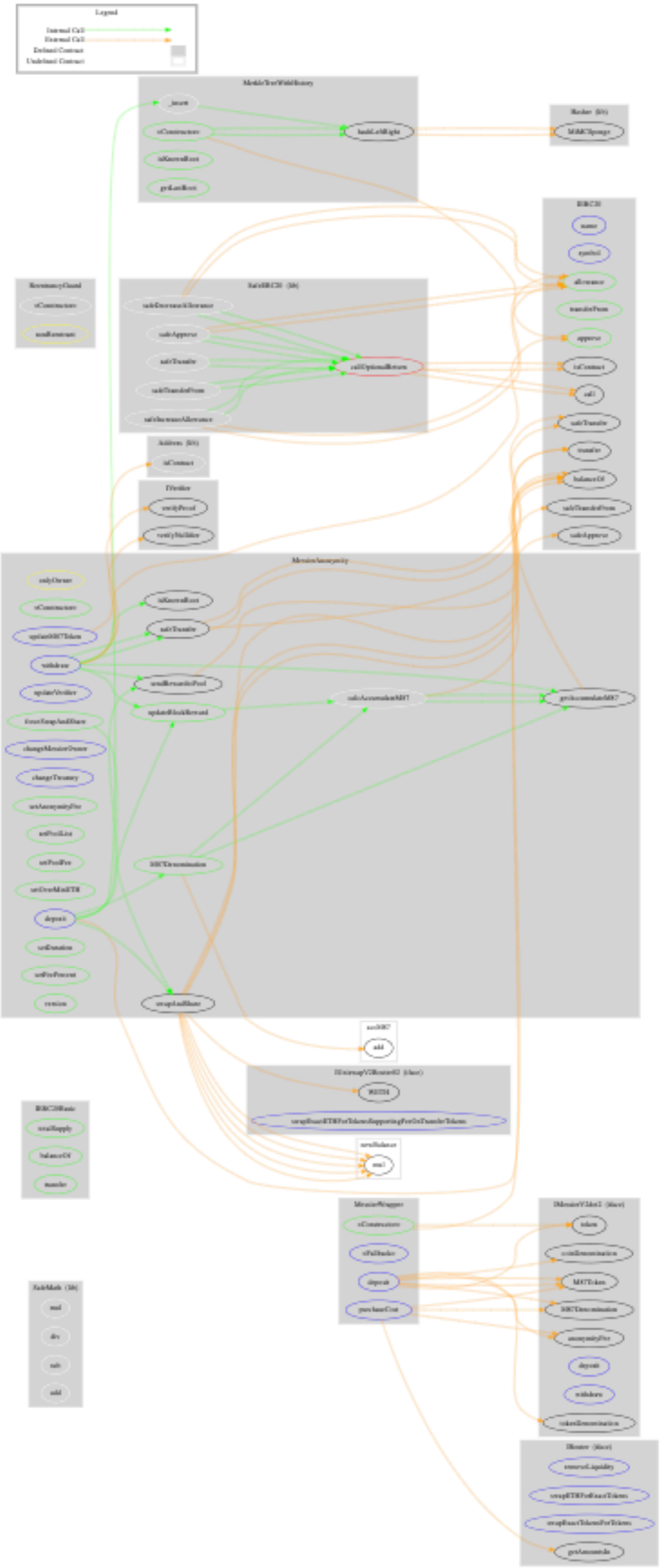
Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
1.0	`yes`	`yes`				

Inheritance Graph

v1.0



CallGraph
v1.0



Scope of Work/Verify Claims

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

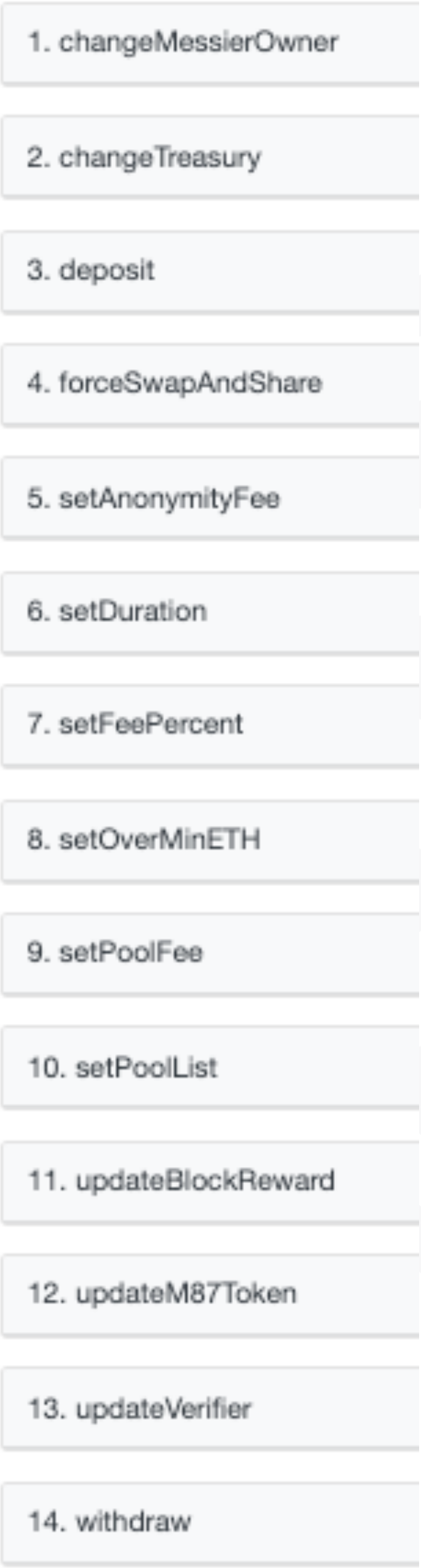
We will verify the following claims:

1. Is contract an upgradeable
2. Overall checkup (Smart Contract Security)

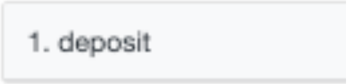
Is contract an upgradeable

Name	
Is contract an upgradeable?	No

Write functions of contract v1.0



1. changeMessierOwner
2. changeTreasury
3. deposit
4. forceSwapAndShare
5. setAnonymityFee
6. setDuration
7. setFeePercent
8. setOverMinETH
9. setPoolFee
10. setPoolList
11. updateBlockReward
12. updateM87Token
13. updateVerifier
14. withdraw



1. deposit

Overall checkup (Smart Contract Security)

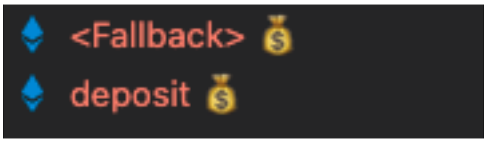
Tested	Verified
✓	✓

Legend

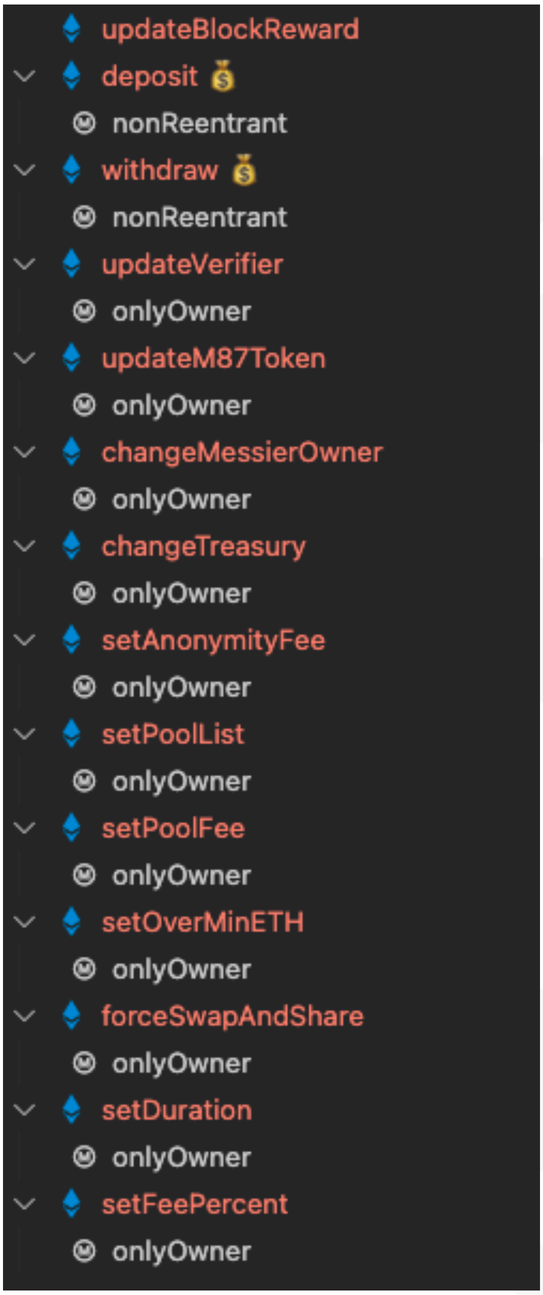
Attribute	Symbol
Verfified / Checked	✓
Partly Verified	⚑
Unverified / Not checked	✗
Not available	–

Modifiers and public functions

v1.0



```
<Fallback> 💰
deposit 💰
```



```
updateBlockReward
deposit 💰
    nonReentrant
withdraw 💰
    nonReentrant
updateVerifier
    onlyOwner
updateM87Token
    onlyOwner
changeMessierOwner
    onlyOwner
changeTreasury
    onlyOwner
setAnonymityFee
    onlyOwner
setPoolList
    onlyOwner
setPoolFee
    onlyOwner
setOverMinETH
    onlyOwner
forceSwapAndShare
    onlyOwner
setDuration
    onlyOwner
setFeePercent
    onlyOwner
```

Comments

- *Deployer can set following state variables without any limitations*
    - duration
    - numDurationBlocks
    - Duration/numDurationBlocks has no functionality in the contract
    - anonymityFee

- *Deployer can set following addresses*
    - poolList
    - treasury
    - messier_owner
    - M87Token
    - verifier

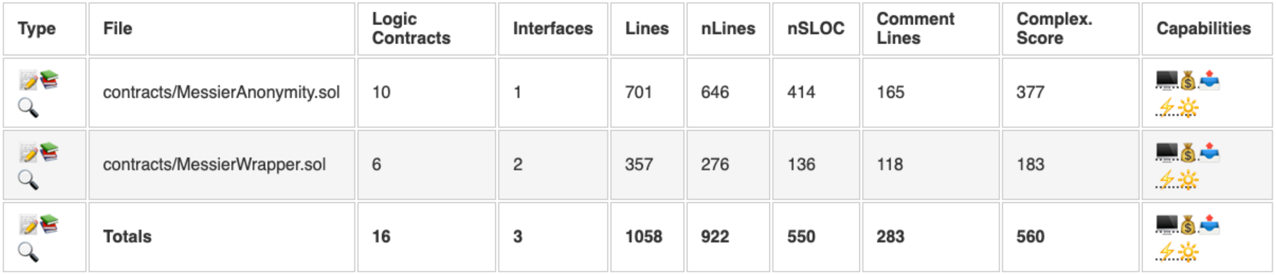
- *Existing Modifiers*
    - onlyOwner
    - nonReentrant

- UpdateBlockReward can be called anytime from anyone
    - It will set the lastRwardBlock to the current block.number
- sharedOfRewards has to be at least 100
- Fee is not used in deposit function but it is set that it can be returned by the function
- Be aware of logic while changing messier owner
    - While the m87Token address will changed, the owner will be approved with the max value but if the owner changed, the old owner is still able to transfer because her was approved before. Our recommendation is to approve old owner to 0 and approve new owner to max while changing owner

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope

v1.0



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚 🔍	contracts/MessierAnonymity.sol	10	1	701	646	414	165	377	🖥💰📥 ⚡🔆
📝📚 🔍	contracts/MessierWrapper.sol	6	2	357	276	136	118	183	🖥💰📥 ⚡🔆
📝📚 🔍	**Totals**	**16**	**3**	**1058**	**922**	**550**	**283**	**560**	🖥💰📥 ⚡🔆

Legend

Attribute	Description
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, ...)

Audit Results

AUDIT PASSED

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

Issue	File	Type	Line	Description
#1	All	Contract doesn't import npm packages from source (like OpenZeppelin etc.)	-	We recommend to import all packages from npm directly without flatten the contract. Functions could be modified or can be susceptible to vulnerabilities
#2	All	A floating pragma is set	At the top of files	Use a specific pragma version instead of floating
#3	Messier Anony mity	Missing Zero Address Validation (missing-zero-check)	589 593	Check that the address is not zero
#4	Messier Wrappe r	Missing Zero Address Validation (missing-zero-check)	298	Check that the address is not zero
#5	Messier Anony mity	Missing Events Arithmetic	589 688 695 622	Emit an event for critical parameter changes

#6	Messier Anonymity	Uninitialised variable	418	The variable “rewardPerBlock” variable is not initialized. That means, that the variable will be 0 forever. This variable is used in the following functions: - calcAccumulateM87 L460 - updateBlockReward L469 - M87Denomination L488
#7	All	Raw mathematical operations	Search for raw mathematical operations and replace it with SafeMath operations	Don’t use raw mathematical operations because of under-/overflow. We recommend you to use SafeMath operations in every cases.

Informational issues

Issue	File	Type	Line	Description
#1	Messier Anonymity	State variables that could be declared constant (constable-states)	418	Add the `constant` attributes to state variables that never change
#2	Messier Wrapper	Unused state variables	294	Remove unused state variables.
#3	All	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.
#4	All	Useless state variable and condition	See description	tokenDenomination L407 will be set to 0 in L452 and will never be changed again. L515-L518 will never be called because of that this state variable will never become higher than 0 Same in MessierWrapper L327 and L330-L332

#5	Messier Wrapper	Set type directly	350	You can set the type of variable directly
#6	Messier Anonymity	State variable in loop	671	We recommend you to set the state variable outside the for loop and set it back to the state after iteration because of gas or use instead the “I” variable in the loop

Commented Code exist

There are some instances of code being commented out in the following files that should be removed:

File	Line	Comment

MessierAnonymity	48-50	// assert(b > 0); // Solidity automatically throws when dividing by 0
		// uint256 c = a / b;
		// assert(a == b * c + a % b); // There is no case in which this doesn't hold

	652	// M87Token.transfer(poolList[0], newBalance.mul(shareOfReward[1]).div(100));
	657	// M87Token.transfer(poolList[1], newBalance.mul(shareOfReward[2]).div(100));
	662	// M87Token.transfer(poolList[2], newBalance.mul(shareOfReward[3]).div(100));
	667	// M87Token.transfer(poolList[3], newBalance.mul(shareOfReward[4]).div(100));

MessierWrapper	103-105	// assert(b > 0); // Solidity automatically throws when dividing by 0
		// uint256 c = a / b;
		// assert(a == b * c + a % b); // There is no case in which this doesn't hold

Recommendation

Remove the commented code, or address them properly.

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/v0.5.10/natspec-format.html) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

05. July 2022:

- Verify contract was not provided to solidproof. Please do your own research here in this case
- Read whole report and modifiers section for more information

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	NOT PASSED
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	**PASSED**
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	**PASSED**
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	**PASSED**
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	**PASSED**
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	**PASSED**
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	**PASSED**
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	**PASSED**
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	**PASSED**
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	**PASSED**
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	**PASSED**

SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	NOT PASSED
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

SolidProof_io

@solidproof_io

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY