



SOLIDProof

Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY

Auragi Audit

**Security Assessment
01. April, 2023**

For



SolidProof_io



@solidproof_io

Disclaimer	3
Description	5
Project Engagement	5
Logo	5
Contract Link	5
Methodology	7
Used Code from other Frameworks/Smart Contracts (direct imports)	8
Tested Contract Files	9
Source Lines	10
Risk Level	10
Capabilities	11
Inheritance Graph	12
CallGraph	13
Scope of Work/Verify Claims	14
Modifiers and public functions	17
Source Units in Scope	18
Critical issues	19
High issues	19
Medium issues	19
Low issues	19
Informational issues	19
Audit Comments	19
SWC Attacks	21

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	01. April 2023	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary

Network

Arbitrum

Website

<https://auragi.finance/>

Twitter

<https://twitter.com/AuragiFinance>

Discord

<https://discord.gg/xGuKne5Mye>



Description

Take advantage of Auragi's minimal slippage, low swapping fees, and deep liquidity

Auragi - Swap tokens at the best rates. Take advantage of minimal slippage, low swapping fees, and deep liquidity with Auragi.

Project Engagement

During the 30th of March 2023, **Auragi 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

- **Minter:** <https://goerli.arbiscan.io/address/0x5C83C2E499552f9f8C17F5508f5aA2d2cEcc5C5a#code>
- **Auragi:** <https://goerli.arbiscan.io/address/0xad99c18f6AD245b87E1692A9408514BEFC3bE557#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 as possible.
Medium	4 - 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 - 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 - 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

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:

Dependency / Import Path	Count
contracts/interfaces/IAuragi.sol	1
contracts/interfaces/IMinter.sol	1
contracts/interfaces/IRewardsDistributor.sol	1
contracts/interfaces/IVoter.sol	1
contracts/interfaces/IVotingEscrow.sol	1
contracts/libraries/Math.sol	1

Tested Contract Files

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

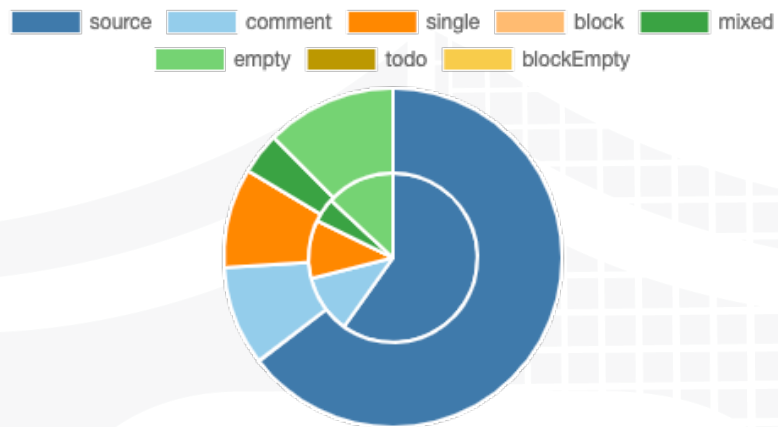
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

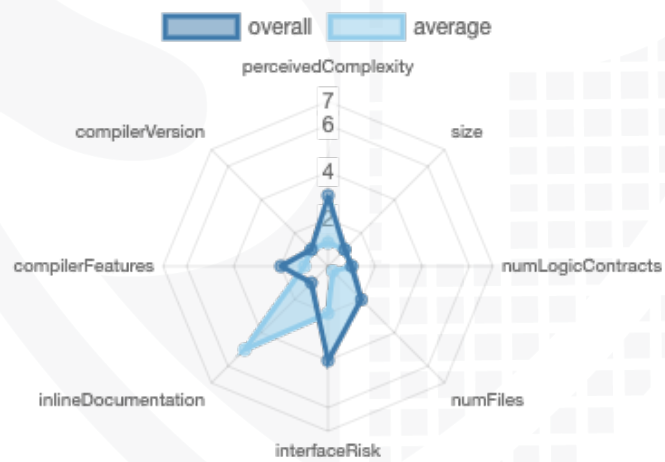
File Name	SHA-1 Hash
contracts/Math.sol	cda673e64f318665e13f86001aae0c3cd9deb679
contracts/Minter.sol	0e1dd0b8160fde7534e7ceb6e34d08973a5c21b6
contracts/IMinter.sol	d1a83d8e97b88d3fc5f59f5c00d814b58ac5d523
contracts/IVoter.sol	98e97b1db23c9ee6458be78883f6cfe6e0f77c52
contracts/IAuragi.sol	a1a039dd7daa78e2a1b31528c6c3c0703d05575d
contracts/ IRewardsDistributor.sol	152ab5a4265b8e72c3f312a1a2ed83f6389c542f
contracts/Auragi.sol	0a03fa6a9cd42559a33ad66355d8cce24940fa33
contracts/ IVotingEscrow.sol	174a4bbc548be7bb662a3f7cc83fc5730846167a

Metrics

Source Lines v1.0



Risk Level v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	1	1	5	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	47	0

Version	External	Internal	Private	Pure	View
1.0	42	32	0	4	20

State Variables

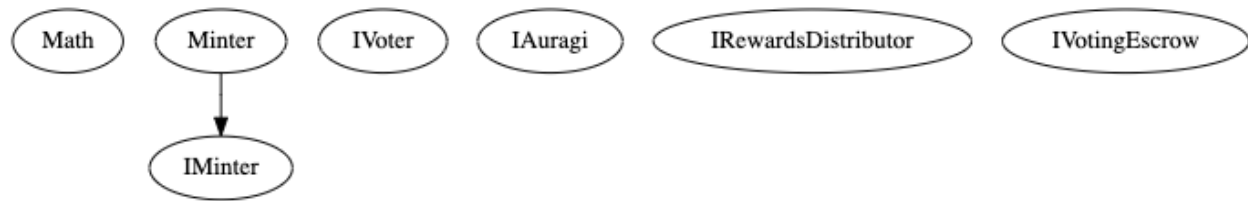
Version	Total	Public
1.0	16	9

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	0.8.13				

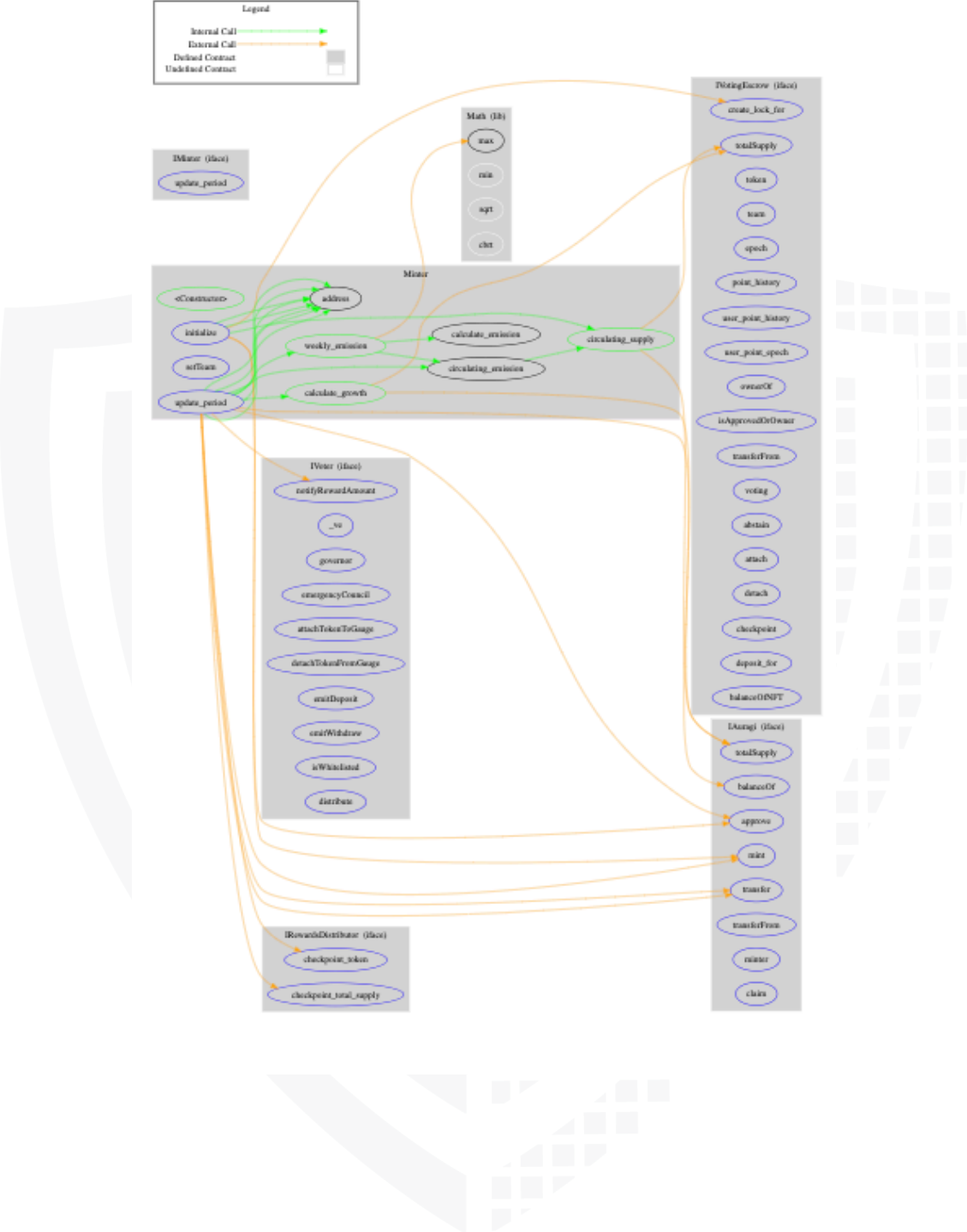
Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	EC Recover	New/Create/Create2
1.0	yes					

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



Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

Attribute	Symbol
Verified / Checked	✓
Partly Verified	⚠
Unverified / Not checked	✗
Not available	—

Modifiers and public functions v1.0

initialize	setMinter
setTeam	setMerkleClaim
update_period	initialMint
	approve
	transfer
	transferFrom
	mint
	claim

Note:

- Following contracts were not provided to solidproof for the audit. Please do your own research
 - Voter
 - VotingEscrow
 - RewardDistributor

Ownership Privileges

- Governor address set by the deployer at the time of deployment can set the following addresses
 - Team
- The Address/Contract with the minter role can mint unlimited tokens and also set the merkleAddress
 - Beware about this
- Merkle Claim address can only claim a maximum of 132M tokens

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

Source Units in Scope

v1.0

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/Math.sol	1	=====	36	36	33	1	12
contracts/Minter.sol	1	=====	135	132	101	21	84
contracts/IMinter.sol	=====	1	7	6	3	1	3
contracts/IVoter.sol	=====	1	16	6	3	1	21
contracts/IAuragi.sol	=====	1	14	6	3	1	17
contracts/IRewardsDistributor.sol	=====	1	8	6	3	1	5
contracts/Auragi.sol	1	=====	107	107	79	10	50
contracts/IVotingEscrow.sol	=====	1	36	14	9	3	37
Totals	3	5	359	313	234	39	229

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalised lines of the source unit (e.g. normalises functions spanning multiple lines)
nSLOC	normalised 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

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	Minter.sol	Missing Length Check	54	Make sure that the length of both arrays are equal before iteration.
#3	Minter.sol	Missing Zero Address Validation (missing-zero-check)	70	Check that the address is not zero
#4	Auragi.sol	Missing Zero Address Validation (missing-zero-check)	32, 38	Check that the address is not zero

Informational issues

Issue	File	Type	Line	Description
#1	All	NatSpec documentation missing	—	If you started to comment your code, also comment all other functions, variables etc.

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information <https://docs.soliditylang.org/en/>)

[latest/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.

01. April 2023:

- There is still an owner (Owner still has not renounced ownership)
- Owner can deploy a new version of the contract which can change any limit and give owner new privileges
- The minter address will be able to mint unlimited tokens until it is set to a zero address.
- Read whole report and modifiers section for more information



SWC Attacks

ID	Title	Relationships	Status
SW C-1 36	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SW C-1 35	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SW C-1 34	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SW C-1 33	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SW C-1 32	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SW C-1 31	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SW C-1 30	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SW C-1 29	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SW C-1 28	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SW C-1 27	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SW C-1 25	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SW C-1 24	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SW C-1 23	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SW C-1 22	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SW C-1 21	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SW C-1 20	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SW C-11 9	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-11 8	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SW C-11 7	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SW C-11 6	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 5	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SW C-11 4	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SW C-11 3	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SW C-11 2	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 1	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SW C-11 0	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SW C-1 09	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-1 08	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-1 07	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-1 06	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SW C-1 05	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SW C-1 04	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SW C-1 03	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	PASSED
SW C-1 02	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SW C-1 01	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SW C-1 00	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

*Solid
Proofed*

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**


MADE IN GERMANY