



SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

MEGASET

Audit

Security Assessment

26. April, 2022

For



MEGASET

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	20
Source Units in Scope	22
Critical issues	23
High issues	23
Medium issues	23
Low issues	23
Informational issues	24
Audit Comments	24
SWC Attacks	25

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	22. April 2022	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary
1.1	26. April 2022	<ul style="list-style-type: none">• Reaudit

Network

Binance Smart Chain (BEP20)

Website

<https://megaset.io/>

Telegram

<https://t.me/MegasetOfficial>

Twitter

<https://twitter.com/MegasetOfficial>

Reddit

<https://www.reddit.com/r/MegasetOfficial/>

Medium

<https://megaset.medium.com/>

Description

MEGASET is the next generation of the asset world. As one of its earliest explorers, you will help its expansion and share in the benefits of this growth.

Project Engagement

During the 20th of April 2022, **MEGASET 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

- Github
 - <https://github.com/megasetofficial/MEGASET>
 - Commit: 5ddcb93615a4e61cbfea83dc795065105c453ad1

v1.1

- Github
 - <https://github.com/megasetofficial/MEGASET>
 - Commit: 2214f177e9d91b2209e071010e8e3fbd74c3ea1a

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.

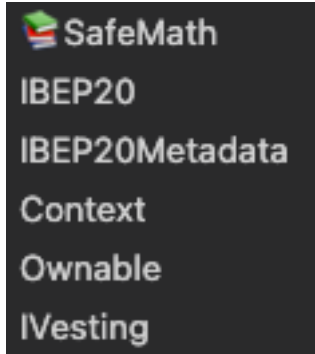
Methodology

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:



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

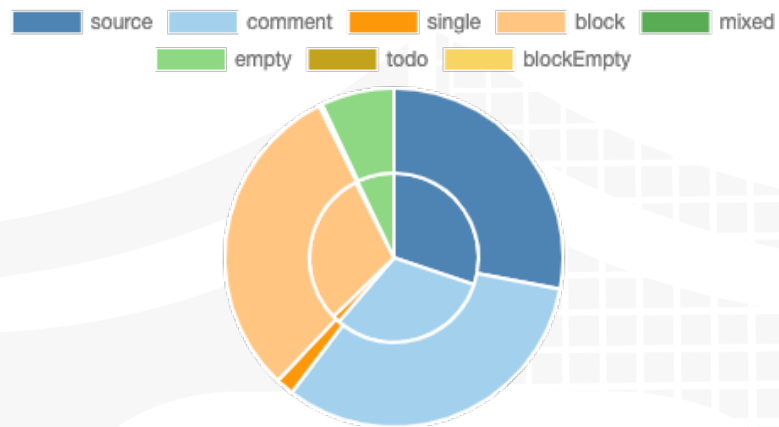
File Name	SHA-1 Hash
contracts/megaset.sol	20d619da842a30eb7f0080c3ceaf8c207617f9e2
contracts/presale.sol	e165a2d8b6528c0eb5d31d4e94d3ed775b3faf08

v1.1

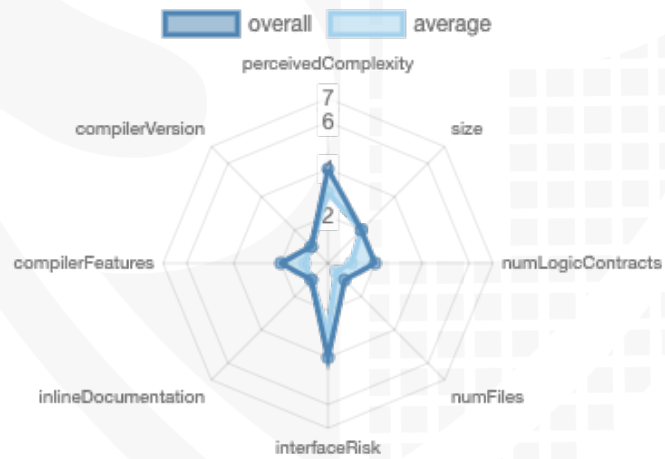
File Name	SHA-1 Hash
contracts/megaset.sol	cadd732b283ad4a879d7b9fa32f094d006f3a0c0
contracts/presale.sol	d85e1ab03fe6e50ea72e0d0de0fbcf996bb634b6

Metrics

Source Lines v1.0



Risk Level v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	2	5	4

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	49	1

Version	External	Internal	Private	Pure	View
1.0	31	90	5	26	23

State Variables

Version	Total	Public
1.0	35	1

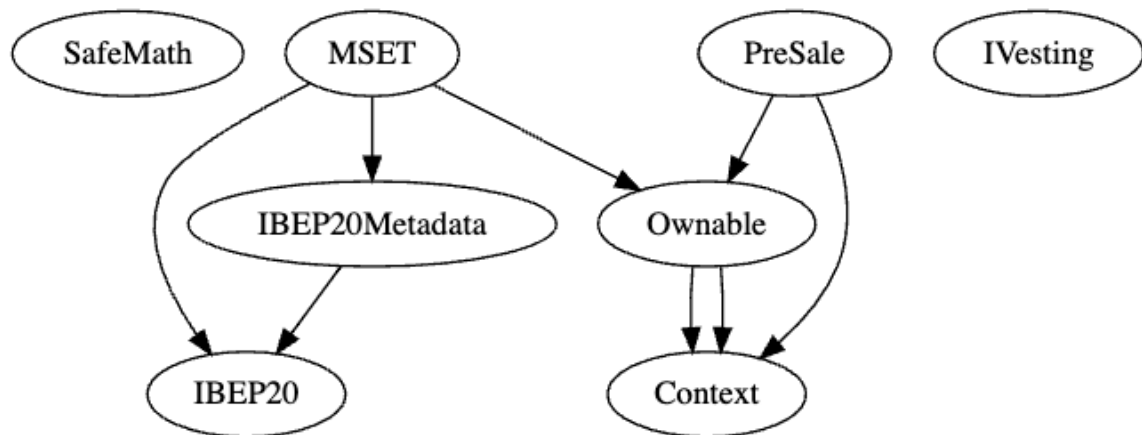
Capabilities

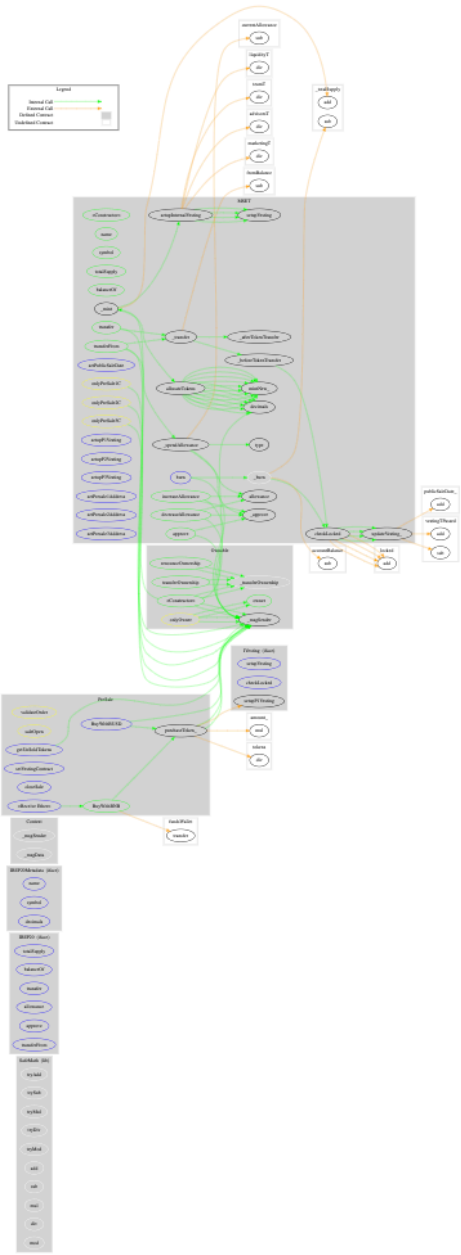
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	<code>^0.8.0</code> <code>^0.8.9</code>		<code>yes</code>		

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

1.0	yes					
-----	-----	--	--	--	--	--

Inheritance Graph 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. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall checkup (Smart Contract Security)

Correct implementation of Token standard

ERC20				
Function	Description	Exist	Tested	Verified
TotalSupply	Provides information about the total token supply	✓	✓	✓
BalanceOf	Provides account balance of the owner's account	✓	✓	✓
Transfer	Executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	Executes transfers of a specified number of tokens from a specified address	✓	✓	✓
Approve	Allow a spender to withdraw a set number of tokens from a specified account	✓	✓	✓
Allowance	Returns a set number of tokens from a spender to the owner	✓	✓	✓

Write functions of contract v1.0

```
BuyWithBUSD  
BuyWithBNB  
setVestingContract  
closeSale  
getUnSoldTokens  
  
renounceOwnership  
transferOwnership
```

```
transfer  
approve  
transferFrom  
increaseAllowance  
decreaseAllowance  
burn  
setPublicSaleDate  
setupP1Vesting  
setupP2Vesting  
setupP3Vesting  
setPresale1Address  
setPresale2Address  
setPresale3Address
```

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓



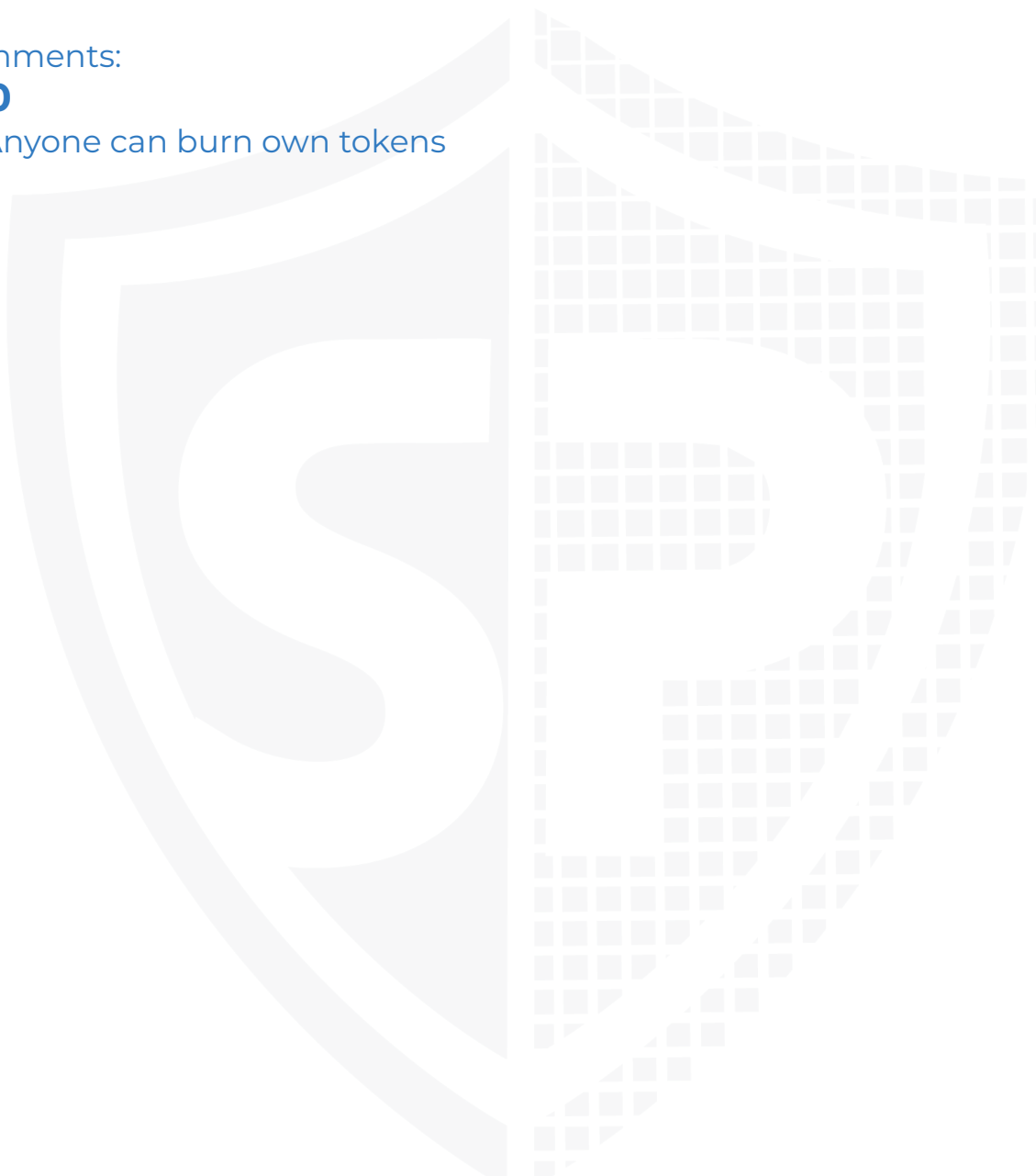
Deployer cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer cannot lock	✓	✓	✓
Deployer cannot burn	✓	✓	✓

Comments:

v1.0

- Anyone can burn own tokens



Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	—	—	—



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

```
transfer
approve
transferFrom
increaseAllowance
decreaseAllowance
burn
setPublicSaleDate
  @ onlyOwner
setupP1Vesting
  @ onlyPreSale1C
setupP2Vesting
  @ onlyPreSale2C
setupP3Vesting
  @ onlyPreSale3C
setPresale1Address
  @ onlyOwner
setPresale2Address
  @ onlyOwner
setPresale3Address
  @ onlyOwner
```

```
BuyWithBUSD
  @ validateOrder
BuyWithBNB
  @ validateOrder
setVestingContract
  @ onlyOwner
closeSale
  @ onlyOwner
getUnSoldTokens
  @ onlyOwner
```

Comments







- Deployer can set following state variables without any limitations
 - presale3Locking[account_].cliff
 - presale3Locking[account_].vestingTime
 - presale3Locking[account_].vestingAmt
 - presale3Locking[account_].vestingAmtLeft
 - presale2Locking[account_].cliff
 - presale2Locking[account_].vestingTime
 - presale2Locking[account_].vestingAmt
 - presale2Locking[account_].vestingAmtLeft
 - presale1Locking[account_].cliff
 - presale1Locking[account_].vestingTime
 - presale1Locking[account_].vestingAmt
 - presale1Locking[account_].vestingAmtLeft

- publicSaleDate
- Deployer can set following addresses
 - vestingC
 - presale3CAddress
 - presale2CAddress
 - presale1CAddress
- Owner can close sale
- Vesting contract was not provided to solidproof. Please do your own research here
- BuyWithBNB function does not transfer amount to owner, it transfers amount to fund wallet, same BuyWithBUSD function

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/megaset.sol	4	3	959	831	365	452	261	
	contracts/presale.sol	4	2	464	397	179	236	123	
	Totals	8	5	1423	1228	544	688	384	

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	Main	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	Main	Missing Zero Address Validation (missing-zero-check)	948, 952, 956	Check that the address is not zero
#3	Presale	Missing Zero Address Validation (missing-zero-check)	425	Check that the address is not zero
#4	Main	Local variables shadowing	702, 722, 520, 535, 600, 580, 512	Rename the local variables that shadow another component
#5	Main	Missing Events Arithmetic	949, 953, 957, 832	Emit an event for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	Main	State variables that could be declared constant (constable-states)	790, 787, 785, 793, 781, 782, 783, 784, 792, 788	Add the `constant` attributes to state variables that never change
#2	Presale	State variables that could be declared constant (constable-states)	400, 399, 398	Add the `constant` attributes to state variables that never change
#3	Main	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.
#4	Main	Incorrect time	814-819	<p>7889400 = 91 days, 7 hours, 30 minutes and 0 seconds.</p> <p>You should change it to 777600 if you want to have 3 months exactly</p> <p>Same for the others 12 months = 365 days = 31536000</p>

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.

22. April 2022:

- Read whole report 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	NOT 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

The logo features the words "SolidProof" in a white, handwritten-style script. The text is superimposed on a blue background that includes a faint, stylized shield emblem with a grid pattern.

SolidProof

Blockchain Security | Smart Contract Audits | KYC



MADE IN GERMANY