



SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment
10. December, 2021

For

METAKINGS

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
Scope of Work	13
Inheritance Graph	13
Verify Claims	14
Modifiers	20
CallGraph	21
Source Units in Scope	22
Critical issues	23
High issues	23
Medium issues	23
Low issues	23
Informational issues	24
Commented Code exist	24
Audit Comments	25
SWC Attacks	26

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	10. December 2021	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary

Network

Binance Smart Chain (BEP20)

Website

<https://metakings.ai/>

Telegram

<https://t.me/metakings>

<https://t.me/metakingsann>

Twitter

<https://twitter.com/metakingsai>

Facebook

Github

<https://github.com/metakings>

Youtube

<https://www.youtube.com/channel/UCe70dNypBRBFkMBZbg47hhQ>

Medium

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

Description

TBA

Project Engagement

During the 6th of December 2021, **Metakings 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

- <https://bscscan.com/token/0x129385c4acd0075e45a0c9a5177bdfec9678a138>

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:

```
./ERC20Token.sol  
./ISellToken.sol  
./Initializable.sol
```



Tested Contract Files

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

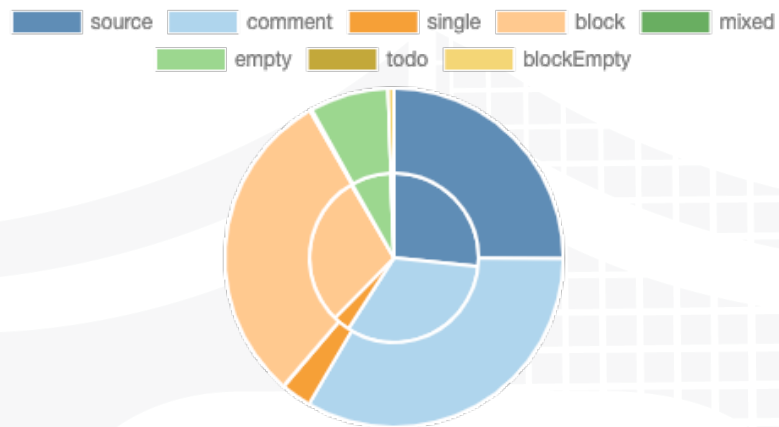
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

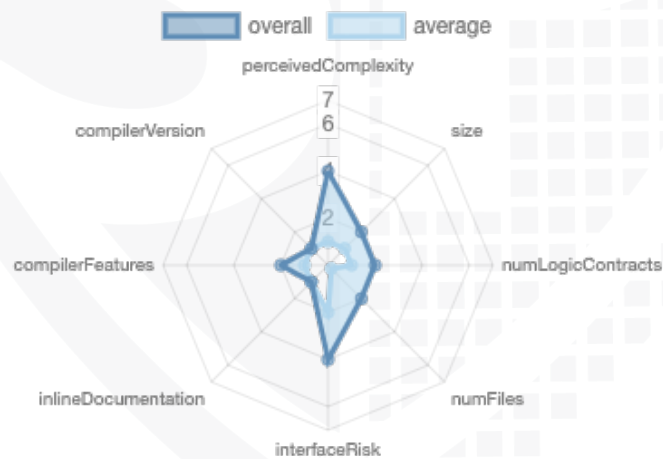
File Name	SHA-1 Hash
contracts/ILockContract.sol	d5ca4114adaebd519477e9439f0778d7d2baa312
contracts/Context.sol	76285d882381efb438fd4474f432ad35ae1ee067
contracts/MetaKings.sol	2fe768cb4f5ab034007b5677ed2133072c2a6272
contracts/Address.sol	67b17af334e94536fda0693869d9c19c7851a4cd
contracts/SafeMath.sol	c4cd6f06c9459a7dc67ca97ea7a52e1d10d1ba70
contracts/ISellToken.sol	e06b0bd5d81af5afa580d135acc04d42f1f8128d
contracts/ERC20Token.sol	1d35b9dce81cbd205084d43c9e267140ed11634a
contracts/Ownable.sol	cbfec717ef32140f645d777f6a3b63b615e59d5
contracts/IERC20.sol	50480800497ef0aa207300c85a22b15e69e59584
contracts/Initializable.sol	0bccb3391a17c7ba15a512cf2372ad87e5d914d4

Metrics

Source Lines v1.0



Risk Level v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	4	2	3	1

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	42	0

Version	External	Internal	Private	Pure	View
1.0	17	87	2	13	28

State Variables

Version	Total	Public
1.0	25	10

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	0.6.12 >=0.4.0 >=0.6.2 <0.8.0 >=0.4.2 4 <0.8.0			yes (3 asm blocks)	

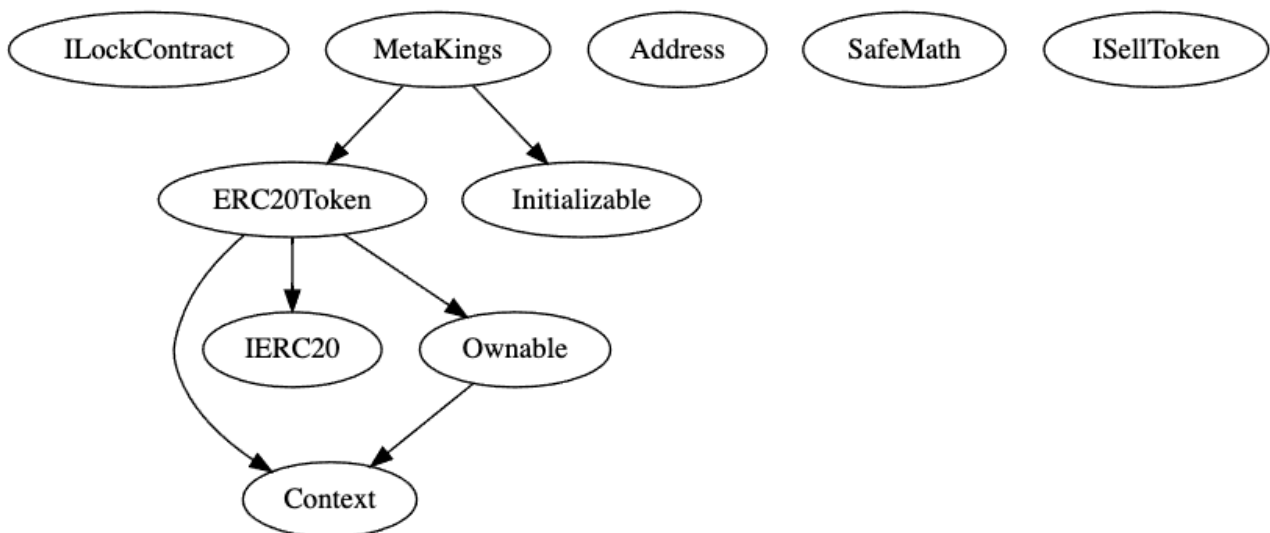
Scope of Work

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)

Inheritance Graph v1.0



Verify Claims

Correct implementation of Token standard

Tested	Verified
✓	✓

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

1. approve

2. bootswitch

3. burn

4. decreaseAllowance

5. deploy

6. dev

7. increaseAllowance

8. renounceOwnership

9. setLockContract

10. transfer

11. transferFrom

12. transferOwnership

13. withdrawBNB

14. withdrawFunc

Deployer cannot mint any new tokens

Name	Exist	Tested	Verified
Deployer cannot mint	✓	✓	🚩

Max / Total Supply: 1.000.000.000

Comments:

v1.0

- Owner can call deploy function to mint tokens only once



Deployer cannot burn or lock user funds

Name	Exist	Tested	Verified
Deployer cannot lock	✓	✓	🚩
Deployer cannot burn	✓	✓	✗

Comments:

v1.0

- ILockContract address was not provided to Solidproof. It is possible that lockedAmount function in line 298, ERC20Token.sol file can lock user funds. DYOR

Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	—	—	—



Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

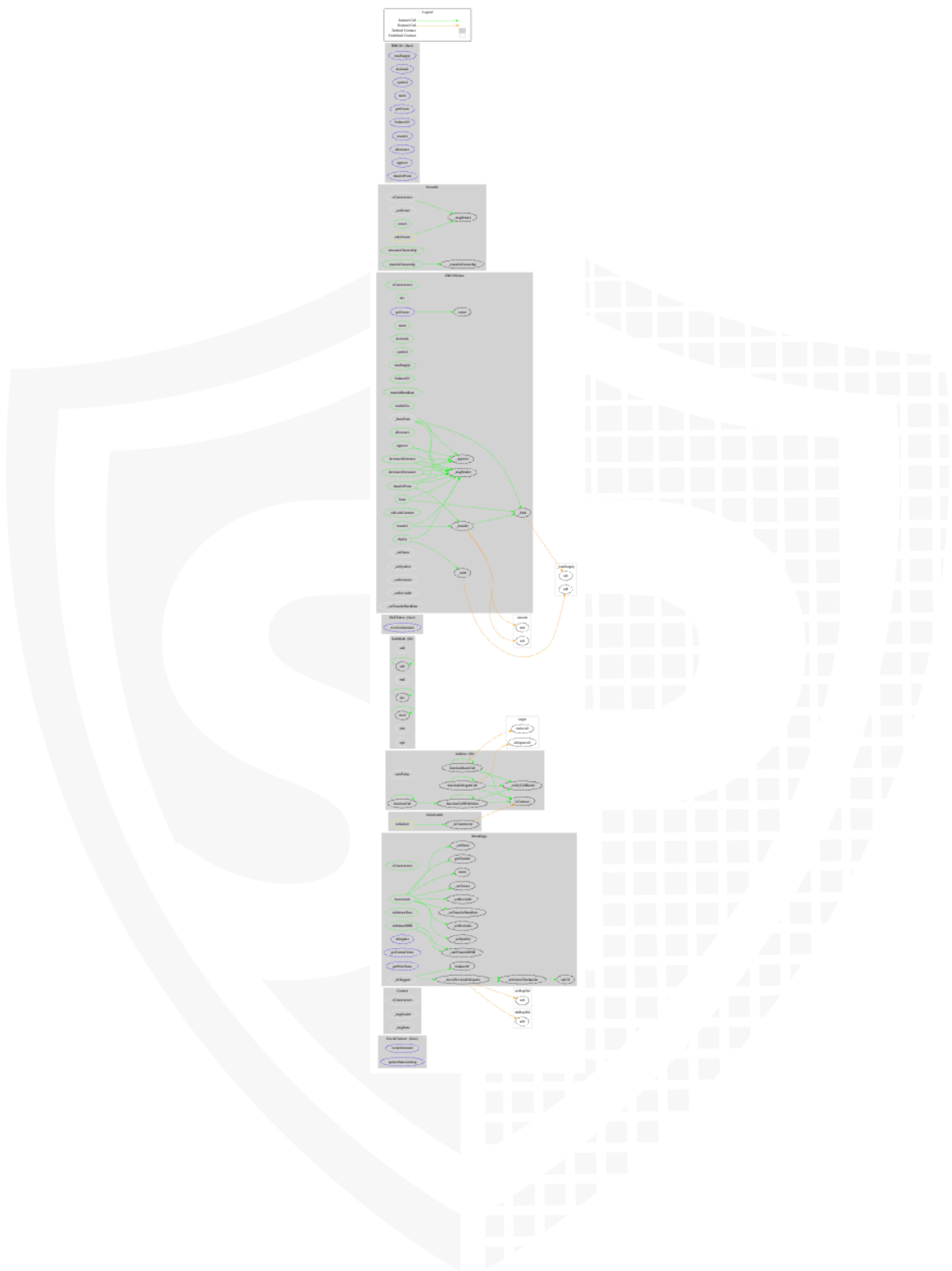
Attribute	Symbol
Verified / Checked	✓
Partly Verified	⚠
Unverified / Not checked	✗
Not available	—

Modifiers

- `Initializer`
 - `bootswitch`
- `onlyOwner`
 - `withdrawFunc`
 - Withdraw tokens
 - `withdrawBNB`
 - Owner can transfer amount of bib to recipient
- `deploy`
 - Can mint new tokens




















CallGraph



Source Units in Scope

v1.0

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/ILockContract.sol	_____	1	24	22	3	18	5	_____
	contracts/Context.sol	1	_____	38	38	11	24	1	
	contracts/MetaKings.sol	1	_____	273	252	132	82	107	
	contracts/Address.sol	1	_____	199	179	78	123	47	
	contracts/SafeMath.sol	1	_____	199	187	54	117	14	_____
	contracts/ISellToken.sol	_____	1	24	22	3	18	3	
	contracts/ERC20Token.sol	1	_____	405	393	165	190	128	_____
	contracts/Ownable.sol	1	_____	90	90	34	46	26	_____
	contracts/ERC20.sol	_____	1	108	33	17	76	21	_____
	contracts/Initializable.sol	1	_____	65	65	21	34	9	
	Totals	7	3	1425	1281	518	728	361	

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 found -

High issues

- no high issues found -

Medium issues

Issue	File	Type	Line	Description
#1	ERC20Token	Tautology or contradiction	299	Fix the incorrect comparison by changing the value type or the comparison

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	Address, Context, ERC20Token, IERC20, Initializable, Ownable, SafeMath	A floating pragma is set	-	The current pragma Solidity directive is not fixed
#3	ERC20Token	Missing Zero Address Validation (missing-zero-check)	86, 259	Check that the address is not zero

Informational issues

Issue	File	Type	Line	Description
#1	ERC20Token	State variables that could be declared constant (constable-states)	59, 63	Add the `constant` attributes to state variables that never change
#2	ERC20Token	Functions that are not used	377	Remove unused functions
#3	MetaKings	Functions that are not used	204, 216, 236, 259	Remove unused functions
#4	ERC20Token	Unused state variables	59	Remove unused state variables
#5	ERC20Token	Wrong comment	7	> 100%Unmintable contract Owner can mint new tokens with deploy function only once but it is not unmintable. Deployer mint after deploying with deploy function.
#6	ERC20Token	Wrong comment	254	//this mints the initial tokens required for project startup, minting is called just once and is never possible to call again Thats wrong, deploy function is not called in constructor and can be called more than once but function is not minting tokens if _totalSupply is not <= 0
#7	ERC20Token	Unnecessary code	299-301	_balance cannot be less then 0 because _balances state variable returns uint256

Commented Code exist

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

Line	Comment
51	// mapping (address => UserInfo) todo lock

Recommendation

Remove the commented code, or address them properly.

Audit Comments

10. December 2021:

- Read whole report for more information



SWC Attacks

ID	Title	Relationships	Status
SW C-13 6	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SW C-13 5	Code With No Effects	CWE-1164: Irrelevant Code	NOT PASSED
SW C-13 4	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SW C-13 3	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SW C-13 2	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SW C-13 1	Presence of unused variables	CWE-1164: Irrelevant Code	NOT PASSED
SW C-13 0	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SW C-12 9	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SW C-12 8	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SW C-12 7	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SW C-12 5	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SW C-12 4	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SW C-12 3	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SW C-12 2	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SW C-12 1	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SW C-12 0	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-111	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-10 9	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-10 8	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-10 7	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-10 6	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SW C-10 5	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SW C-10 4	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SW C-10 3	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	NOT PASSED
SW C-10 2	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SW C-10 1	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SW C-10 0	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

The logo features the words "Solid Proofed" in a white, elegant script font. The word "Solid" is positioned above "Proofed". Behind the text is a faint, stylized shield emblem with a grid-like pattern, rendered in a darker shade of blue. The entire composition is set against a solid blue background.

Solid
Proofed

Blockchain Security | Smart Contract Audits | KYC

A horizontal bar representing the German flag, with black, red, and gold stripes.

MADE IN GERMANY