



**SOLIDProof**  
*Bring trust into your projects*

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

MADE IN GERMANY

# WhiteDoge Audit

**Security Assessment**  
**16. March, 2022**

**For**



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	21
Critical issues	22
High issues	22
Medium issues	22
Low issues	22
Informational issues	23
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	16. March 2022	<ul style="list-style-type: none"><li>• Layout project</li><li>• Automated- /Manual-Security Testing</li><li>• Summary</li></ul>

## **Network**

Binance Smart Chain (BEP20)

## **Website**

<https://whitedoge.pro/>



## Description

WhiteDoge Coin one of the fastest growing community in crypto memes. WhiteDoge is on a mission to bring crypto to the average person while also helping save dogs in need. WhiteDoge is built on binance smart chain with extremely fast 5 second block times and cheaper gas fees than ethereum. White Doge Coin has learned a few tricks and lessons from his meme father, Doge. A new crypto birthed by fans of the Doge Meme online community. White Doge seeks to impress his father by showing his

new improved transaction speeds & adorableness. He is Hyper-deflationary with static reflection that rewards holders, so more

USDT are being automatically added to your wallet each transaction

## Project Engagement

During the 14th of March 2022, **WhiteDoge 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/address/0xa44a2df775cdee8be37b8197214f85bc5e366d2f#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)
<b>Critical</b>	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.
<b>High</b>	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.
<b>Medium</b>	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.
<b>Low</b>	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.
<b>Informational</b>	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:

```
IERC20
IERC20Metadata
DividendPayingTokenInterface
DividendPayingTokenOptionalInterface
IUniswapV2Router01
IUniswapV2Router02
IUniswapV2Factory
IERC20Upgradeable
IERC20MetadataUpgradeable
📦 Clones
📦 SafeMath
📦 SafeMathInt
📦 SafeMathUint
📦 IterableMapping
Context
Ownable
Initializable
ContextUpgradeable
OwnableUpgradeable
ERC20
ERC20Upgradeable
DividendPayingToken
WhiteDogeDividendTracker
```



# Tested Contract Files

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

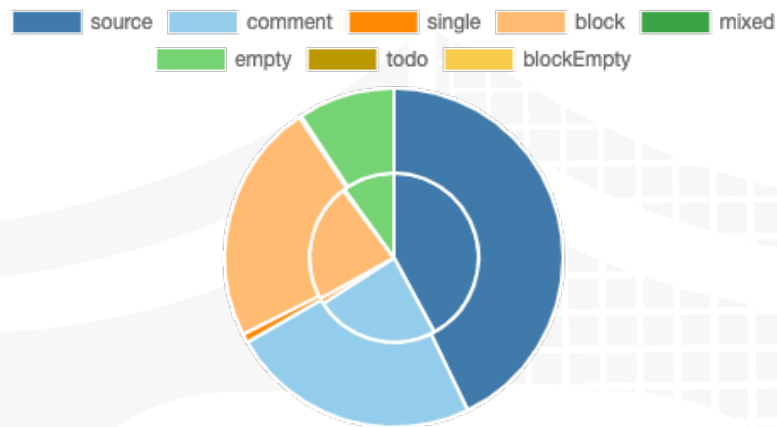
*A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.*

## v1.0

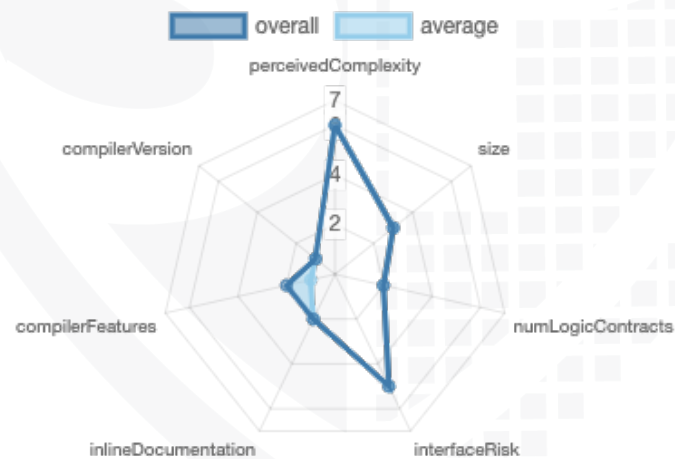
File Name	SHA-1 Hash
contracts/whitedoge.sol	5d9dfdb5e923da22725071d3cc21f7c48379c283

# Metrics

## Source Lines v1.0



## Risk Level v1.0



## Capabilities

### Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	5	5	9	5

### 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	133	6

Version	External	Internal	Private	Pure	View
1.0	79	156	9	28	67

### State Variables

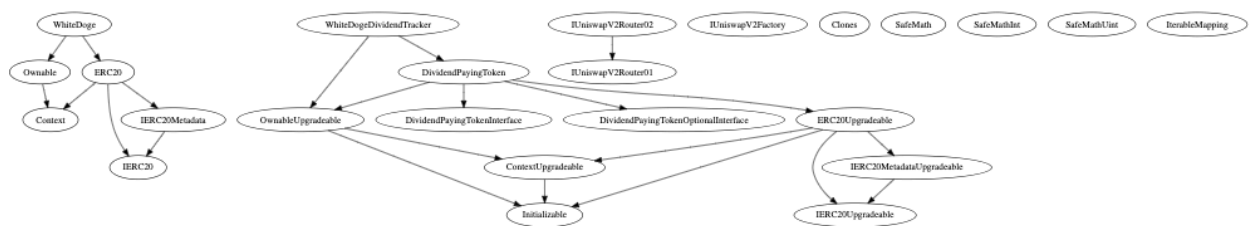
Version	Total	Public
1.0	45	19

### Capabilities

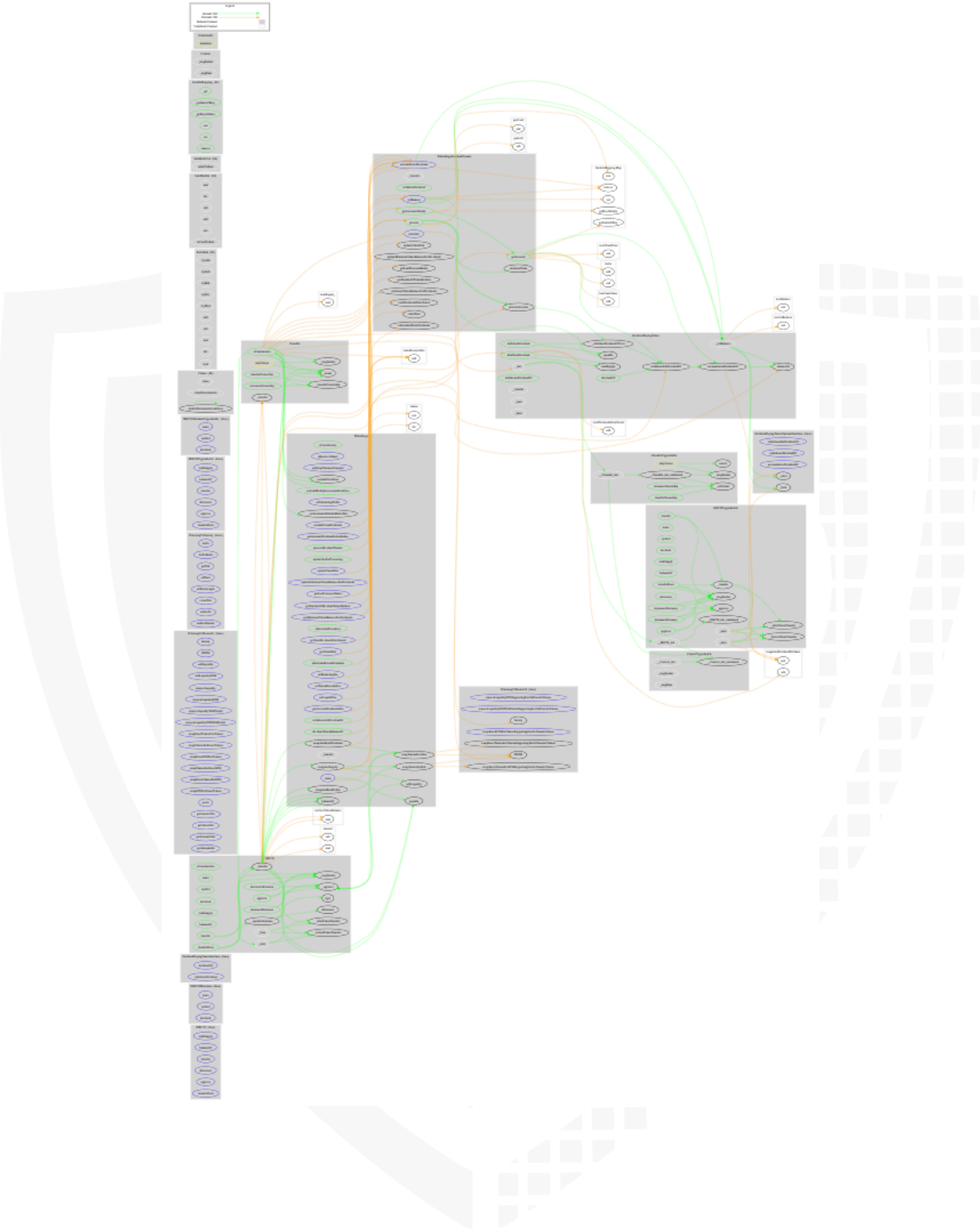
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	^0.8.0		yes	yes (3 asm blocks)	

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	EC Recover	New/Create/Create2
1.0	yes					yes → AssemblyCall:name:create → AssemblyCall:name:create2

## 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. 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

1. approve

2. claim

3. decreaseAllowance

4. excludeFromDividends

5. excludeFromFees

6. excludeMultipleAccountsFromFees

7. increaseAllowance

8. processDividendTracker

9. renounceOwnership

10. setLiquiditFee

11. setMarketingFee

12. setMarketingWallet

13. setSwapTokensAtAmount

14. setTokenRewardsFee

15. transfer

16. transferFrom

17. transferOwnership

18. updateClaimWait

19. updateGasForProcessing

20. updateMinimumTokenBalanceForDividends

## Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓
Max / Total Supply	42.000.000.000.000		

Comments:

**v1.0**

- `_setBalance` function will mint new tokens





## Deployer cannot burn or lock user funds

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

Comments:

**v1.0**

- `_setBalance` function will burn 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

```

  ✓ 🔹 setSwapTokensAtAmount
    🔹 onlyOwner
  ✓ 🔹 excludeFromFees
    🔹 onlyOwner
  ✓ 🔹 excludeMultipleAccountsFromFees
    🔹 onlyOwner
  ✓ 🔹 setMarketingWallet
    🔹 onlyOwner
  ✓ 🔹 setTokenRewardsFee
    🔹 onlyOwner
  ✓ 🔹 setLiquiditFee
    🔹 onlyOwner
  ✓ 🔹 setMarketingFee
    🔹 onlyOwner
  ✓ 🔹 updateGasForProcessing
    🔹 onlyOwner
  ✓ 🔹 updateClaimWait
    🔹 onlyOwner
  ✓ 🔹 updateMinimumTokenBalanceForDividends
    🔹 onlyOwner
  ✓ 🔹 excludeFromDividends
    🔹 onlyOwner
    🔹 processDividendTracker
    🔹 claim

```

```

  🔹 transfer
  🔹 approve
  🔹 transferFrom
  🔹 increaseAllowance
  🔹 decreaseAllowance

```

```

  ✓ 🔹 renounceOwnership
    🔹 onlyOwner
  ✓ 🔹 transferOwnership
    🔹 onlyOwner

```

```

  ✓ 🔹 initialize
    🔹 initializer
  ✓ 🔹 excludeFromDividends
    🔹 onlyOwner
  ✓ 🔹 updateClaimWait
    🔹 onlyOwner
  ✓ 🔹 updateMinimumTokenBalanceForDividends
    🔹 onlyOwner
  ✓ 🔹 setBalance
    🔹 onlyOwner
  🔹 process
  ✓ 🔹 processAccount
    🔹 onlyOwner

```

```

  ✓ 🔹 distributeDividend
    🔹 onlyOwner
  🔹 withdrawDividend

```





## Comments

- Deployer can set following state variables without any limitations
  - swapTokensAtAmount
  - minimumTokenBalanceForDividends
- Deployer can enable/disable following state variables
  - \_isExcludedFromFees
  - excludedFromDividends
- Deployer can set following addresses
  - \_marketingWalletAddress

**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/whitedoge.sol	15	9	2232	1878	1045	704	979	
	Totals	15	9	2232	1878	1045	704	979	

### 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	A floating pragma is set	6	The current pragma Solidity directive is „^0.8.0”.
#3	Main	Missing Zero Address Validation (missing-zero-check)	1978	Check that the address is not zero
#4	Main	Local variables shadowing	1611, 1613, 1656, 1644, 1648, 1652	Rename the local variables that shadow another component
#5	Main	Missing Events Arithmetic	1988-1989, 1994-19951957, 1982-1983	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)		Add the `constant` attributes to state variables that never change
#2	Main	Unused state variables	627	Remove unused state variables
#3	Main	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.
#4	Main	Error message is missing	1723, 1665, 1618	Provide an error message for require statement

## 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.

### 16. March 2022:

- Read whole report for more information





## SWC Attacks

ID	Title	Relationships	Status
<a href="#">SW C-1 36</a>	Unencrypted Private Data On-Chain	<a href="#">CWE-767: Access to Critical Private Variable via Public Method</a>	PASSED
<a href="#">SW C-1 35</a>	Code With No Effects	<a href="#">CWE-1164: Irrelevant Code</a>	PASSED
<a href="#">SW C-1 34</a>	Message call with hardcoded gas amount	<a href="#">CWE-655: Improper Initialization</a>	PASSED
<a href="#">SW C-1 33</a>	Hash Collisions With Multiple Variable Length Arguments	<a href="#">CWE-294: Authentication Bypass by Capture-replay</a>	PASSED
<a href="#">SW C-1 32</a>	Unexpected Ether balance	<a href="#">CWE-667: Improper Locking</a>	PASSED
<a href="#">SW C-1 31</a>	Presence of unused variables	<a href="#">CWE-1164: Irrelevant Code</a>	NOT PASSED
<a href="#">SW C-1 30</a>	Right-To-Left-Override control character (U+202E)	<a href="#">CWE-451: User Interface (UI) Misrepresentation of Critical Information</a>	PASSED
<a href="#">SW C-1 29</a>	Typographical Error	<a href="#">CWE-480: Use of Incorrect Operator</a>	PASSED
<a href="#">SW C-1 28</a>	DoS With Block Gas Limit	<a href="#">CWE-400: Uncontrolled Resource Consumption</a>	PASSED

<a href="#">SW C-1 27</a>	Arbitrary Jump with Function Type Variable	<a href="#">CWE-695: Use of Low-Level Functionality</a>	<b>PASSED</b>
<a href="#">SW C-1 25</a>	Incorrect Inheritance Order	<a href="#">CWE-696: Incorrect Behavior Order</a>	<b>PASSED</b>
<a href="#">SW C-1 24</a>	Write to Arbitrary Storage Location	<a href="#">CWE-123: Write-what-where Condition</a>	<b>PASSED</b>
<a href="#">SW C-1 23</a>	Requirement Violation	<a href="#">CWE-573: Improper Following of Specification by Caller</a>	<b>PASSED</b>
<a href="#">SW C-1 22</a>	Lack of Proper Signature Verification	<a href="#">CWE-345: Insufficient Verification of Data Authenticity</a>	<b>PASSED</b>
<a href="#">SW C-1 21</a>	Missing Protection against Signature Replay Attacks	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	<b>PASSED</b>
<a href="#">SW C-1 20</a>	Weak Sources of Randomness from Chain Attributes	<a href="#">CWE-330: Use of Insufficiently Random Values</a>	<b>PASSED</b>
<a href="#">SW C-11 9</a>	Shadowing State Variables	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	<b>NOT PASSED</b>
<a href="#">SW C-11 8</a>	Incorrect Constructor Name	<a href="#">CWE-665: Improper Initialization</a>	<b>PASSED</b>
<a href="#">SW C-11 7</a>	Signature Malleability	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	<b>PASSED</b>

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

<a href="#">SW</a> <a href="#">C-1</a> <a href="#">05</a>	Unprotected Ether Withdrawal	<a href="#">CWE-284: Improper Access Control</a>	<b>PASSED</b>
<a href="#">SW</a> <a href="#">C-1</a> <a href="#">04</a>	Unchecked Call Return Value	<a href="#">CWE-252: Unchecked Return Value</a>	<b>PASSED</b>
<a href="#">SW</a> <a href="#">C-1</a> <a href="#">03</a>	Floating Pragma	<a href="#">CWE-664: Improper Control of a Resource Through its Lifetime</a>	<b>NOT PASSED</b>
<a href="#">SW</a> <a href="#">C-1</a> <a href="#">02</a>	Outdated Compiler Version	<a href="#">CWE-937: Using Components with Known Vulnerabilities</a>	<b>PASSED</b>
<a href="#">SW</a> <a href="#">C-1</a> <a href="#">01</a>	Integer Overflow and Underflow	<a href="#">CWE-682: Incorrect Calculation</a>	<b>PASSED</b>
<a href="#">SW</a> <a href="#">C-1</a> <a href="#">00</a>	Function Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	<b>PASSED</b>

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

MADE IN GERMANY