



SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Dash2Trade Audit

Security Assessment

07.September,2022

For



DASH2 TRADE



[SolidProof.io](https://solidproof.io)



[@solidproof_io](https://t.me/solidproof_io)

Disclaimer	2
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	24
Source Units in Scope	26
Critical issues	27
High issues	27
Medium issues	27
Low issues	27
Informational issues	28
Audit Comments	28
SWC Attacks	29

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	29.August,2022	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary

Network

Binance (BSC)

Website

<https://dash2trade.vercel.app/>

Twitter

@dash2_trade

Discord

<https://discord.gg/pqynxhnQA2>



Description

TBA

Project Engagement

During the 29th of August 2022, **Dash2Trade** team engaged Solidproof.io to audit the smart contracts that they created. The engagement was technical in nature and focused on identifying the security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo



Contract Links

v1.0

Provided as files

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 analyzing 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:

```
📄 @openzeppelin/contracts/utils/math/SafeMath.sol
📄 @openzeppelin/contracts/access/Ownable.sol
📄 @openzeppelin/contracts/utils/Counters.sol
{ } pragma ... (1)
🔗 D2TPresale
• IERC20
• IpD2T
• IUniswapV2Router01
• IUniswapV2Router02
```

```
📄 @openzeppelin/contracts/access/Ownable.sol
📄 @openzeppelin/contracts/token/ERC20/ERC20.sol
📄 @openzeppelin/contracts/utils/math/SafeMath.sol
{ } pragma ... (1)
• IUniswapV2Router01
• IUniswapV2Router02
• IUniswapV2Factory
```


Tested Contract Files

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

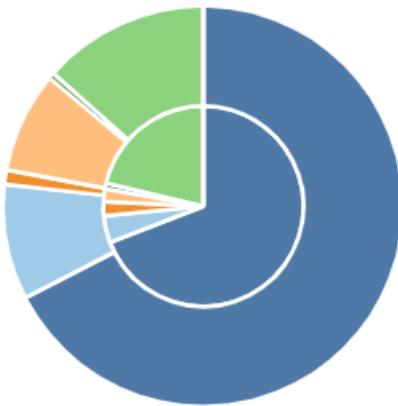
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

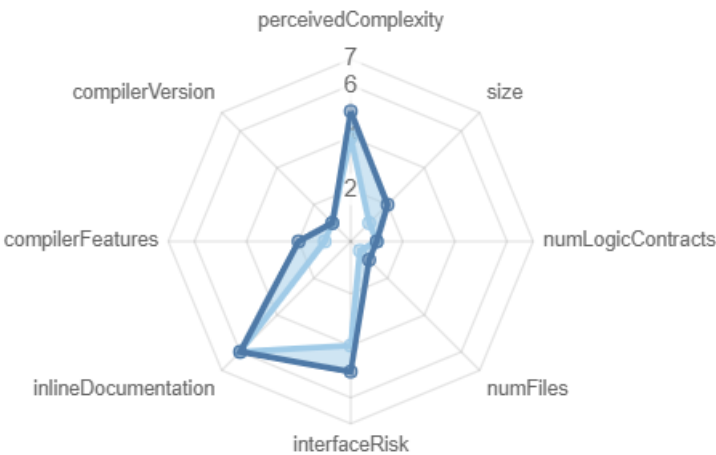
File Name	SHA-1 Hash
contracts/presale.sol	d6e63583ee4991e166c397d152b1f2dde0c97e12
contracts/D2T_Token.sol	344dc58e37744b0663f0aeefaf1fe672d6ebbad7

Metrics

Source Lines v1.0



Risk Level v1.0



Capabilities

v1.0

Components

 Contracts	 Libraries	 Interfaces	 Abstract
2	0	7	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.





 Public	 Payable
89	10







External	Internal	Private	Pure	View
73	59	3	10	23

StateVariables

Total	 Public
30	25

Capabilities

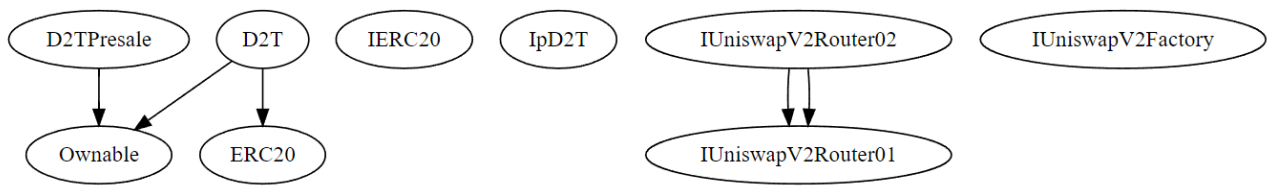
Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
^0.8.7		yes		

 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 Ecrecover	 New/Create/Create2
yes					

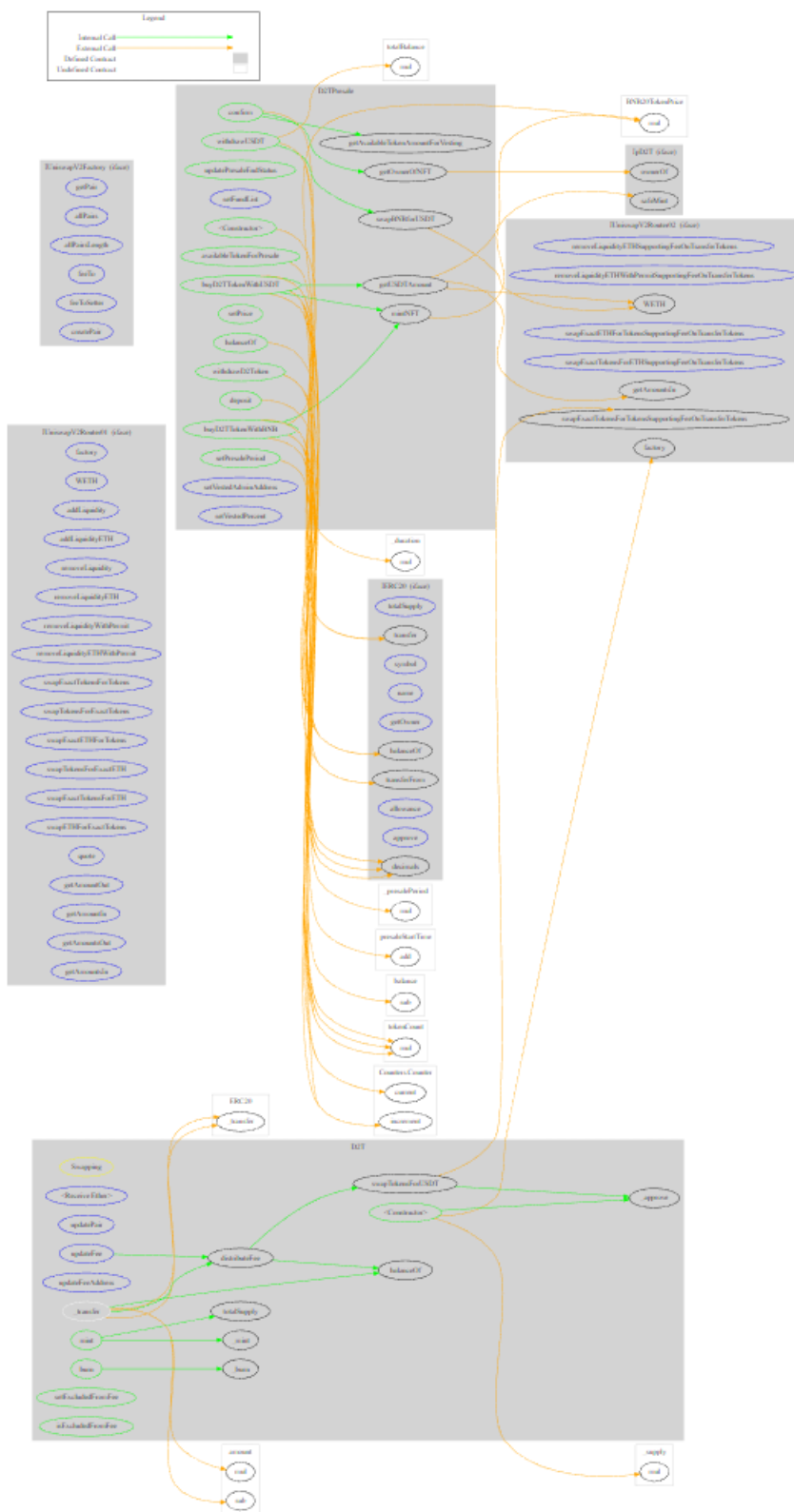
 TryCatch	Σ Unchecked

Inheritance Graph

v1.0



Call Graph



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. Correct implementation of Token standard
3. Deployer cannot mint any new tokens
4. Deployer cannot burn or lock user funds
5. Deployer cannot pause the contract
6. Deployer can set fees
7. Deployer can blacklist/antisnipe address
8. Overall checkup (Smart Contract Security)

Is contract an upgradeable

Name	
Is contract an upgradeable?	No



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			

ERC721				
Function	Description	Exist	Tested	Verified
BalanceOf	Count all NFTs assigned to an owner			
OwnerOf	Find the owner of an NFT			
SafeTransferFrom	Transfers the ownership of an NFT from one address to another address			
SafeTransferFrom	See above - Difference is that this function has an extra data parameter			
TransferFrom	Transfer ownership of an NFT			
Approve	Change or reaffirm the approved address for an NFT			
SetApprovalForAll	Enable or disable approval for a third party ("operator") to manage all of `msg.sender`'s assets			
GetApproved	Get the approved address for a single NFT			
IsApprovedForAll	Query if an address is an authorized operator for another address			

SupportsInterface	Query if a contract implements an interface			
Name	Provides information about the name			
Symbol	Provides information about the symbol			
TokenURI	Provides information about the TokenUri			

Write functions of contracts v1.0

Presale

- ◆ deposit
- ◆ updatePresaleEndStatus
- ◆ setFundList
- ◆ setPresalePeriod
- ◆ withdrawD2Token
- ◆ confirm
- ◆ setPrice
- ◆ buyD2TTokenWithBNB 💰
- ◆ buyD2TTokenWithUSDT
- ◆ withdrawUSDT
- ◆ setVestedAdminAddress
- ◆ setVestedPercent

D2T_Token

- ◆ updatePair
- ◆ updateFee
- ◆ updateFeeAddress
- ◆ mint
- ◆ burn
- ◆ setExcludedFromFee

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint			
Max / Total Supply	N/A		

Comments:

- The supply will be decided at the time of deployment.
- The owner can mint tokens but not more than the max supply.

Deployer cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer cannot lock			
Deployer cannot burn			

Comments:

- The owner can burn tokens from any account.
- Deployer can lock funds in the presale contract because the fees can be set as 100% or more

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause			



Deployer can set fees

Name	Exist	Tested	Status
Deployer can set fees over 25%			
Deployer can set fees to nearly 100% or more			

Comments:

- The fees can be set to 100% or more by the owner because there is no protection against it.

Deployer cannot blacklist/antisnipe addresses

Name	Exist	Tested	Status
Deployer can blacklist/antisnipe addresses			



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

Presale

- deposit
- onlyOwner
- updatePresaleEndStatus
- onlyOwner
- setFundList
- onlyOwner
- setPresalePeriod
- onlyOwner
- withdrawD2Token
- onlyOwner
- confirm
- setPrice
- onlyOwner
- buyD2TokenWithBNB
- buyD2TokenWithUSDT
- withdrawUSDT
- onlyOwner
- setVestedAdminAddress
- onlyOwner
- setVestedPercent

D2T_Token

- updatePair
- onlyOwner
- updateFee
- onlyOwner
- updateFeeAddress
- onlyOwner
- mint
- onlyOwner
- burn
- onlyOwner
- setExcludedFromFee
- onlyOwner

Comments:

The owner can:

- Set tax and modify tax
- Mint but not more than the max supply
- Deposit tokens in the presale contract
- Update presale status
- Set fund list and presale period
- Withdraw the native D2Token from the presale contract after the presale ends but it can be done earlier because the presale period can be set by the owner as many times they please.
- Set the BNB token price for the native token in the presale contract to whatever is desired without any limitations.
- Withdraw USDT from the presale contract
- Include/exclude wallets from fee
- Set vested percent but not more than the percent precision

Source Units in Scope

v1.0

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/presale.sol	1	4	488	267	203	78	279
contracts/D2T_Token.sol	1	3	292	143	99	3	173
Totals	2	7	780	410	302	81	452

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

Issue	File	Type	Line	Description
#1	D2T_Token.sol	Fees can be 100%	206	The owner can set the tax as 100% or more which may result in loss and lock of user funds.

Low issues

Issue	File	Type	Line	Description
#1	D2T_Token.sol	Missing Events	181,200,206, 214,283	Emit an event for critical parameter changes.
#2	D2T_Token.sol	Missing zero check		Check that the address is not zero
#3	D2T_Token.sol	Shadowing Local Variables	214,207	Rename the local variables that shadow another component
#4	Presale.sol	Missing Events	All	Emit an event for critical parameter changes. There are no events in the whole contract.
#5	Presale.sol	Missing zero check	71,216	Check that the address is not zero
#6	All	Floating Pragma	-	The current pragma Solidity directive is “^0.8.7”. Contracts should be deployed with the same compiler version and flags

				that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions.
--	--	--	--	--

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.
#2	D2T_Token.sol	Constable State	173	The state variable should be declared constant
#3	D2T_Token.sol	Unchecked return value	63,108,121	Always check the return value
#4	Presale.sol	Unchecked return value	155-173,196	Always check the return value

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.

07. September, 2022:

- There is still an owner (Owner still has not renounced ownership)
- In the presale contract there are a few functions which are called in an interface and the actual code of that interface was not provided to us so we can't comment on the source code of those interfaces. However, it can be anything. For example, @L468, and @L350
- Read the whole report and modifiers section for more information.

SWC Attacks

ID	Title	Relationships	Status
SWC-1136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-1135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-1134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-1133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-1132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-1131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED

131			
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:	Write to Arbitrary	CWE-123: Write-what-where Condition	PASSED

<u>1</u> <u>2</u> <u>4</u>	Storage Location		
<u>S</u> <u>W</u> <u>C</u> : <u>1</u> <u>2</u> <u>3</u>	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
<u>S</u> <u>W</u> <u>C</u> : <u>1</u> <u>2</u> <u>2</u>	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
<u>S</u> <u>W</u> <u>C</u> : <u>1</u> <u>2</u> <u>1</u>	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
<u>S</u> <u>W</u> <u>C</u> : <u>1</u> <u>2</u> <u>0</u>	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
<u>S</u> <u>W</u> <u>C</u> : <u>1</u> <u>1</u> <u>9</u>	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED

S W C : 1 1 8	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
S W C : 1 1 7	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED
S W C : 1 1 6	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
S W C : 1 1 5	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
S W C : 1 1 4	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
S W C : 1 1 3	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED

S W C : 1 1 2	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
S W C : 1 1 1	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
S W C : 1 1 0	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
S W C : 1 0 9	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
S W C : 1 0 8	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
S W C : 1 0 7	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED

SWC-1106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED
SWC-1105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SWC-1104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SWC-1103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	NOT PASSED
SWC-1102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SWC-1101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED

<div> <div> <div>S</div> <div>W</div> <div>C</div> <div>.</div> <div>1</div> <div>1</div> <div>0</div> <div>0</div> <div>0</div> <div>1</div> </div> </div>	<div>Function</div> <div>Default</div> <div>Visibility</div>	<div> <div>CWE-710: Improper Adherence</div> <div>to Coding Standards</div> </div>	<div>PASSED</div>
---	--	--	-------------------





SolidProof.io



[@solidproof_io](https://t.me/solidproof_io)

Solid
Proofed

Blockchain Security | Smart Contract Audits | KYC


MADE IN GERMANY