SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Wistaverse

Audit

Security Assessment

11. July, 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	13. June 2023	• Layout project • Automated- /Manual-Security Testing • Summary
1.1	15. June 2023	• Reaudit
	11. July 2023	• Reaudit

Note - This Audit report comprises a security analysis of the **Wistaverse** smart contracts. This analysis did not include functional testing (or unit testing) of the contract's logic.

Network
Polygon

Website
https://www.wistaverse.com/

Twitter
https://twitter.com/wistaverse

Description

Wistaverse is an innovative tool for shared democracy. Reinventing social actions using blockchain technologies and the metaverse to unite, protect and give a voice to people. Your wallet is your digital identity, your avatar is your digital self. Come protest in the Wistaverse in a fully immersive live event with your community. Make sure your voice is heard no matter your age, your health condition or your geographic location.

Project Engagement

During the Date of 11 June 2023, **Wistaverse Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical 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://github.com/wistaverse-developer/contracts/tree/main
  - Commit: 9e8a0a4

v1.1

- https://github.com/wistaverse-developer/contracts/tree/main
  - Commit: 3bedd8e

Note for Investors: We only Audited a simple token contract and a staking contract for the **Wistaverse Team**. However, If the project has other contracts (for example, a Presale contract etc), and they were not provided to us in the audit scope, then we cannot comment on its security, and we are not responsible for it in any way.

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit the vulnerability and the impact of that event on the organization or system. The 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 aspossible.
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
@openzeppelin/contracts/access/Ownable.sol	2
@openzeppelin/contracts/token/ERC20/ERC20.sol	2
@openzeppelin/contracts/utils/structs/EnumerableSet.sol	1

Tested Contract Files

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

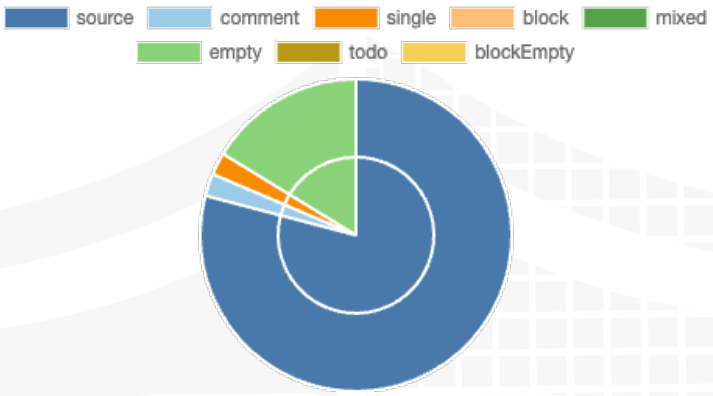
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

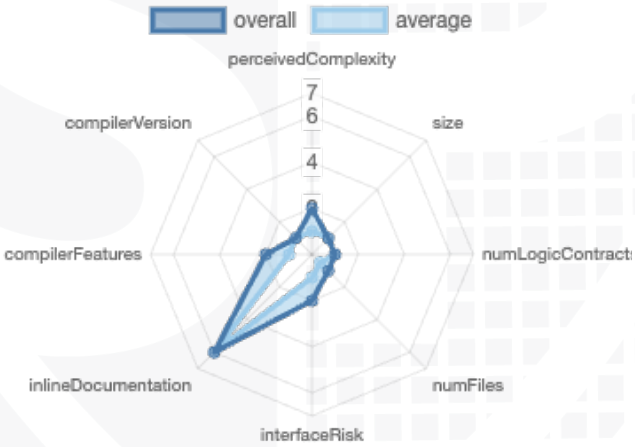
File Name	SHA-1 Hash
contracts/Wistake.sol	c20b4d883a4c901dc480b40fb98d354b3612ec33
contracts/StakingContract.sol	29e33abd5cbe9d91fcc02903c6b8ac0ba92d7be6

Metrics

Source Lines
v1.1



Risk Level
v1.1



Capabilities

Components

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
2	0	0	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
6	0

External	Internal	Private	Pure	View
5	6	0	0	2

StateVariables

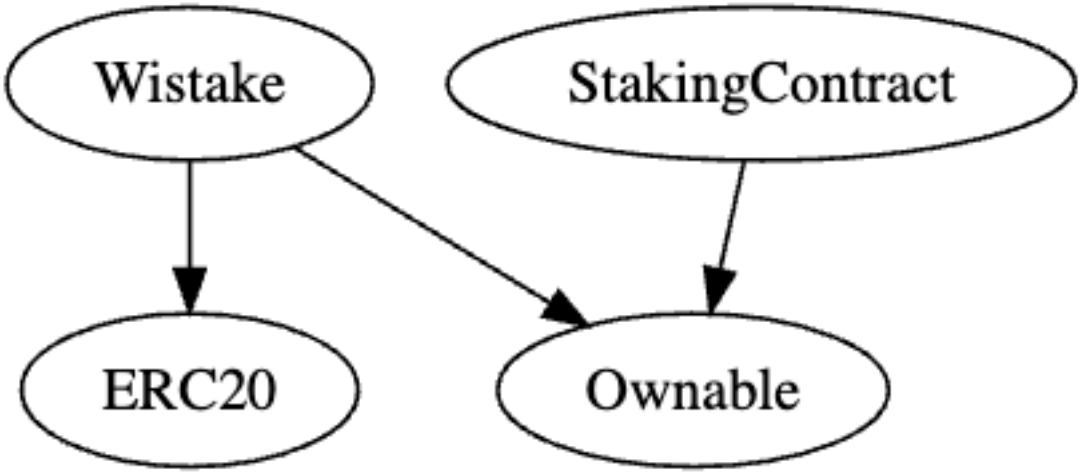
Total	🌐Public
4	0

Capabilities

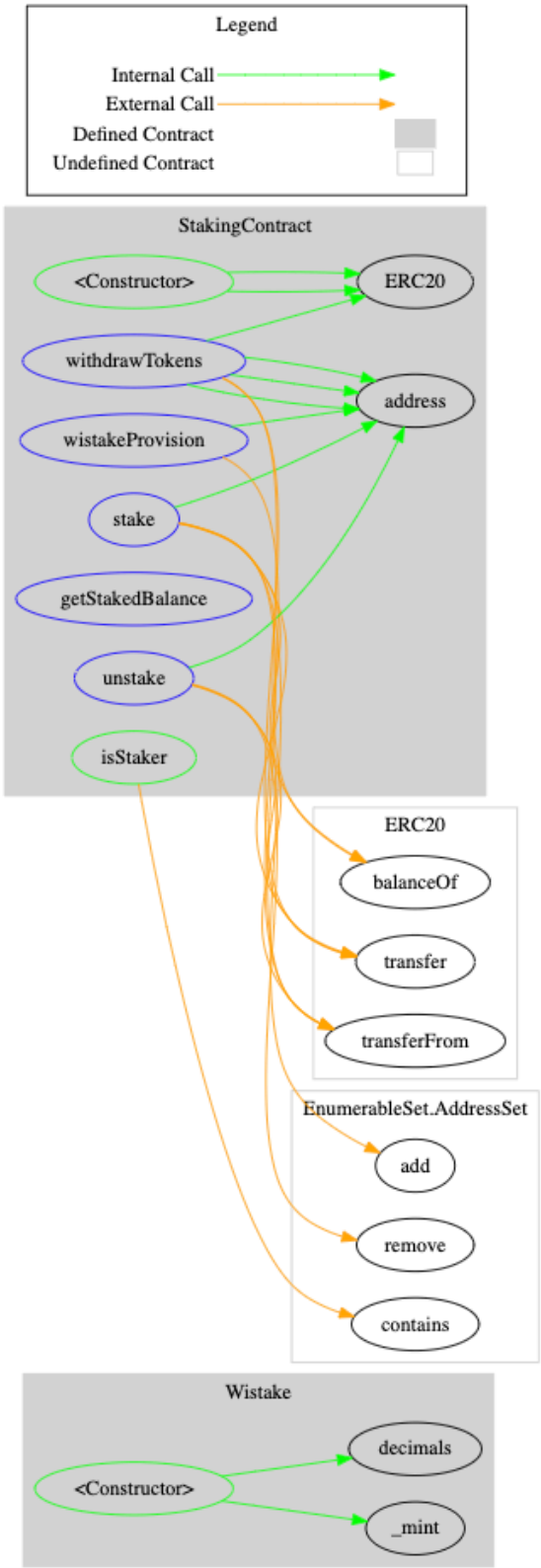
Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
`0.8.17`		—	—	—

📩 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🧹 ECRecover	🌀 New/Create/Create2
`yes`	—	—	—	—	—

Inheritance Graph
v1.0



CallGraph
v1.1



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

```
wistakeProvision
onlyOwner
stake
unstake
```

Ownership Privileges

- The owner can deposit wistake tokens to the contract. Moreover, the users will deposit "Wistaverse Tokens" in the contract and will get equal amounts of wistake tokens. However, this functionality may come to a halt if the owner doesn't deposit wistake tokens in the staking contract

v1.1

```
wistakeProvision
onlyOwner
stake
unstake
withdrawTokens
onlyOwner
```

Ownership Privileges

- The owner can withdraw stuck tokens from contract but cannot withdraw wistake/wistaverse tokens.

Note for Developers - The contract has a risk of being paused unintenationally for staking because if the contract doesn't have enough wistake tokens in the balance then no user will be able to stake anymore.

Source Units in Scope

v1.0

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/Wistake.sol	1	————	11	11	8	1	9
contracts/StakingContract.sol	1	————	66	66	57	1	46
Totals	**2**	————	**77**	**77**	**65**	**2**	**55**

v1.1

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/Wistake.sol	1	————	11	11	8	1	9	————
📝	contracts/StakingContract.sol	1	————	73	73	60	1	51	🧰
📝	**Totals**	**2**	————	**84**	**84**	**68**	**2**	**60**	🧰

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

Medium issues acknowledged

Issue	File	Type	Line	Description	Status
#1	Staking.sol	Missing Timelock	38	The contract doesn't have a timelock for unstaking which means that the funds are available to withdraw right after staking which is not recommended.	Acknowledged

Low issues

Issue	File	Type	Line	Description	Status
#1	Staking.sol	Missing Events Arithmetic	19	Emit an event for critical parameter changes	Acknowledged

Informational issues

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

Audit Comments

We recommend you use the particular form of comments (NatSpec Format, Follow the 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 variable, functions etc., do.

11. July 2023:

- Unit tests with 95% code coverage were not provided to SolidProof, so we cannot ensure complete functional correctness of the code's logic.
- We recommend **Wistaverse** team conduct unit and fuzz tests thoroughly to avoid unwanted logical and calculation errors.
- We recommend carefully setting the addresses in the constructor of the staking contract because they cannot be modified later.
- There is still an owner (The owner still has not renounced ownership)
- Read the whole report and modifiers section for more information

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
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-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	**PASSED**
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	**PASSED**
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	**PASSED**
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	**PASSED**
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	**PASSED**
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	**PASSED**
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	**PASSED**
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	**PASSED**
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	**PASSED**
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	**PASSED**
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	**PASSED**

SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	**PASSED**
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	**PASSED**
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	**PASSED**
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	**PASSED**
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	**PASSED**
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	**PASSED**

SolidProof_io

@solidproof_io

SolidProofed

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY