



SOLIDProof
Bring trust into your projects

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

MADE IN GERMANY

Trust Bitcoin Referrals

Audit

**Security Assessment
29. May, 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	20
Source Units in Scope	21
Critical issues	22
High issues	22
Medium issues	22
Low issues	22
Informational issues	22
Audit Comments	22
SWC Attacks	23

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	23. May 2023	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary
1.1	29. May 2023	<ul style="list-style-type: none">• Reaudit

Note - This Audit report consists of a security analysis of the **Trust Bitcoin Referral** smart contract. This analysis did not include functional testing (or unit testing) of the contract’s logic.

Network

TBA

Website

www.trustbitcoin.io

Telegram

TBA

Twitter

TBA

Description

Trust Bitcoin 100% Decentralised Investment Plan. Trust Bitcoin is created as a way for people to send money over the internet. The digital currency will be intended to provide an alternative payment system that would operate free of central control but otherwise be used just like traditional currencies.

Project Engagement

During the Date of 22 May 2023, **Trust Bitcoin 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

- Provided as files

v1.1

- 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 they were discovered.

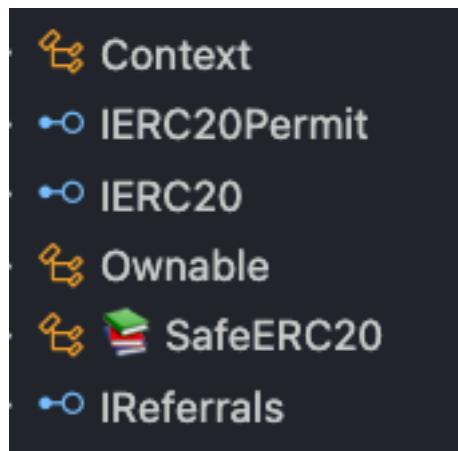
Methodology

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.

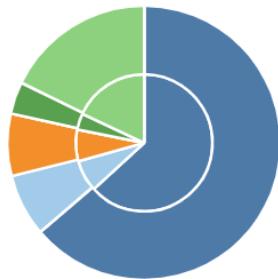
v1.0

File Name	SHA-1 Hash
contracts/Referrals.sol	bba3c24c0ab407329277d79f712b24a6ee6e3cc8

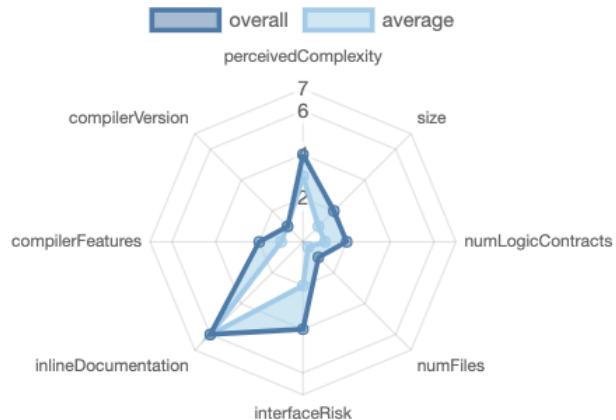
Metrics

Source Lines v1.0

source comment single block mixed
empty todo blockEmpty



Risk Level v1.0



Capabilities

Components

Contracts	Libraries	Interfaces	Abstract
1	0	0	2

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
7	0

External	Internal	Private	Pure	View
2	13	0	0	6

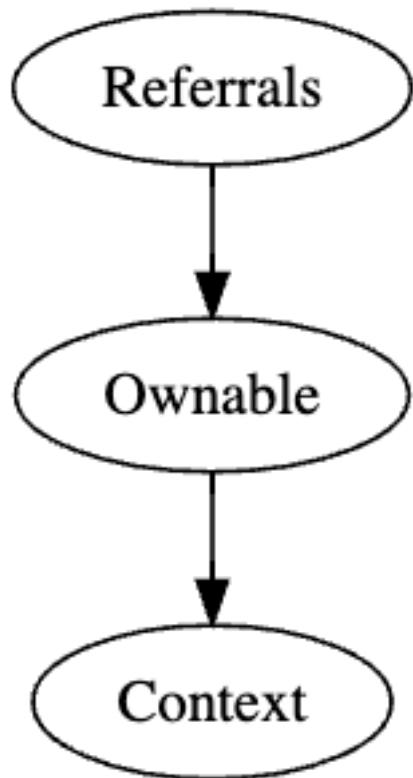
StateVariables

Total	Public
6	5

Capabilities

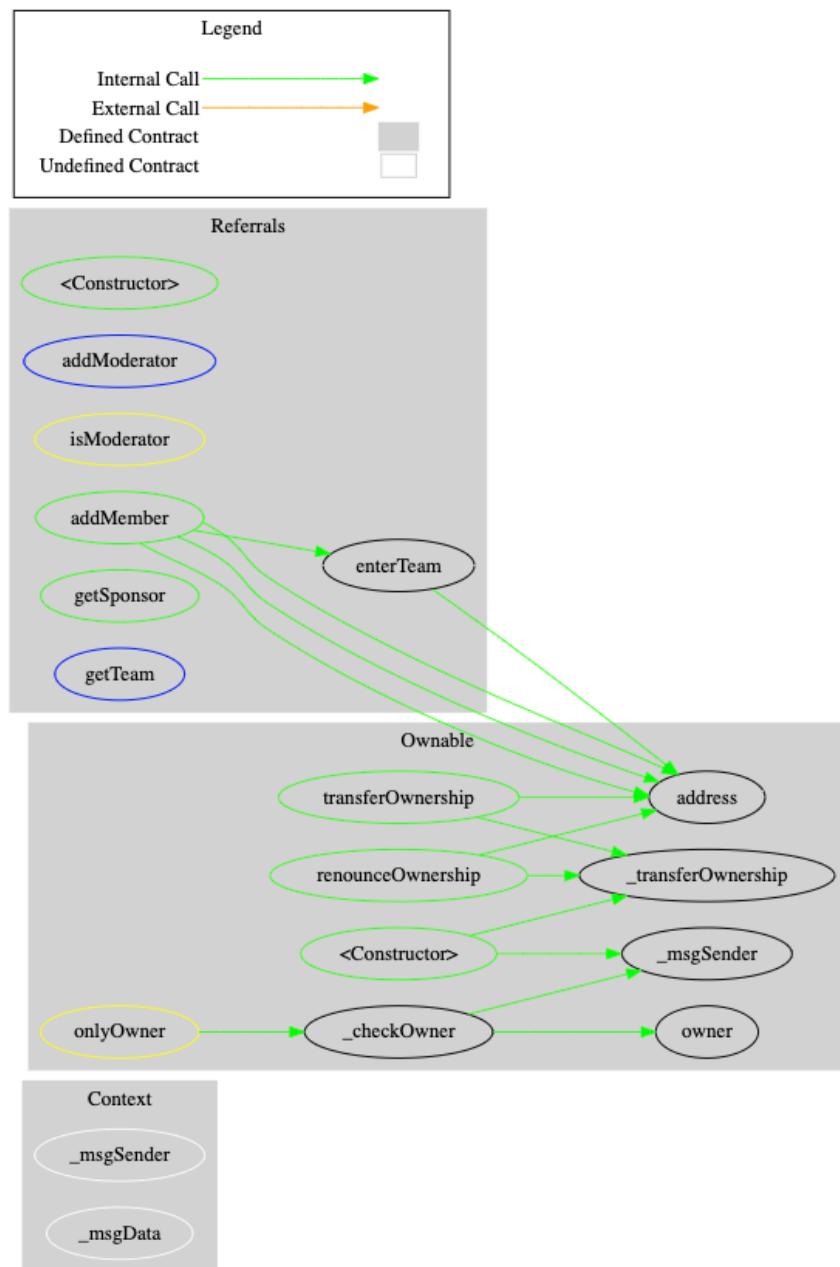
Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
0.8.2				
Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover
TryCatch	Unchecked			

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. Is contract an upgradeable
2. Deployer cannot lock user funds
3. Deployer cannot pause the contract
4. Deployer cannot set fees
5. Overall checkup (Smart Contract Security)

Is contract an upgradeable

Name

Is contract an upgradeable?

No



Deployer cannot lock user funds

Name	Exist	Tested	Status
Deployer can lock	-	-	-

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	-	-	-

Deployer cannot set fees

Name	Exist	Tested	Status
Deployer can set fees over 10%	-	-	-
Deployer can set fees to nearly 100% or to 100%	-	-	-

Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

Attribute	Symbol
Verified / Checked	✓
Partly Verified	🚩
Unverified / Not checked	✗
Not available	-

Modifiers and public functions

v1.1

◆ addMember

Ownership Privileges

- The investment plan contract address can add moderator addresses

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope

v1.0

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/Referrals.sol	3	—	126	126	97	1	60
Totals	3	—	126	126	97	1	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

No medium issues

Low issues

No low issues

Informational issues

No informational issues

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow 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 variables, functions etc. do.

29. May 2023:

- There is still an owner (Owner still has not renounced ownership)
- Owner can deploy a new version of the contract which can change any limit and give owner new privileges
- Read whole report and modifiers section 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

<u>SW C-1 27</u>	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
<u>SW C-1 25</u>	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
<u>SW C-1 24</u>	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
<u>SW C-1 23</u>	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
<u>SW C-1 22</u>	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
<u>SW C-1 21</u>	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
<u>SW C-1 20</u>	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
<u>SW C-11 9</u>	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
<u>SW C-11 8</u>	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
<u>SW C-11 7</u>	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

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

<u>SW C-1 05</u>	Unprotected Ether Withdrawal	<u>CWE-284: Improper Access Control</u>	PASSED
<u>SW C-1 04</u>	Unchecked Call Return Value	<u>CWE-252: Unchecked Return Value</u>	PASSED
<u>SW C-1 03</u>	Floating Pragma	<u>CWE-664: Improper Control of a Resource Through its Lifetime</u>	PASSED
<u>SW C-1 02</u>	Outdated Compiler Version	<u>CWE-937: Using Components with Known Vulnerabilities</u>	PASSED
<u>SW C-1 01</u>	Integer Overflow and Underflow	<u>CWE-682: Incorrect Calculation</u>	PASSED
<u>SW C-1 00</u>	Function Default Visibility	<u>CWE-710: Improper Adherence to Coding Standards</u>	PASSED



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

MADE IN GERMANY