



SOLIDProof

Blockchain Security | Smart Contract Audits

MADE IN GERMANY

Audit Passed

Security Assessment
25. June, 2021

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
Source Lines	9
Risk Level	9
Capabilities	9
CallGraph	10
Source Units in Scope	10
Critical issues	11
High issues	11
Medium issues	11
Low issues	11
Informational issues	11
SWC Attacks	12

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.

Overview

Network

Ethereum (ERC20)

Website

<https://jollyroger.finance/>

Telegram

<https://t.me/jollyrogereth>

Twitter

https://twitter.com/jollyroger_eth

Discord

<https://jollyroger.finance/#>

Medium

<https://jollyrogertoken.medium.com/>

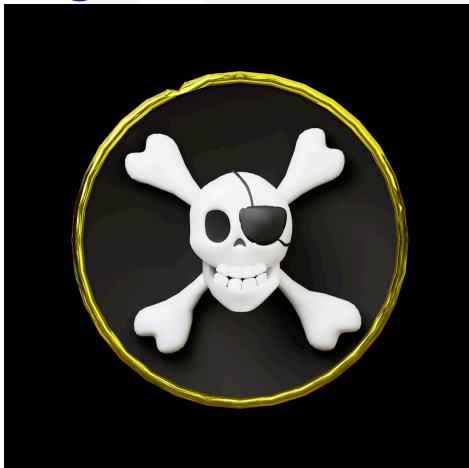
Description

Jolly Roger is the first hybrid decentralized lottery token. Jolly Roger offers multiple opportunities to win large amounts of treasure (\$LOOT). This is done through a bi-weekly lottery system/- giveaway roadmap/community contests that rewards holders. The utility of the \$LOOT token will be used on their DEX, farm, pools and lottery gaming.

Project Engagement

During the 22nd of June, **JollyRoger 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. **JollyRoger Team** provided Solidproof.io with access to their code repository and whitepaper.

Logo



Contract Link

<https://etherscan.io/address/0xa20fcd8cfa055d65c9565dd1c89c89ddcc2309d3#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)

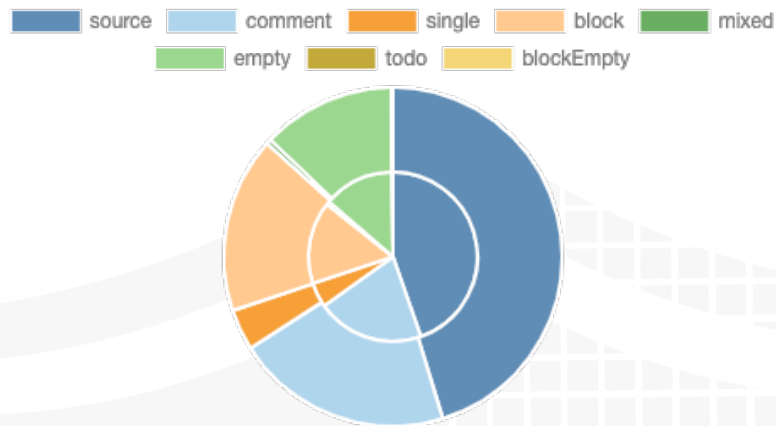
Imported packages:

- OpenZeppelin
 - Address
 - Ownable
 - SafeMatch
- Uniswap
 - UniswapV2Factory
 - UniswapV2Pair
 - UniswapV2Router01
 - UniswapV2Router02

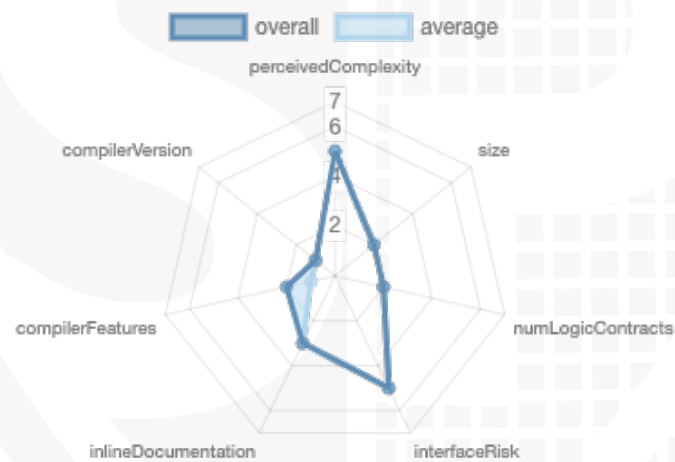


Metrics





Source Lines



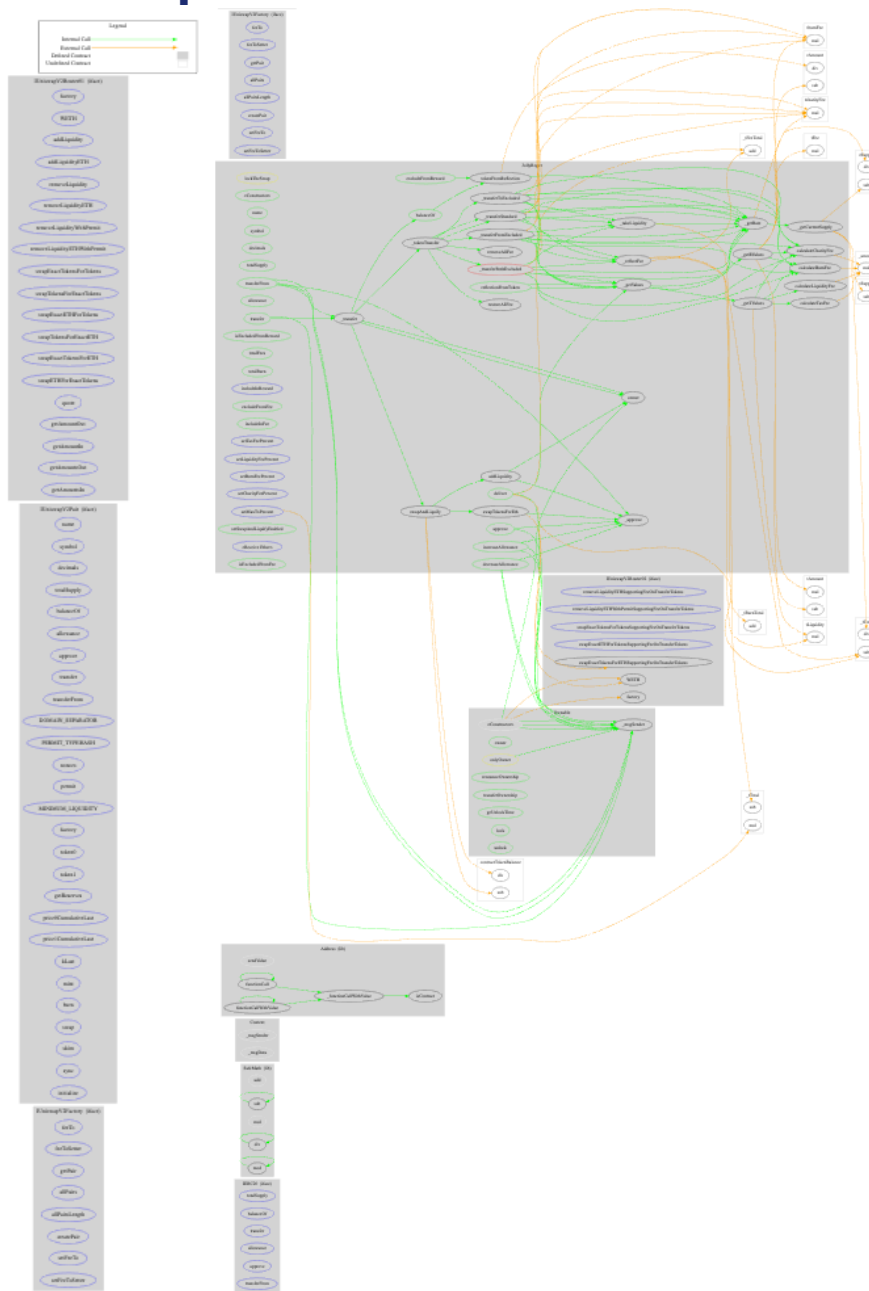
Risk Level



Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
^0.6.12		Yes	Yes (2 asm blocks)	

CallGraph



Source Units in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/poly_roger.sol	5	5	1298	1018	583	344	590	
	Totals	5	5	1298	1018	583	344	590	

Audit Results

AUDIT PASSED

Critical issues

- no critical issues found -

High issues

- no high issues found -

Medium issues

- no medium issues found -

Low issues

- no low issues found -

Informational issues

- no informational issues found -

Issue	File	Type	Line	Description
#1	Main	Multiple calls are executed in the same transaction	779	This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee
#2	Main	A floating pragma is set	26	The current pragma Solidity directive is ""^0.6.12""
#3	Main	Local variables shadowing (shadowing-local)	826	JollyRoger.allowance(address,address).owner shadows: <ul style="list-style-type: none">• Ownable.owner() (Line: 437-439) (function)

SWC Attacks

ID	Title	Relationships	Status
SW C-13 1	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SW C-13 0	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SW C-12 9	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SW C-12 8	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED
SW C-12 7	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SW C-12 5	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SW C-12 4	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SW C-12 3	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SW C-12 2	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED

SW C-12 1	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SW C-12 0	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	NOT 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	NOT PASSED
SW C-11 2	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED

SW C-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SW C-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SW C-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED
SW C-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SW C-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SW C-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	NOT PASSED
SW C-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SW C-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED

SW C-10 0	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
---	-----------------------------------	---	---------------





*Solid
Proofed*

Blockchain Security | Smart Contract Audits


MADE IN GERMANY