SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Juni

Audit

Security Assessment
24. August, 2022

For

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	19. August 2022	• Layout project • Automated- /Manual-Security Testing • Summary
1.1	22. August 2022	• Reaudit
1.2	24. August 2022	• Mainnet address added

Network
Binance Smart Chain (BEP20)

Website
https://www.juni.gg/about
https://juni.gg/

Telegram
https://t.me/JackpotUniverse

Twitter
https://twitter.com/JUNIBSC

Facebook
https://www.facebook.com/JackpotUniverse-103182732442228

Description

JUNI is a **completely new approach** for jackpot projects on blockchain. A **never before seen** smart contract which not only **increases the price floor** for its token constantly, but also **rewards players** of the Jackpot Game with **huge prizes in BNB**.

7 seperate and unique smart contracts are running simultaneously to give our holders and players a **seemless user experience** in the JUNI Ecosystem.

The JUNI team has proven itself as first movers inside the BSC community.

Project Engagement

During the 18th of August 2022, **JUNI 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

v1.1

- Provided as files

v1.2

- https://bscscan.com/address/
  0xB7755d3A6aCc671D941C83cbc9993Dc4C8F3B6E1

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 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/token/ERC20/IERC20.sol	2
@openzeppelin/contracts/utils/Address.sol	3
@openzeppelin/contracts/utils/structs/EnumerableSet.sol	1
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	1
@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol	1
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	2

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/Treasury.sol	22900d01f97428a23abbdd3aec894158e31ae7de
contracts/JackpotToken.sol	12ffa188c155c51d4dab2f194dc725f684ecdd82
contracts/IJackpotGuard.sol	6d5b444e9960514f5a4c3d364e6d42bf38deeccd
contracts/JackpotUniverse.sol	5394d93e2fc6257dc424e15c9ae2af47aab802a2
contracts/Ownable.sol	d454cd2693b50bf37253d76510e246fd9d7cffb0
contracts/IJackpotBroker.sol	bb39093e412b6a5d1d99d248c41cb65f90feef9c

Metrics

Source Lines
v1.0
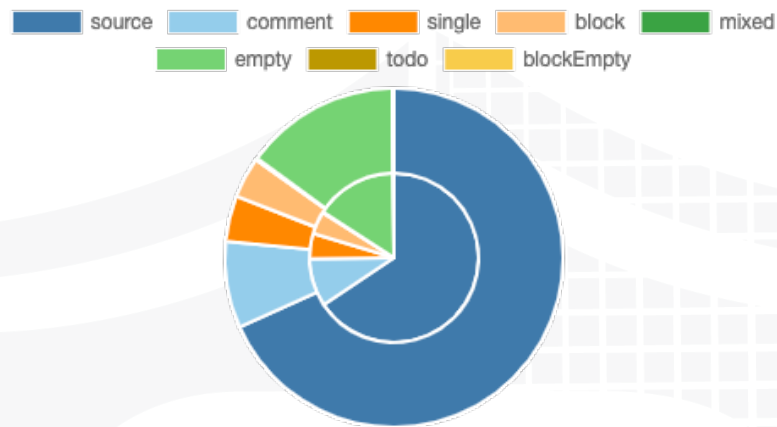
Risk Level
v1.0
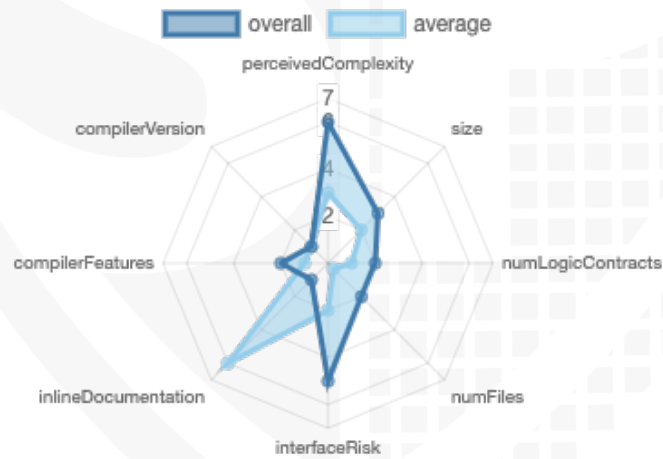
Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	0	2	3

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	99	5

Version	External	Internal	Private	Pure	View
1.0	46	103	12	6	37

State Variables

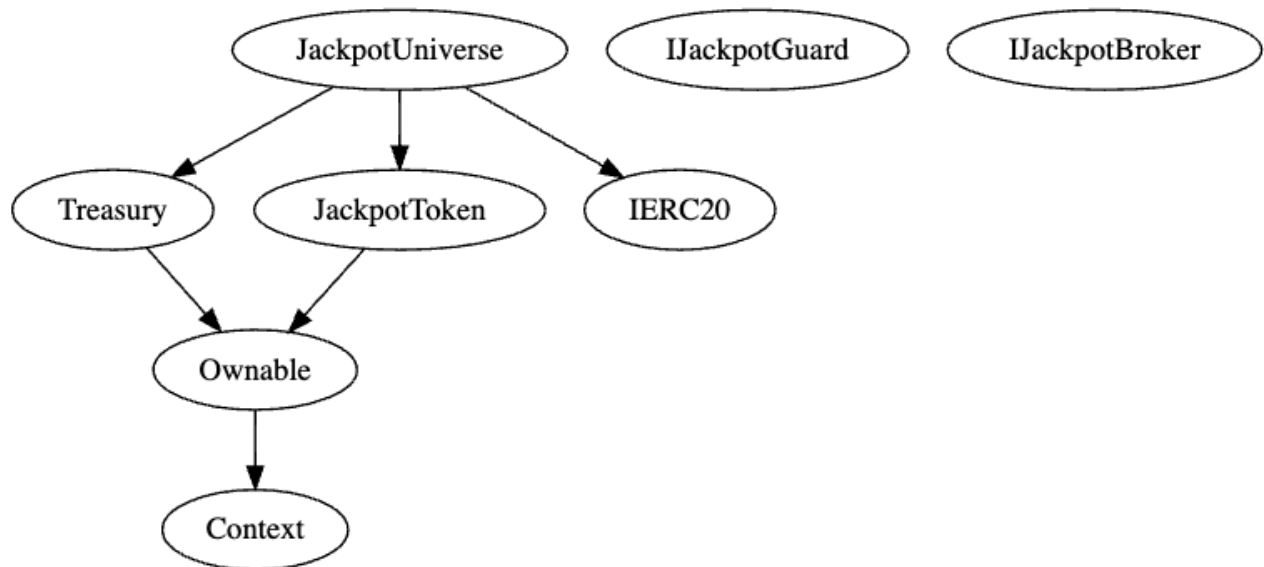| Version	Total	Public
1.0	62	15

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`0.8.16`		yes		

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

Inheritance Graph
v1.0

CallGraph
v1.0
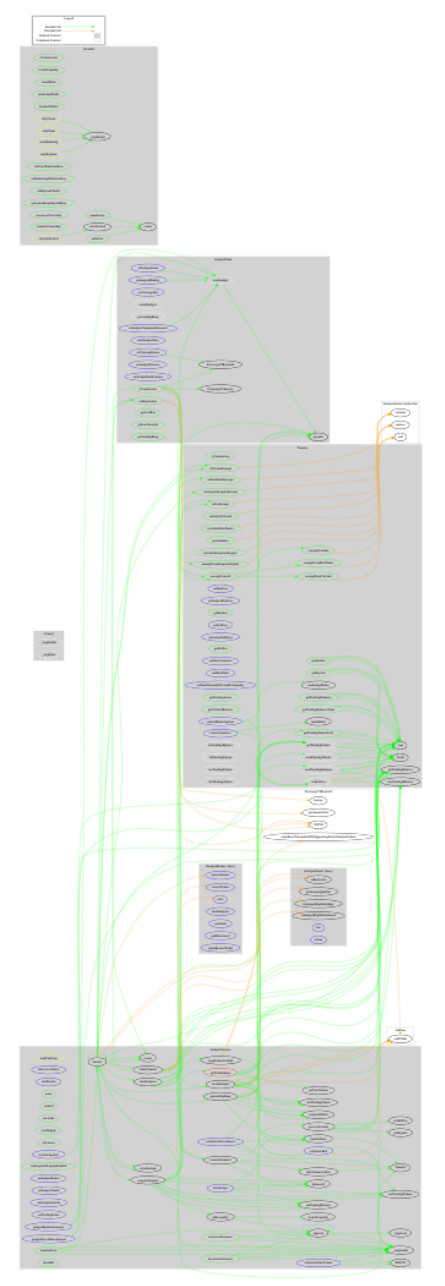
Scope of Work/Verify Claims

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. 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 cannot set fees
7. Deployer cannot blacklist/antisnipe addresses
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	✓	✓	✓

Write functions of contract v1.0

approve	setJackpotStat...
authorize	setJackpotTim...
collectMarketi...	setLockedLiqui...
collectTeamFe...	setMarketing...
decreaseAllow...	setMaxTxAmo...
exemptFromAll	setMaxWallet
exemptFromFee	setNumTokens...
exemptFromM...	setSellFees
exemptFromS...	setSwapAndLi...
exemptFromTx...	setTeamWalle...
fundJackpot	setTradingStat...
includeInFee	setUniswapPair
includeInMax...	setUniswapRo...
includeInSwap...	transfer
includeInTxLimit	transferFrom
increaseAllow...	transferOwner...
renounceOwn...	unauthorize
resetJackpotExt	withdrawBnb
setBuybackW...	withdrawNati...
setBuyFees	withdrawOthe...
setJackpotBro...	
setJackpotBuy...	
setJackpotFeat...	
setJackpotGua...	
setJackpotHar...	
setJackpotLimi...	
setJackpotMin...	
setJackpotSelL...	

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	–	–	–
Max / Total Supply			

Deployer cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer cannot lock	✓	✓	✗
Deployer cannot burn	–	–	–

Comments:

v1.0

- Owner can lock user funds by
  - Setting tradingOpen to false

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	✓	✓	✗

Comments:

v1.0

- Owner can pause contract by setting tradingOpen to false

Deployer cannot set fees

Name	Exist	Tested	Status
Deployer cannot set fees over 25%	✓	✓	✓
Deployer cannot set fees to nearly 100% or to 100%	✓	✓	✓

Deployer can blacklist/antisnipe addresses

Name	Exist	Tested	Status
Deployer cannot blacklist/antisnipe addresses	✓	✓	✗

Comments:

v1.0

- According to the IJackpotGuard interface they can ban users
- JackpotGuard was not provided to solidproof. Please do your own research here
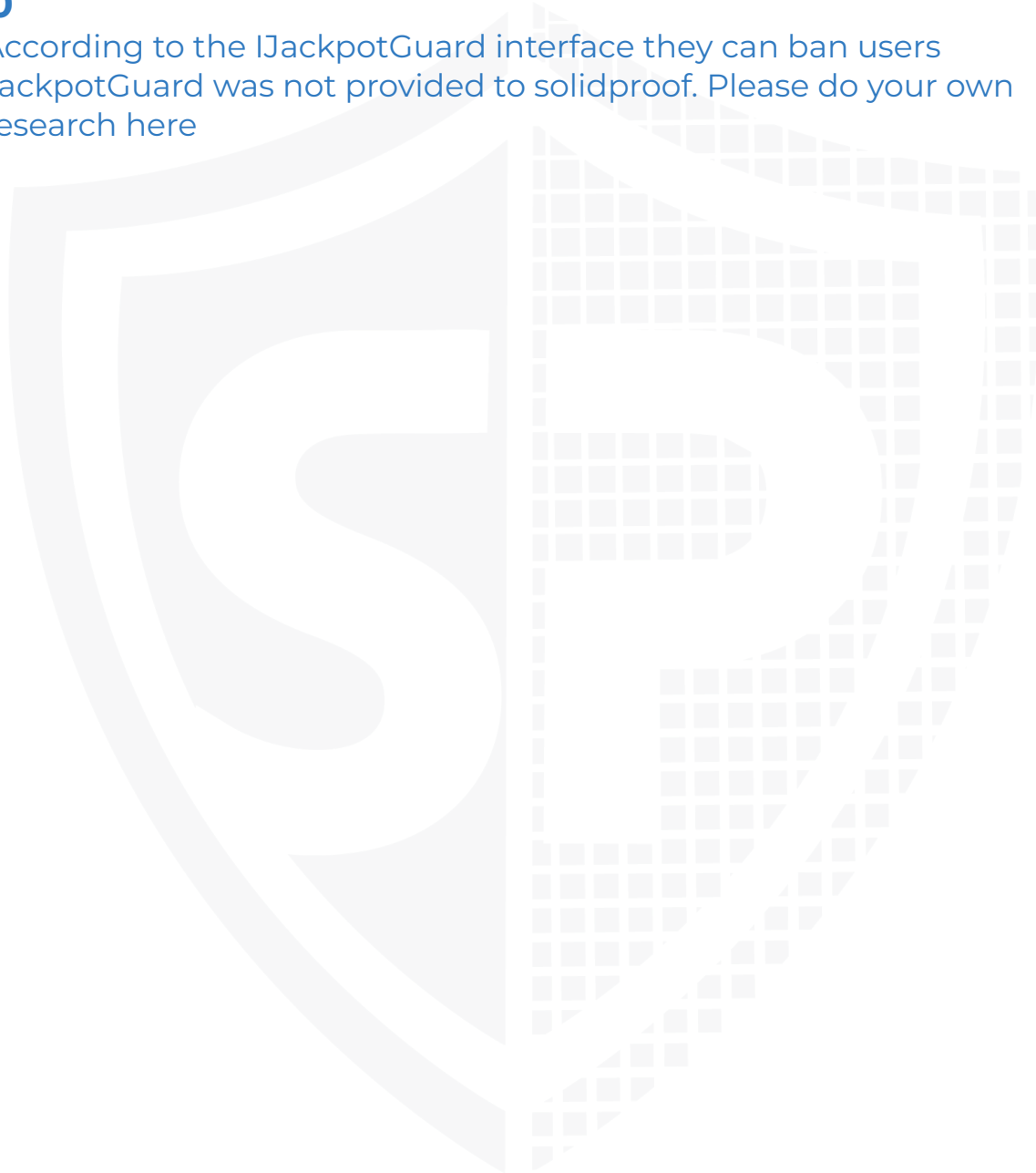
Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

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

Modifiers and public functions
v1.0

Treasury

- exemptFromTxLimit
  - onlyAuthorized
- includeInTxLimit
  - onlyAuthorized
- exemptFromMaxWallet
  - onlyAuthorized
- includeInMaxWallet
  - onlyAuthorized
- exemptFromFee
  - onlyAuthorized
- includeInFee
  - onlyAuthorized
- exemptFromSwapAndLiquify
  - onlyOwner
- includeInSwapAndLiquify
  - onlyOwner
- exemptFromAll
  - onlyOwner
- setBuyFees
  - onlyAuthorized
- setJackpotBuyFees
  - onlyAuthorized
- setSellFees
  - onlyAuthorized
- setJackpotSellFees
  - onlyAuthorized
- setMaxTxAmount
  - onlyAuthorized
- setMaxWallet
  - onlyAuthorized
- setNumTokensSellToAddToLiquidity
  - onlyAuthorized
- collectMarketingFees
  - onlyMarketing
- collectTeamFees
  - onlyTeam

JackpotUniverse

- transfer
- approve
- transferFrom
- increaseAllowance
- decreaseAllowance
- fundJackpot 💰
  - onlyAuthorized
- setSwapAndLiquifyEnabled
  - onlyOwner
- setJackpotBroker
  - onlyOwner
- setJackpotGuard
  - onlyOwner
- setJackpotLimited
  - onlyOwner
- withdrawBnb
  - onlyOwner
- withdrawNativeTokens
  - onlyOwner
- withdrawOtherTokens
  - onlyOwner
- setTradingStatus
  - onlyOwner
- resetJackpotExt
  - onlyAuthorized

Ownable
JackpotToken

Ownable	JackpotToken
∨ ♦ setTeamWalletAddress	∨ ♦ setUniswapRouter
⊚ onlyOwner	⊚ onlyOwner
∨ ♦ setMarketingWalletAddress	∨ ♦ setUniswapPair
⊚ onlyOwner	⊚ onlyOwner
∨ ♦ setBuybackWallet	∨ ♦ setJackpotStatus
⊚ onlyOwner	⊚ onlyAuthorized
∨ ♦ setLockedLiquidityAddress	∨ ♦ setJackpotMinBuy
⊚ onlyOwner	⊚ onlyAuthorized
∨ ♦ renounceOwnership	∨ ♦ setJackpotFeatures
⊚ onlyOwner	⊚ onlyAuthorized
∨ ♦ transferOwnership	∨ ♦ setJackpotHardFeatures
⊚ onlyOwner	⊚ onlyAuthorized
∨ ♦ authorize	∨ ♦ setJackpotTimespanInSeconds
⊚ onlyOwner	⊚ onlyAuthorized
∨ ♦ unauthorize	
⊚ onlyOwner	

Note: Not listed functions/modifiers were provided from implemented libraries

Comments
- *Deployer can set following state variables without any limitations*
  - JackpotToken
    - jackpotTimespan
    - jackpotHardLimit
    - jackpotMinBuy
  - Treasury
    - numTokensSellToAddToLiquidity

- *Deployer can enable/disable following state variables*
  - JackpotToken
    - jackpotEnabled
  - Treasury
    - _feeExempt
    - _maxWalletExempt
    - _txLimitExempt

- • _swapExempt
  - • Ownable
    - • authorizations
  - • JackpotUniverse
    - • tradingOpen
    - • jackpotLimited
    - • swapAndLiquifyEnabled

- • *Deployer can set following addresses*
  - • JackpotToken
    - • uniswapV2Pair
    - • uniswapV2Router
  - • Ownable
    - • _owner
    - • _lockedLiquidity
      - • Can be set once only
    - • _buybackWallet
    - • _marketingWallet
    - • _teamWallet
  - • JackpotUniverse
    - • jGuard
    - • jBroker

- • *Existing Modifiers*
  - • onlyAuthorized
  - • onlyBuyback
  - • onlyMarketing
  - • onlyTeam
  - • onlyOwner
  - • lockTheSwap

- • There are several authorities which are authorized to call some functions, that means, if the owner is renounced, another address is still authorized to call functions
  - • Be aware of this
- • JackpotUniverse
  - • Authorized address are able to reset Jackpot (will set "_lastBuyTimestamp to 0 and _lastBuyer to contract address)
  - • fundJackpot function
    - • Authorized addresses are able to call fundJackpot with a msg.value of 0 in L247. If it is necessary to send an amount of bnb, we recommend you to implement a "require statement" that this function is only callable when the

caller pays some bnb like it was applied to the fallback/receive function

- awardJackpot/fundJackpots
  - We recommend you to check the balances before go deeper in the awardJackpot function
- AddLiquidity
 - Be aware of the lockedLiquidity address. The "_lockedLiquidity" contract is not locked in this case because it is just a wallet address where the liquidity will be transferred. The owner can drain the liquidity out of the contract if it has a private key. Also he/she is able to change the address.
- JackpotToken
  - jackpotCashout
    - If value is set to 0 they bnb/tokens amount of buyBack (JackpotUniverse.sol, function: jackpotBuybackAmount L434) will always be 0 and also in the awardJackpot function L486 it will be 0 for tokens out L488 and cashedOut L487. The same applies to the "jackpotBuyerShare" value
      - We recommend you to implement a range above 0 and below MAX_PCT to make sure, that the lastBuyer investor can get an award
    -

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/Treasury.sol	1	——	478	449	342	14	238	——
🎨	contracts/JackpotToken.sol	1	——	251	210	159	11	109	——
🔍	contracts/IJackpotGuard.sol	——	1	29	6	3	1	17	——
📝	contracts/JackpotUniverse.sol	1	——	771	705	515	80	380	💰📥
📝🎨	contracts/Ownable.sol	2	——	250	234	142	61	83	——
🔍	contracts/IJackpotBroker.sol	——	1	76	49	42	1	21	💰
📝🔍🎨	**Totals**	**5**	**2**	**1855**	**1653**	**1203**	**168**	**848**	💰📥

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

AUDIT PASSED

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/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.

22. August 2022:

- Following files were not provided to solidproof
  - JackpotGuard
- We recommend you to do your own research here.

- Read whole report and modifiers section for more information, don't skip that section!
- No test cases were provided by the team
- Team acknowledged informational issues like
    - Unused state variables
    - Unused functions
    - NatSpec documentation

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**