SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment

18. August, 2021

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.

Overview

Network

Binance Smart Chain (BEP20)

Website

https://realdealprotocol.com/

Telegram

https://t.me/RealDealProtocol_Official

Twitter

https://twitter.com/realdealtoken

Reddit

https://www.reddit.com/r/RealDealToken

Description

The digital trading era is the unavoidable future they are facing. Just a couple of years ago the stock market trading was only accessible for wealthy people, and now anyone with a smartphone and internet connection can trade stocks instantly. In addition, the big advancements in cryptocurrencies made digital trading even more accessible and decentralized. Despite all these available opportunities, the general public still looks at digital trading as a complex and difficult piece of the financial system. Their vision is to introduce digital trading to the general public in a simple and gamified form. People of any age worldwide can learn and experience digital trading in a sandboxed, fully automated fair environment.

Project Engagement

During the 19th of July 2021, **RealDeal 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. **RealDeal Team** provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Link

v1.0

https://bscscan.com/address/0x187f5a88b563c52016530565a80ff8d1e000f806#code

v1.1

https://bscscan.com/address/0x42a999dd3263e7d401b61bc19bd4339a3b32f5ed#code

v1.2

https://bscscan.com/address/0xf4c3c0e2afb4dbe5b84ec3050ed6919609d3b9bb#code

v1.3

https://bscscan.com/address/0xeC390a7cf8F8575eBc4689D9A428F3ba2a4d92Fc#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 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)

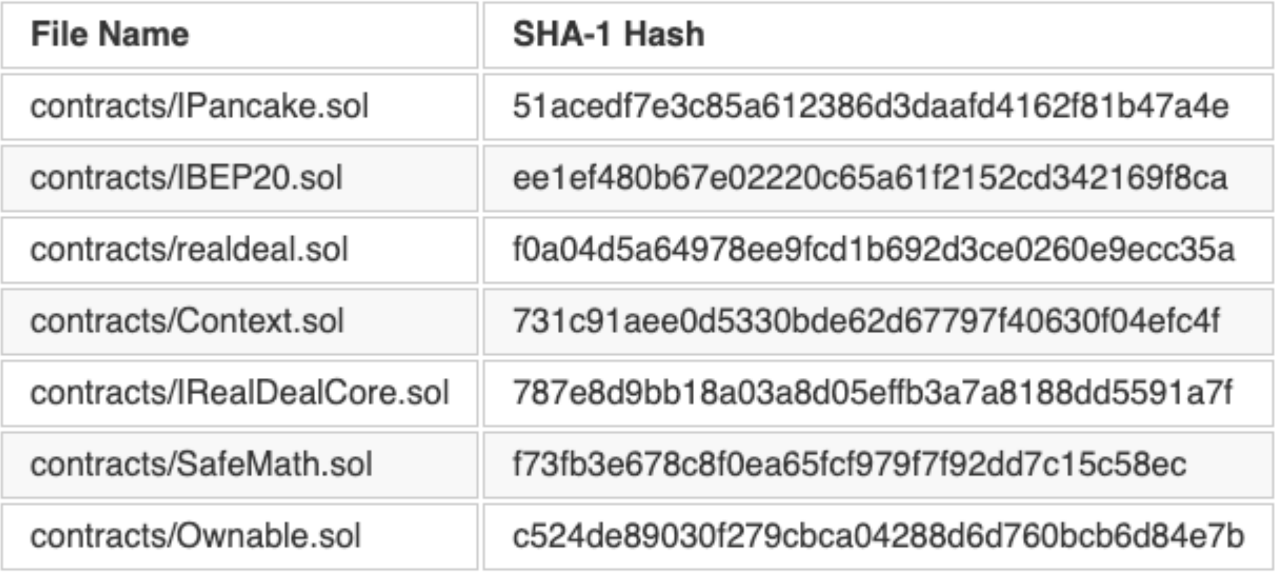
Imported packages:

- OpenZeppelin
  - Ownable
  - SafeMatch
- Pancakeswap
  - PancakeFactory
  - PancakePair
  - PancakeRouter01
  - PancakeRouter02

Tested Contract Files

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

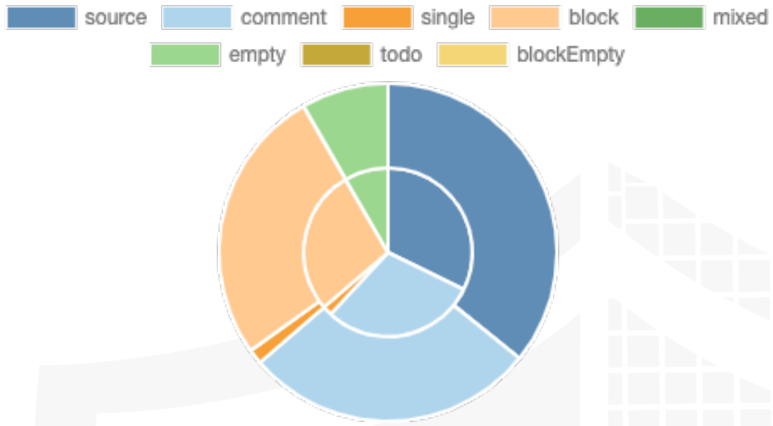
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.



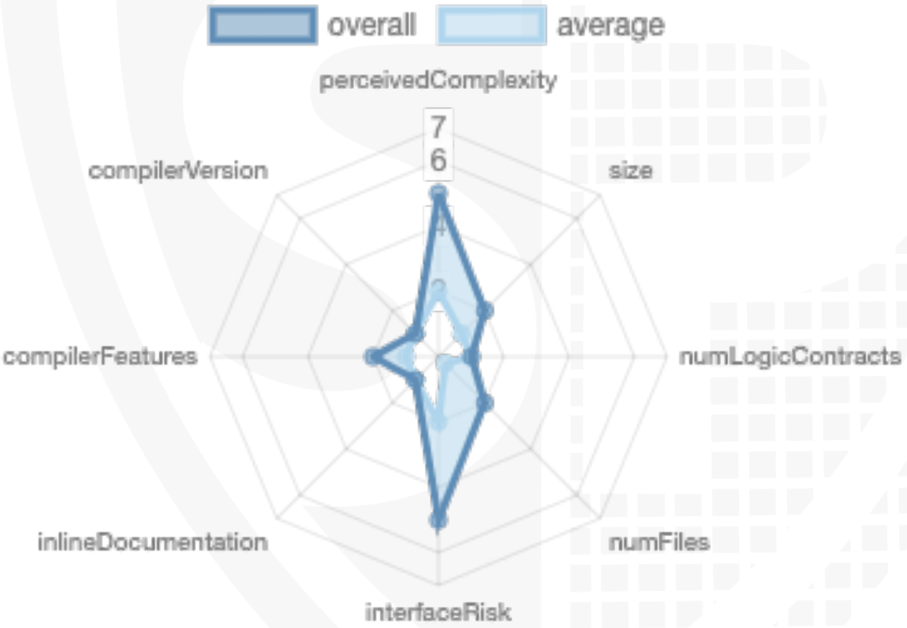
File Name	SHA-1 Hash
contracts/IPancake.sol	51acedf7e3c85a612386d3daafd4162f81b47a4e
contracts/IBEP20.sol	ee1ef480b67e02220c65a61f2152cd342169f8ca
contracts/realdeal.sol	f0a04d5a64978ee9fcd1b692d3ce0260e9ecc35a
contracts/Context.sol	731c91aee0d5330bde62d67797f40630f04efc4f
contracts/IRealDealCore.sol	787e8d9bb18a03a8d05effb3a7a8188dd5591a7f
contracts/SafeMath.sol	f73fb3e678c8f0ea65fcf979f7f92dd7c15c58ec
contracts/Ownable.sol	c524de89030f279cbca04288d6d760bcb6d84e7b

Metrics

Source Lines



Risk Level



Capabilities

Components

Contracts	Libraries	Interfaces	Abstract
1	1	6	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
125	5

External	Internal	Private	Pure	View
118	84	4	23	53

State Variables

Total	Public
31	6

Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts

`^0.8.6`
`0.8.6`
		`yes`	****
			(0 asm blocks)	

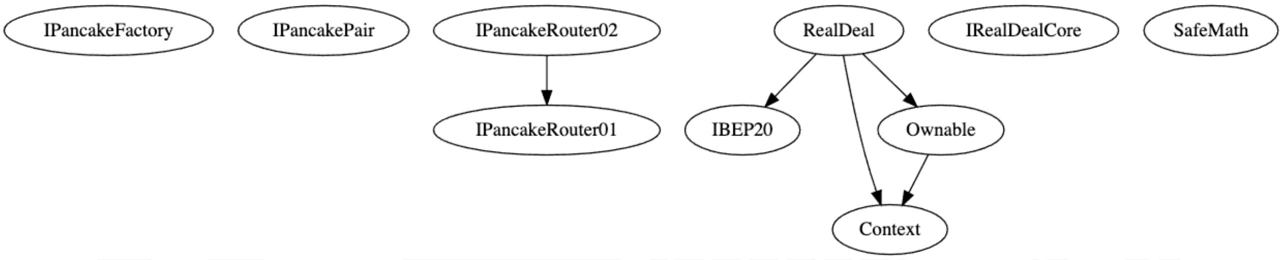
Scope of Work

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. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall checkup (Smart Contract Security)

Inheritance Graph



Verify Claims

Correct implementation of Token standard

Tested	Verified
✓	✓

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	✓	✓	✓

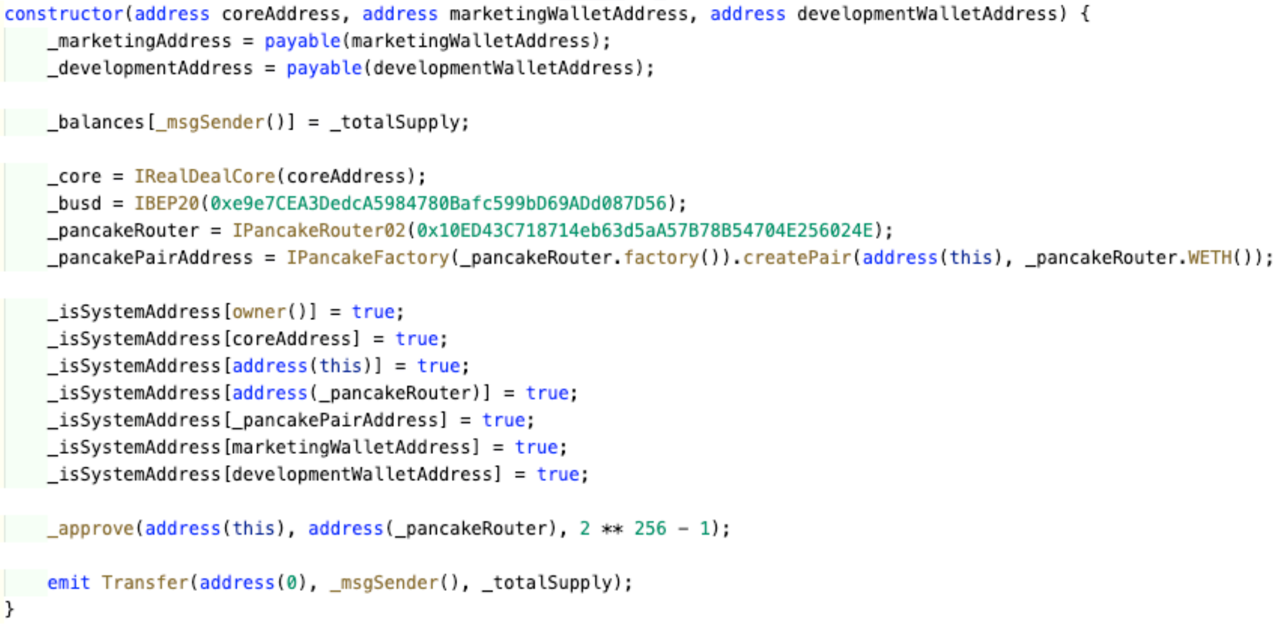
Optional implementations

Function	Description	Exist	Tested	Verified
renounceOwnership	Owner renounce ownership for more trust	✓	✓	✗

Deployer cannot mint any new tokens

Name	Exist	Tested	Verified	File
Deployer cannot mint	✓	✓	✓	Main
Comment	Line: -			

Max / Total Supply: 1.000.000



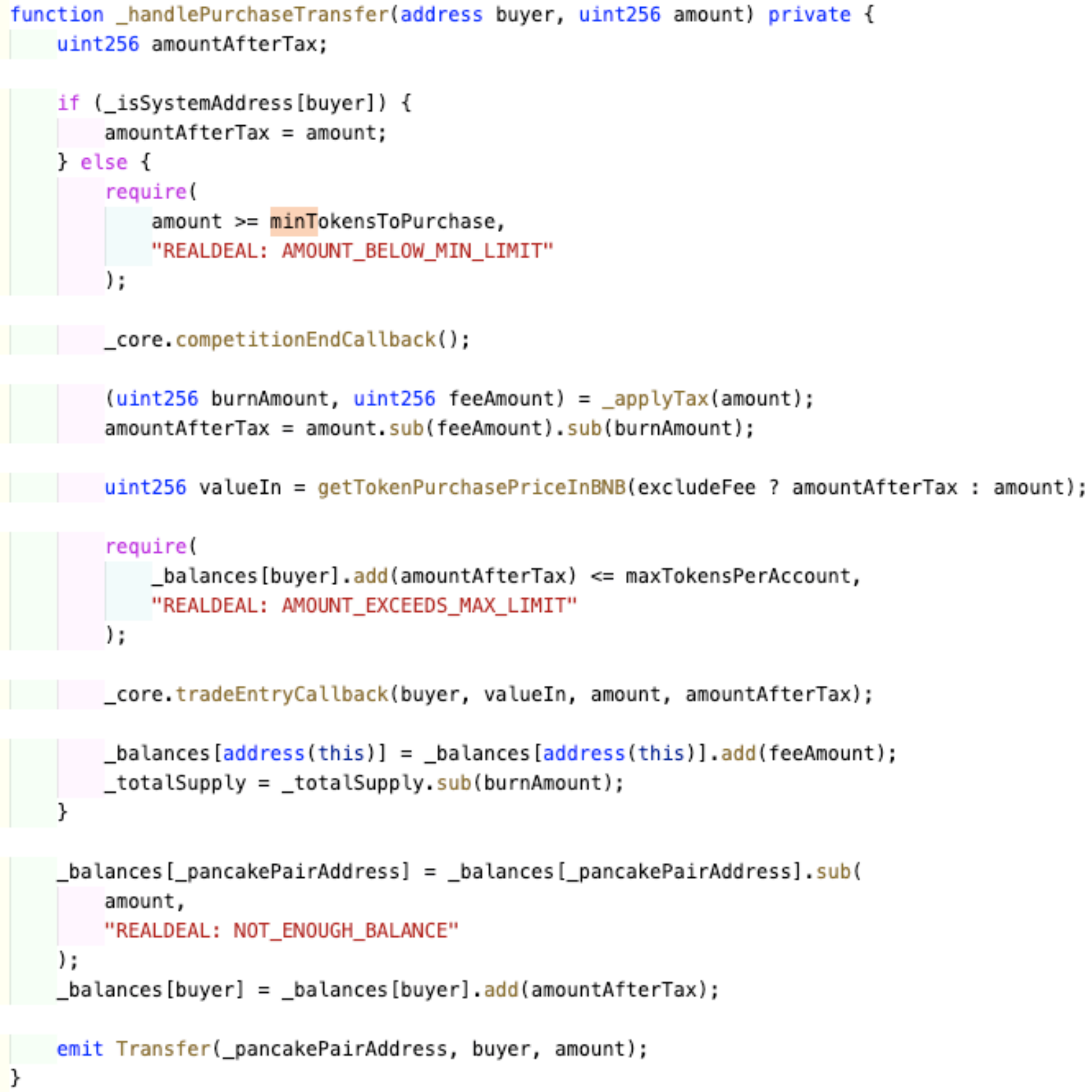
```
constructor(address coreAddress, address marketingWalletAddress, address developmentWalletAddress) {
    _marketingAddress = payable(marketingWalletAddress);
    _developmentAddress = payable(developmentWalletAddress);

    _balances[_msgSender()] = _totalSupply;

    _core = IRealDealCore(coreAddress);
    _busd = IBEP20(0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56);
    _pancakeRouter = IPancakeRouter02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
    _pancakePairAddress = IPancakeFactory(_pancakeRouter.factory()).createPair(address(this), _pancakeRouter.WETH());

    _isSystemAddress[owner()] = true;
    _isSystemAddress[coreAddress] = true;
    _isSystemAddress[address(this)] = true;
    _isSystemAddress[address(_pancakeRouter)] = true;
    _isSystemAddress[_pancakePairAddress] = true;
    _isSystemAddress[marketingWalletAddress] = true;
    _isSystemAddress[developmentWalletAddress] = true;

    _approve(address(this), address(_pancakeRouter), 2 ** 256 - 1);

    emit Transfer(address(0), _msgSender(), _totalSupply);
}
```

Deployer cannot burn or lock user funds

Name	Exist	Tested	Verified
Deployer cannot lock	✓	✓	✓
Deployer cannot burn	✓	✓	🚩

Comments:

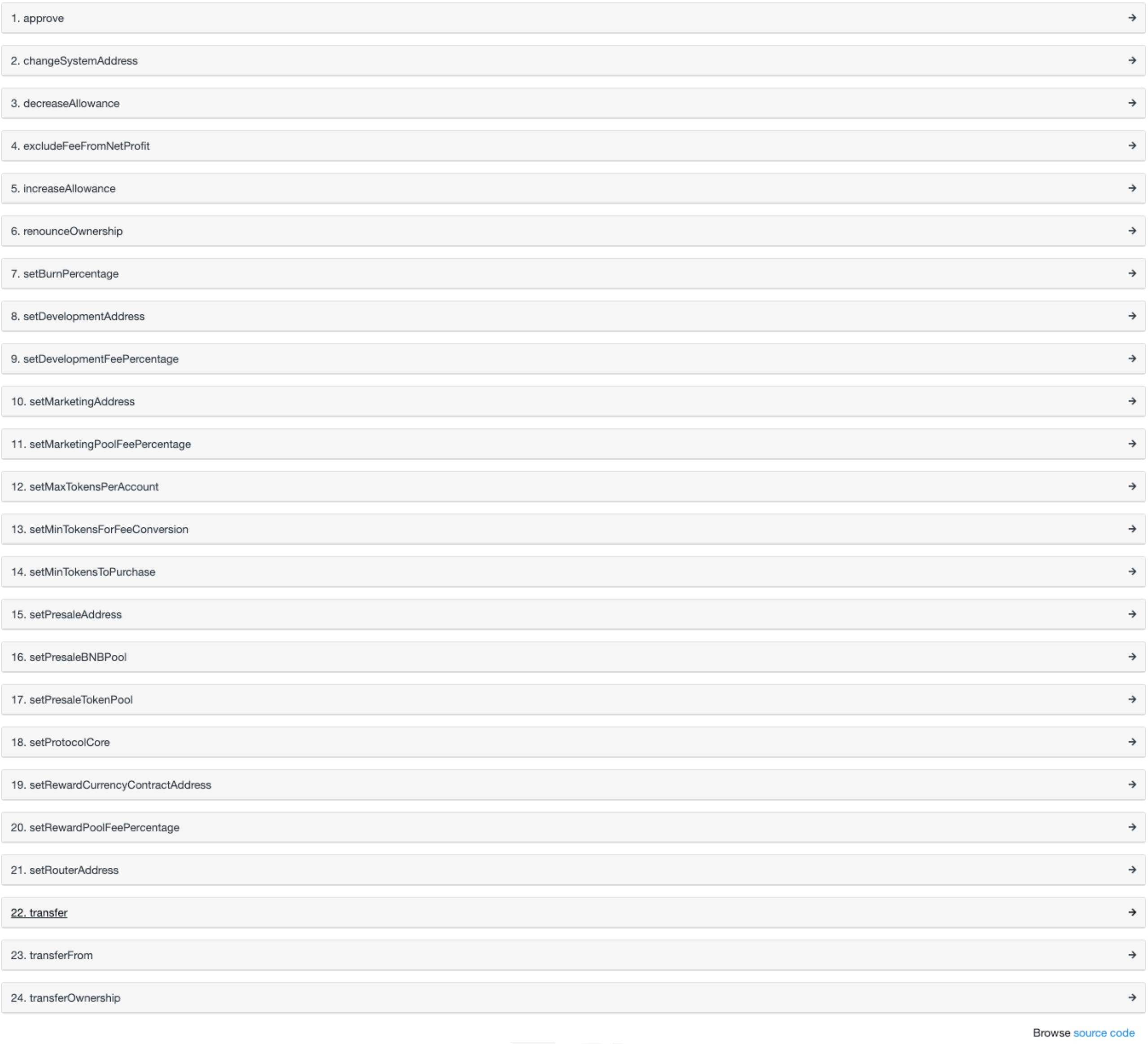
- Deployer cannot burn directly
    - _handlePurchaseTransfer burns from total supply when buyer is not system address



```
function _handlePurchaseTransfer(address buyer, uint256 amount) private {
    uint256 amountAfterTax;

    if (_isSystemAddress[buyer]) {
        amountAfterTax = amount;
    } else {
        require(
            amount >= minTokensToPurchase,
            "REALDEAL: AMOUNT_BELOW_MIN_LIMIT"
        );

        _core.competitionEndCallback();

        (uint256 burnAmount, uint256 feeAmount) = _applyTax(amount);
        amountAfterTax = amount.sub(feeAmount).sub(burnAmount);

        uint256 valueIn = getTokenPurchasePriceInBNB(excludeFee ? amountAfterTax : amount);

        require(
            _balances[buyer].add(amountAfterTax) <= maxTokensPerAccount,
            "REALDEAL: AMOUNT_EXCEEDS_MAX_LIMIT"
        );

        _core.tradeEntryCallback(buyer, valueIn, amount, amountAfterTax);

        _balances[address(this)] = _balances[address(this)].add(feeAmount);
        _totalSupply = _totalSupply.sub(burnAmount);
    }

    _balances[_pancakePairAddress] = _balances[_pancakePairAddress].sub(
        amount,
        "REALDEAL: NOT_ENOUGH_BALANCE"
    );
    _balances[buyer] = _balances[buyer].add(amountAfterTax);

    emit Transfer(_pancakePairAddress, buyer, amount);
}
```

Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	✓	✓	✓



1. approve →

2. changeSystemAddress →

3. decreaseAllowance →

4. excludeFeeFromNetProfit →

5. increaseAllowance →

6. renounceOwnership →

7. setBurnPercentage →

8. setDevelopmentAddress →

9. setDevelopmentFeePercentage →

10. setMarketingAddress →

11. setMarketingPoolFeePercentage →

12. setMaxTokensPerAccount →

13. setMinTokensForFeeConversion →

14. setMinTokensToPurchase →

15. setPresaleAddress →

16. setPresaleBNBPool →

17. setPresaleTokenPool →

18. setProtocolCore →

19. setRewardCurrencyContractAddress →

20. setRewardPoolFeePercentage →

21. setRouterAddress →

22. transfer →

23. transferFrom →

24. transferOwnership →

Browse source code

Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

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

CallGraph

Source Units in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	contracts/IPancake.sol		4	204	13	9	1	136	💰
🔍	contracts/IBEP20.sol		1	93	22	17	66	21	
📝	contracts/realdeal.sol	1		883	839	429	299	355	💰
🎨	contracts/Context.sol	1		24	24	10	12	1	
🔍	contracts/IRealDealCore.sol		1	75	9	3	46	27	
📚	contracts/SafeMath.sol	1		218	218	53	150	10	
🎨	contracts/Ownable.sol	1		68	68	27	33	24	
📝📚🔍🎨	**Totals**	**4**	**6**	**1565**	**1193**	**548**	**607**	**574**	💰

Audit Results

AUDIT PASSED

Critical issues

- no critical issues found -

High issues

- no high issues found -

Medium issues

- no medium issues found -

Low issues

Issue	File	Type	Line	Description
#1	Main	Missing Zero Address Validation (missing-zero-check)	102, 103	RealDeal.constructor(address,address,address).developmentWalletAddress (realdeal.sol:101) lacks a zero-check on : • _developmentAddress = address(developmentWalletAddress) (realdeal.sol#103) • _marketingAddress = address(marketingWalletAddress)

Informational issues

- no informational issues found -

Audit Comments

v.1.1 (31. July 2021):

- New contract address added
- Total supply changed from 10.000.000 to 1.000.000

v1.2 (4. August 2021):

- New contract address added

v1.3 (18. August 2021):

- New contract address added
- There is still an owner (Owner still has not renounced ownership
- maxTokensPerAccount could be 0
  - When the sender is pancake pair address, it is not allowed to use _handlePurchaseTransfer when maxTokensPerAccount is set to 0 because the amount exceeds max limit

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Message call with hardcoded gas amount	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Message call with hardcoded gas amount	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Presence of unused variables	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Presence of unused variables	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

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY