MemeWars

Token and Game

SMART CONTRACT AUDIT

23.11.2021

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer 3
2. About the Project and Company 4
   2.1 Project Overview 5
3. Vulnerability & Risk Level 6
4. Auditing Strategy and Techniques Applied 7
   4.1 Methodology 7
   4.2 Used Code from other Frameworks/Smart Contracts 8
   4.3 Tested Contract Files 9
   4.4 Metrics / CallGraph 10
   4.5 Metrics / Source Lines & Risk 12
   4.6 Metrics / Capabilities 13
5. Scope of Work 16
   5.1 Manual and Automated Vulnerability Test 17
   5.1.1 Unused function 17
   5.1.2 Missing zero address validation on owner functions 18
   5.1.3 Uninitialized state variables 19
   5.1.4 Weak randomness 19
   5.1.5 Public functions should be declared external 20
   5.1.6 Variables using too many digits 21
   5.1.7 A floating pragma is set 21
   5.1.8 Check for boolean equality 22
   5.2. SWC Attacks 23

6. Executive Summary 27
7. Deployed Smart Contract 27

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Swim Company Ltd (MemeWars). If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (15.11.2021)	Layout
0.2 (17.11.2021)	Test Deployment
0.5 (18.11.2021)	Automated Security Testing Manual Security Testing
0.6 (20.11.2021)	Testing SWC Checks
0.7 (21.11.2021)	Verify Claims
0.9 (22.11.2021)	Summary and Recommendation
1.0 (22.11.2021)	Final document
1.1 (23.11.2021)	Adding deployed contract address

2. About the Project and Company

Company address:

Swim Company Ltd
Lavida Plus Officetel
Nguyen Van Linh Street
Tan Phong, District 7
Ho Chi Minh City
Vietnam

Website: https://memewars.finance

Twitter: https://twitter.com/MemeWarsCrypto

Telegram: https://t.me/officialmemewars

Medium: https://medium.com/@MemeWars

2.1 Project Overview

MemeWars is an exciting, cutting-edge blockchain game based entirely on-chain with no off-chain logic whatsoever. The entire game is decentralized and permissionless, creating a first-of-its-kind crypto game that will run autonomously with no assistance needed from the dev team after it's launch (although, improvements can be made). This new type of "decentralized gaming" is different from most of the leading GameFi projects on the market today, which require ongoing dev team support, as well as some off-chain logic to make them run.

MemeWars is built on Binance Smart Chain, and uses only a single currency, the $MWAR token. Unlike most play-to-earn models, MemeWars requires players to stake and burn their tokens in order to participate in the game, creating a new type of GameFi concept called "Burn-to-Earn".

3. 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

4. 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.

4.1 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 Chainsulting 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 Chainsulting 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.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

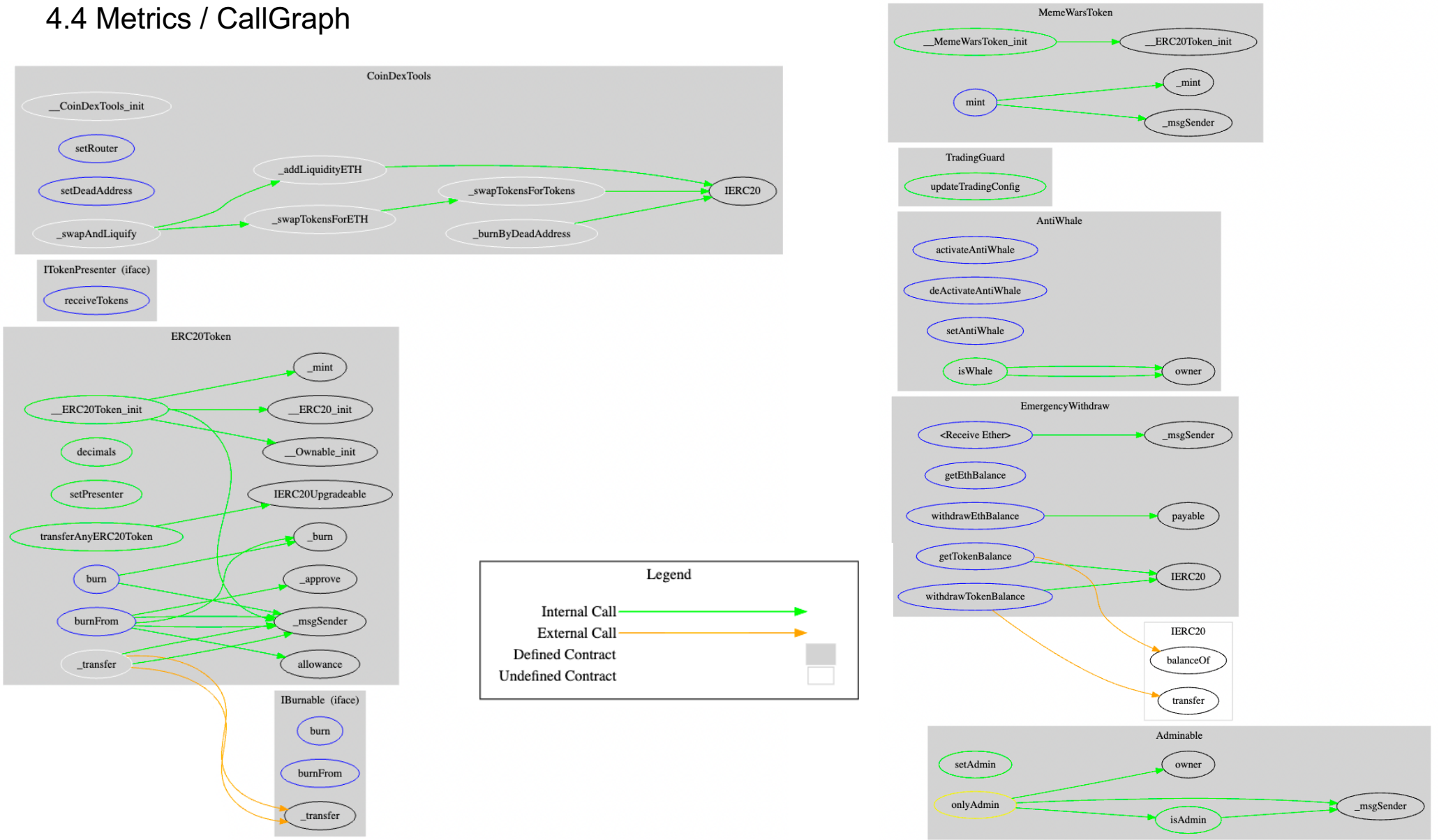
Dependency / Import Path	Source
@chainlink/contracts/src/v0.8/VRFRequestIDBase.sol	https://github.com/smartcontractkit/chainlink/tree/develop/contracts/src/v0.8/VRFRequestIDBase.sol
@chainlink/contracts/src/v0.8/interfaces/LinkTokenInterface.sol	https://github.com/smartcontractkit/chainlink/tree/develop/contracts/src/v0.8/interfaces/LinkTokenInterface.sol
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/access/OwnableUpgradeable.sol
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/security/ReentrancyGuardUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.3.1/token/ERC20/ERC20Upgradeable.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.1/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.3.1/contracts/token/ERC20/extensions/IERC20Metadata.sol

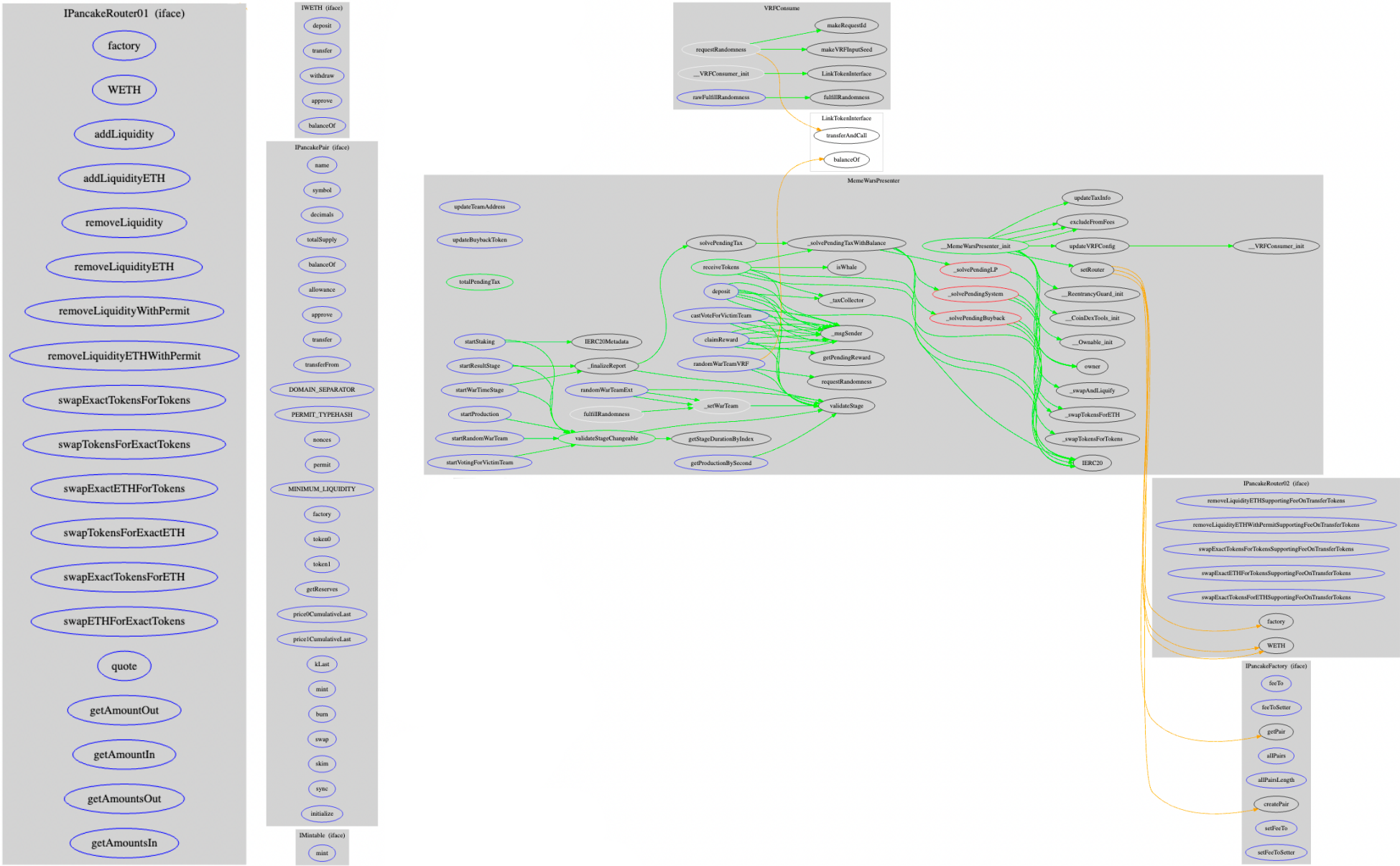
4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

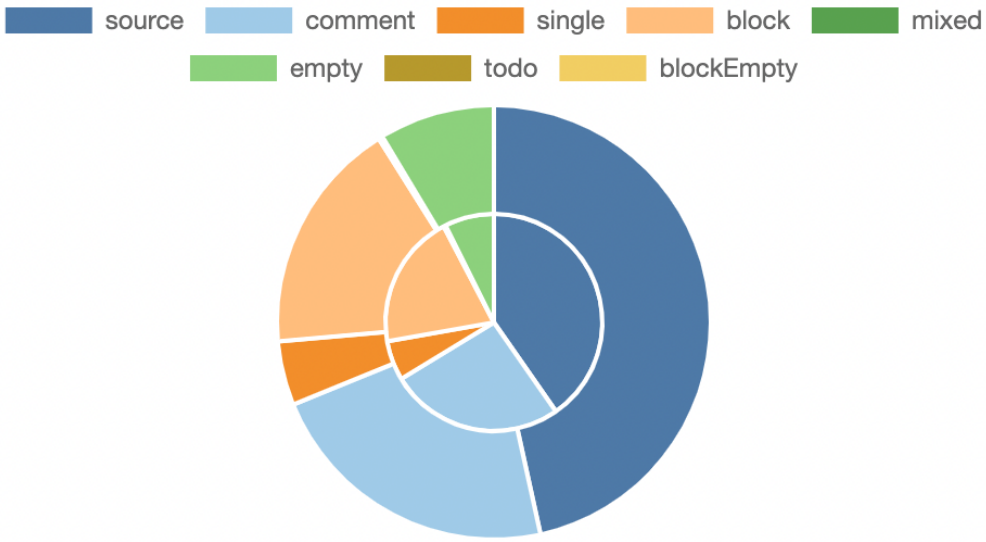
File	Fingerprint (MD5)
interfaces/IMintable.sol	93b4a6dae18c65f9c1055394d8a2b71e
interfaces/IPancakePair.sol	7275b7a292b0728aba0dcb5de6679d40
interfaces/IWETH.sol	63a3d745d3544a347053e1213fabc7c3
interfaces/IPancakeRouter02.sol	00503b1d9e07f59868170420ecf1e891
interfaces/IPancakeRouter01.sol	f038a168e5607ac6b9b4c02c35221de0
interfaces/IBurnable.sol	45a581aafa7de6d56918bcf74d62bd23
interfaces/ITokenPresenter.sol	85fd9e2a830cfe65951d2574573e5d89
interfaces/IPancakeFactory.sol	b0e775a7e605fe9f5b8269ad6da42f6e
MemeWarsPresenter.sol	432ee0c7a6d8fd638fb9d5f703d75157
utils/CoinDexTools.sol	580386e4db4692ec7b02a99b2b840b54
utils/Adminable.sol	39b6270fbdef357bcb5bb95dcc1df905
utils/EmergencyWithdraw.sol	5cab03e0473c4782298192c63a9ce255
utils/AntiWhale.sol	24efdb245b6c079b8e853a7bec38af8b
utils/TradingGuard.sol	a95af16cf8ec77d4624df41f55698a3d
libraries/VRFConsume.sol	8ac6b6ce1d61e2c086e666bb0433c358
MemeWarsToken.sol	9e33e6fefd6f843d2d6ce0c203623d1b
ERC20Token.sol	04d7ca2071c692028bee17002522ab99

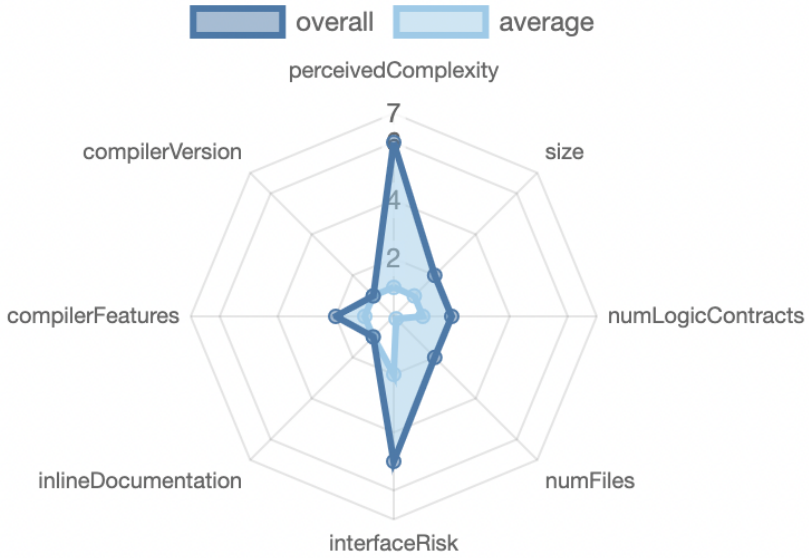
4.4 Metrics / CallGraph




IPancakeRouter01 (iface)
factory
WETH
addLiquidity
addLiquidityETH
removeLiquidity
removeLiquidityETH
removeLiquidityWithPermit
removeLiquidityETHWithPermit
swapExactTokensForTokens
swapTokensForExactTokens
swapExactETHForTokens
swapTokensForExactETH
swapExactTokensForETH
swapETHForExactTokens
quote
getAmountOut
getAmountIn
getAmountsOut
getAmountsIn
IWETH (iface)
deposit
transfer
withdraw
approve
balanceOf
IPancakePair (iface)
name
symbol
decimals
totalSupply
balanceOf
allowance
approve
transfer
transferFrom
DOMAIN_SEPARATOR
PERMIT_TYPEHASH
nonces
permit
MINIMUM_LIQUIDITY
factory
token0
token1
getReserves
price0CumulativeLast
price1CumulativeLast
kLast
mint
burn
swap
skim
sync
initialize
IMintable (iface)
mint
VRFConsume
makeRequestId
requestRandomness
makeVRFInputSeed
__VRFConsumer_init
LinkTokenInterface
rawFulfillRandomness
fulfillRandomness
LinkTokenInterface
transferAndCall
balanceOf
MemeWarsPresenter
updateTeamAddress
updateBuybackToken
totalPendingTax
startStaking
startResultStage
startWarTimeStage
startProduction
startRandomWarTeam
startVotingForVictimTeam
IERC20Metadata
_finalizeReport
randomWarTeamExt
fulfillRandomness
_setWarTeam
validateStageChangeable
getStageDurationByIndex
getProductionBySecond
solvePendingTax
receiveTokens
deposit
castVoteForVictimTeam
claimReward
randomWarTeamVRF
validateStage
_solvePendingTaxWithBalance
isWhale
_taxCollector
_msgSender
getPendingReward
requestRandomness
__MemeWarsPresenter_init
_solvePendingLP
_solvePendingSystem
_solvePendingBuyback
updateTaxInfo
excludeFromFees
updateVRFConfig
setRouter
__ReentrancyGuard_init
__CoinDexTools_init
__Ownable_init
owner
_swapAndLiquify
_swapTokensForETH
_swapTokensForTokens
IERC20
__VRFConsumer_init
IPancakeRouter02 (iface)
removeLiquidityETHSupportingFeeOnTransferTokens
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForETHSupportingFeeOnTransferTokens
factory
WETH
IPancakeFactory (iface)
feeTo
feeToSetter
getPair
allPairs
allPairsLength
createPair
setFeeTo
setFeeToSetter

4.5 Metrics / Source Lines & Risk





4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts

`0.8.4`
`>=0.5.0`
`^0.8.4`
		`yes`	****
			(0 asm blocks)	

📩 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧱 Uses Hash Functions	🪚 ECRecover	🌀 New/Create/Create2
`yes`			`yes`		

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
117	6

External	Internal	Private	Pure	View
96	102	4	10	31

StateVariables

Total	🌐Public
37	26

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	interfaces/IMintable.sol		1	6	5	3	2	3	
🔍	interfaces/IPancakePair.sol		1	97	8	5	1	55	
🔍	interfaces/IWETH.sol		1	15	6	3	1	14	💰
🔍	interfaces/IPancakeRouter02.sol		1	51	7	4	1	16	💰
🔍	interfaces/IPancakeRouter01.sol		1	155	5	3	1	48	💰
🔍	interfaces/IBurnable.sol		1	8	5	3	1	5	
🔍	interfaces/ITokenPresenter.sol		1	11	5	3	1	3	🔆
🔍	interfaces/IPancakeFactory.sol		1	22	7	4	1	17	
📝	MemeWarsPresenter.sol	1		762	733	467	207	293	🛋️🧱
📝	utils/CoinDexTools.sol	1		183	164	90	57	63	🛋️
📝	utils/Adminable.sol	1		21	21	15	1	14	

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	utils/EmergencyWithdraw.sol	1		53	49	23	19	28	💰🔽
📝	utils/AntiWhale.sol	1		62	54	27	19	24	
📝	utils/TradingGuard.sol	1		33	33	21	7	11	
🎨	libraries/VRFConsume.sol	1		197	161	23	164	17	
📝	MemeWarsToken.sol	1		22	22	12	7	14	
📝	ERC20Token.sol	1		93	80	44	27	48	🔽
📝🔍🎨	**Totals**	**9**	**8**	**1791**	**1365**	**750**	**517**	**673**	💰🔽🧱🔆

Legend: [➖]

- **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, ...)

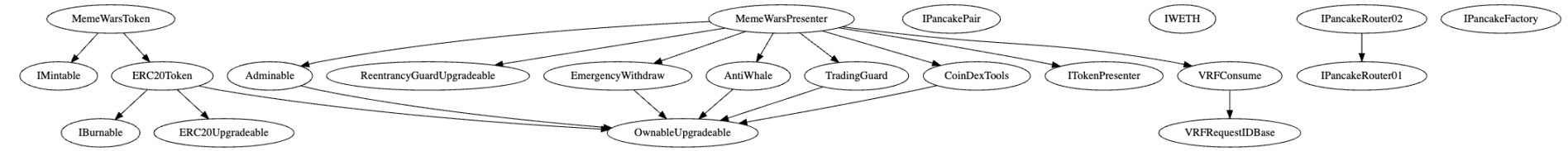
5. Scope of Work

The MemeWars Team provided us with the files that needs to be tested. The scope of the audit are the token and game contracts.

The team put forward the following assumptions regarding the security, usage of the contracts:

- The smart contract is coded according to the newest standards and in a secure way.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES
During the audit, Chainsulting‘s experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES
During the audit, Chainsulting’s experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES
During the audit, Chainsulting’s experts found **no Medium issues** in the code of the smart contract

LOW ISSUES

5.1.1 Unused function
Severity: LOW
Code: SWC-131
Status: ACKNOWLEDGED
File(s) affected: CoinDexTools.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation is an internal function declared, which is not used in any contract.	Line 180-182: _burnByDeadAddress()	It is recommended to remove the unused code to improve code readability and decrease gas consumption.

5.1.2 Missing zero address validation on owner functions
Severity: LOW
Code: NA
Status: ACKNOWLEDGED
File(s) affected: EmergencyWithdraw.sol, ERC20Token.sol, Adminable.sol, MemeWarsPresenter.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several owner functions are not checking for zero addresses. Setting an address to the zero address can result in unintended behavior.	EmergencyWithdraw.sol Line 29: address(_to).transfer(_amount) Adminable.sol Line 10: admin = _admin ERC20Token.sol Line 43: presenter = _presenter MemeWarsPresenter Line 131-135: _token, _admin, _teamAddress, _buybackToken Line 243: router = _router Line 304: teamAddress = _teamAddress Line 312: buybackToken = _buybackToken	It is recommended to check addresses for the zero address before setting them. Indeed, the owner should know what he is doing, the zero address check is preventing unintended bahavior.

5.1.3 Uninitialized state variables

Severity: LOW
Code: NA
Status: ACKNOWLEDGED
File(s) affected: TradingGuard.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation two state variables are not initialized with a default value, which can lead to unintended behavior during contract execution.	Line 14 & 15: buyConfig and sellConfig	We highly recommend initializing state variables with a default value to prevent the event of having uninitialized state variables at any time of contract execution.

5.1.4 Weak randomness

Severity: LOW
Code: SWC-116
Status: ACKNOWLEDGED
File(s) affected: MemeWarsPresenter.sol

Attack / Description	Code Snippet	Result/Recommendation
Generation of randomness with the block timestamp can be affected by the miners and can be predicted by some degree.	Line 551: randomWarTeamExt()	We recommend to use VFR randomness if a stronger randomness is required by your use case.

INFORMATIONAL ISSUES

5.1.5 Public functions should be declared external

Severity: INFORMATIONAL
Code: NA
Status: ACKNOWLEDGED
File(s) affected: Adminable.sol, TradingGuard.sol, MemeWarsToken.sol, MemeWarsPresenter.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several functions are declared as public where they could be external. For public functions Solidity immediately copies array arguments to memory, while external functions can read directly from calldata. Because memory allocation is expensive, the gas consumption of public functions is higher.	Adminable.sol Line 9-11: setAdmin() TradingGuard.sol Line 23-32: updateTradingConfig() MemeWarsToken.sol Line 11-13: __MemeWarsToken_init() MemeWarsPresenter.sol Line 126-151: __MemeWarsPresenter_init() Line 175-236: receiveTokens() Line 325-327: totalPendingTax()	We recommend declaring functions as external if they are not used internally. This leads to lower gas consumption and better code readability.

5.1.6 Variables using too many digits
Severity: INFORMATIONAL
Code: NA
Status: ACKNOWLEDGED
File(s) affected: MemeWarsToken.sol, MemeWarsPresenter.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation are hard to read literals with too many digits. The use of hard to read literals can be error-prone.	MemeWarsToken.sol Line 12: __ERC20Token_init(MemeWars,MWAR,18,250000000) MemeWarsPresenter.sol Line 379: cycle_.productionAmount = 10000000 * 10 ** (IERC20Metadata(token).decimals())	We recommend using scientific notation or underscores to increase code readability. For example: 250e6 or 250_000_000 is easier to read than 250000000 See: https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals

5.1.7 A floating pragma is set
Severity: INFORMATIONAL
Code: SWC-103
Status: ACKNOWLEDGED
File(s) affected: CoinDexTools.sol

Attack / Description	Code Snippet	Result/Recommendation
The current pragma Solidity directive is "^0.8.4". It is recommended to specify a	Line 1: `pragma solidity ^0.8.4;`	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did

fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.		not foresee. i.e. Pragma solidity 0.8.4 See SWC-103: https://swcregistry.io/docs/SWC-103

5.1.8 Check for boolean equality
Severity: INFORMATIONAL
Code: NA
Status: ACKNOWLEDGED
File(s) affected: AntiWhale.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several require checks are using a comparison to a Boolean constant. This leads to unnecessary gas consumption.	Line 16: require(bool,string)(antiWhaleActivated == false,Already activated) Line 24: require(bool,string)(antiWhaleActivated == true,Already activated) Line 56: antiWhaleActivated == false	It is recommended to remove the equality check to the Boolean constant and use the value itself.

5.2. SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	X
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✅
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✅
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✅
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✅
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	X
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✅
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✅
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✅
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✅
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✅
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✅
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✅
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✅
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✅

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	X
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✅
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✅
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅

6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the November 22, 2021.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the functions. During the audit, no critical issues were found after the manual and automated security testing and the claims been successfully verified. Please check the low and informational issues and get back to your auditor.

7. Deployed Smart Contract

VERIFIED

Token:
- proxy address: https://bscscan.com/address/0x9f28455a82BAA6B4923A5e2d7624aAf574182585
- implementation address: https://bscscan.com/address/0xf2F0362D88c3226D9ca8F036da382b09E267C6fA

Presenter:
- proxy address: https://bscscan.com/address/0x5773fe9A3b6175EE8B0F9f1c0c27a5ECd8424773
- implementation address: https://bscscan.com/address/0xd718A263bE4e54D37ba2e0Ef44C399DA0D18acDa