MARSWAP

SECURITY AUDIT REPORT FOR

BAD IDEA AI - (BAD)

0x32b86b99441480a7e5bd3a26c124ec2373e3f015

MARSWAP Audit provided by Fintech Global Services

Audit Tools used include

Manticore, Visual Code, Mythril, and Remix

Table of Contents

MARSWAP
SAFETY & SECURITY

SUMMARY OF AUDIT
DETAILS OF AUDITED PROJECT

Audited Project:	BAD IDEA AI – (BAD)
Source Code:	https://etherscan.io/token/0x32b86b99441480a7e5bd3a26c124ec2373e3f015#code (verified)
Solidity File:	BADIdeaAI.sol
Security Audit Date:	Aug. 28 - 2023
Revisions:	Initial Audit Aug.28.2023
Auditing Methods:	Automatic review + Manual review

AUDITING METHODS AND COVERING SECTORS

Evaluation objective for the security audit:

- Quality of smart contracts code
- Issues and vulnerabilities with security
- Documentation, project specifics and commenting on smart contract
- Correctness of specifications regarding the use-case
- Following the best practices on smart contract

a little bit of body text

Audit covers these sectors of smart contract for possible vulnerabilities, issues and recommendations for better practices in case of severe or medium issues:

- Dependence Transaction Order
- Single and Cross-Function Reentrancy
- Time Dependency
- Integer Overflow
- Integer Underflow
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Number rounding errors
- Insufficient gas issues
- Logical oversights
- Access control
- Centralization of power
- Logic-Specification
- Contradiction
- Functionality duplication

- Malicious contract behaviour and abusable functions
- Possible DoS vulnerabilities

The code review conducted for this audit follows the following structure:

1. Review of the specifications, documentation and commenting provided by the project owners regarding the functionality of the smart contract

2. Automated analysis of the smart contract followed by manual, line-by-line analysis of the smart contract
3. Assessment of smart contract's correctness regarding the documentation and commenting compared to functionality
4. Assessment of following the best practices
5. Recommendations for better practices in case severe or medium vulnerabilities

SMART CONTRACT DETAILS

Contract Address:	0x32b86b99441480a7e5bd3a26c124ec2373e3f015 (verified)
Blockchain:	Ethereum
Language Used:	Solidity
Compiler Version:	v0.8.13+commit.abaa5c0e
Etherscan Verification:	Unknown (verified)
Type of Smart Contract:	ERC20 Token – Standard Token
Libraries Used:	
Optimization Enabled:	Yes with 200 runs

Number of Interfaces: 2
Number of Contracts: 4
Solidity Versions: ^0.8.0
Total Lines: 516
Sell Tax: (0% Contract) 0%
Buy Tax: (0% Contract) 0%
Adjustable Taxes: NO
Total Supply: 831,041,059,897,327.3110117
Circulating Supply: 64.37%
Contract is Proxy: NO
Blacklist Functions: NO
Can Mint: NO
Pausable: NO
Can Limit TX amount: No

MARSWAP
SAFETY & SECURITY

SUMMARY OF AUDIT RESULTS

Marswap Security Scope smart contract audit for BAD IDEA AI – (BAD) is marked as PASSED result without severe issues on the contract logic and functions of the contract. Review of the project and description of the projects use-case as an digital asset and the contract itself follows the line of good practices. Contract is very well commented and gives the proper understanding for third parties. There are no severe- or lower level vulnerability findings in the contract.

CHANGEABLE VARIABLES AND SIDE NOTES

*Contract is not renounced
*Contract follows the OpenZeppeling standard without any modifications done

SEVERITY OF RISKS AND VULNERABILITIES

LOW-SEVERITY	MEDIUM-SEVERITY	HIGH-SEVERITY
0	0	0

REPORTED VULNERABILITIES, ISSUES AND INFORMATIONAL NOTES

LEVEL OF SEVERITY	Description	FILE	CODELINES AFFECTED
INFORMATIONAL	A floating pragma is set.	BADIdeaAI	L: 7 C: 0

FINDINGS IN-DEPTH

A FLOATING PRAGMA IS SET

The current pragma Solidity directive is ^0.8.0.
It is recommended to specify a fixed and locked compiler version to ensure that the bytecode produced does not vary between the builds. This is especially important if you rely on bytecode-level verification of the code.

SUGGESTION FOR FUTURE REFERENCE

Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively. As the severity of the finding, there are no adjustments needed and the contract has been successfully deployed and marked as informational.

DOCUMENTATION AND COMMENTING

The contract is very well commented on and gives the proper understanding of the contract for developers and other third parties for the understanding in a human-readable format. Overall, the contract follows OpenZeppelin standard contracts and is very clean without any editions made for the core contracts.

CORRECTNESS OF SPECIFICATIONS

Smart contract follows the functionality that is stated in the documentation and description of the contract. The use case is also in line with what is described about the project and no adjustments can be made by the owner except burn tokens.

FOLLOWING THE BEST PRACTICES

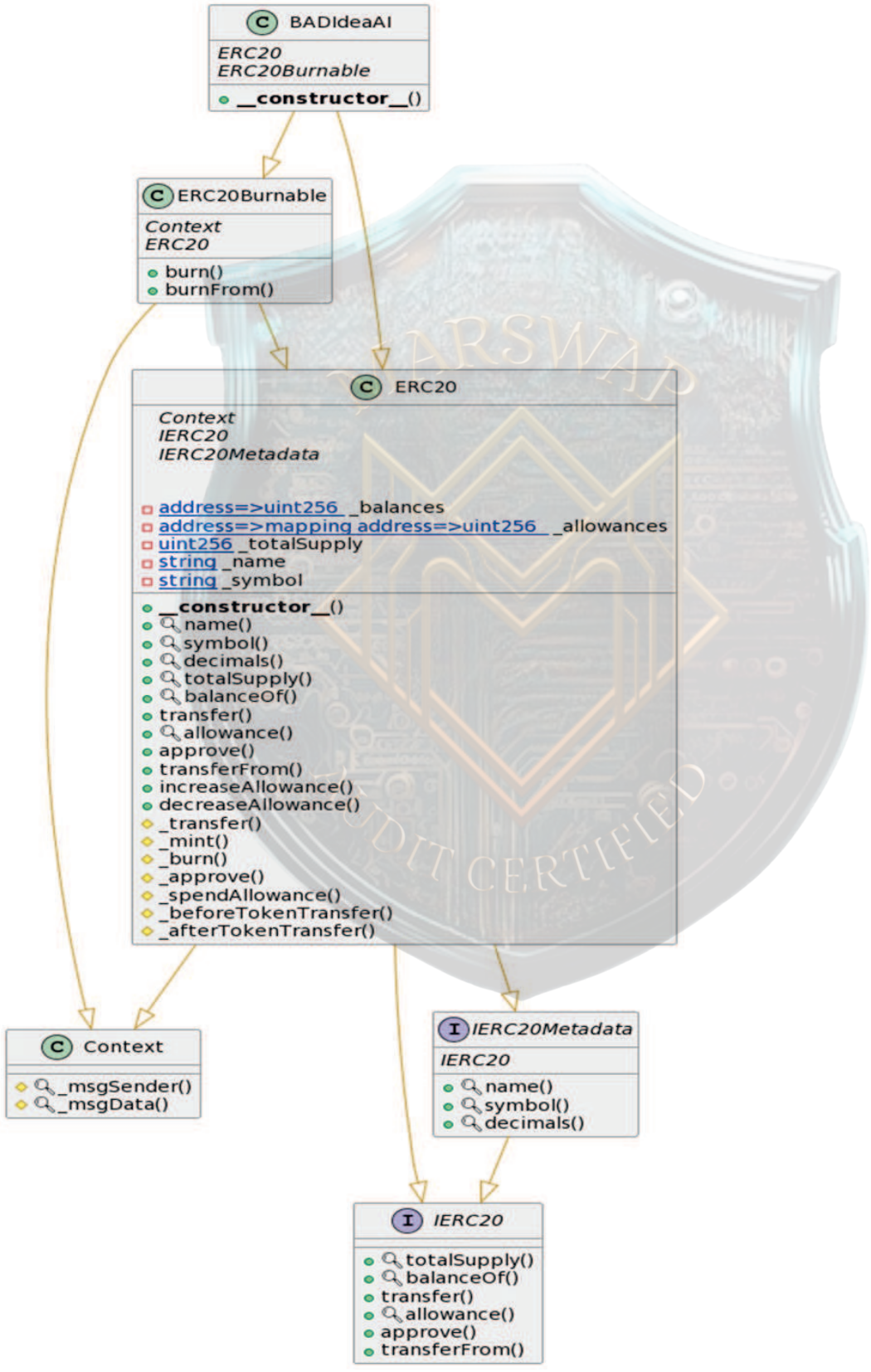
The contract follows the best practices from all parts and there are no concerns from the auditor of any malicious use of the contract's functions as the contract is a non-adjustable OpenZeppelin ERC20 Burnable token. Well-commended and clean code.

HISTORY OF REVISIONS, FUNCTIONS AND VARIABLES

FUNCTIONS AND SIGNATURES

Sighash | Function Signature
==========
39509351 => increaseAllowance(address,uint256)
119df25f => _msgSender()
8b49d47e => _msgData()
18160ddd => totalSupply()
70a08231 => balanceOf(address)
a9059cbb => transfer(address,uint256)
dd62ed3e => allowance(address,address)
095ea7b3 => approve(address,uint256)
23b872dd => transferFrom(address,address,uint256)
06fdde03 => name()
95d89b41 => symbol()
313ce567 => decimals()
a457c2d7 => decreaseAllowance(address,uint256)
30e0789e => _transfer(address,address,uint256)
4e6ec247 => _mint(address,uint256)
6161eb18 => _burn(address,uint256)
104e81ff => _approve(address,address,uint256)
1532335e => _spendAllowance(address,address,uint256)
cad3be83 => _beforeTokenTransfer(address,address,uint256)
8f811a1c => _afterTokenTransfer(address,address,uint256)
42966c68 => burn(uint256)
79cc6790 => burnFrom(address,uint256)

SMART CONTRACT UML



HISTORY OF REVISIONS

Initial audit was performed August. 28-2023 and no need for further revisions of the smart contract audit.

Team has been informed of a clean audit result and the smart contract follows all the golden standards without actual vulnerabilities. Notifications are informative for future reference.