ASSURE DEFI®
THE VERIFICATION GOLD STANDARD

Security Assessment:

FLEXIMINE TOKEN

FLEXIMINE

March 16, 2024

- Audit Status: **Pass**
- Audit Edition: **Advance**

VERIFIED BY ASSURE DEFI®
INTEGRITY • TRUST • CREDIBILITY

Risk Analysis

Classifications of Manual Risk Results

Classification	Description
🔴 Critical	Danger or Potential Problems.
🟠 High	Be Careful or Fail test.
🟢 Low	Pass, Not-Detected or Safe Item.
ℹ️ Informational	Function Detected

Manual Code Review Risk Results

Contract Privilege	Description
🟢 Buy Tax	0%
🟢 Sale Tax	0%
🟢 Cannot Buy	Pass
🟢 Cannot Sale	Pass
🟢 Max Tax	0%
ℹ️ Modify Tax	Yes
🟢 Fee Check	Pass
🟢 Is Honeypot?	Not Detected
🟢 Trading Cooldown	Not Detected
🟢 Can Pause Trade?	Pass
🟢 Pause Transfer?	Not-Detected
🟢 Max Tx?	Pass
🟢 Is Anti Whale?	Not-Detected
🟢 Is Anti Bot?	Not Detected

Contract Privilege	Description
🟢 Is Blacklist?	Not-Detected
🟢 Blacklist Check	Pass
🟢 is Whitelist?	Not-Detected
🟢 Can Mint?	Pass
🟢 Is Proxy?	Not Detected
🟢 Can Take Ownership?	Not Detected
🟢 Hidden Owner?	Not Detected
ℹ Owner	0x02a85195C4A16321Eb52399608A80e61f801aC2A
🟢 Self Destruct?	Not Detected
🟢 External Call?	Not-Detected
🟢 Other?	Not Detected
🟢 Holders	1
🟢 Auditor Confidence	High
🟡 KYC Present	No
🟡 KYC URL	

The following quick summary it's added to the project overview; however, there are more details about the audit and its results. Please read every detail.

Project Overview

Token Summary

Parameter	Result
Address	0x0cC0E8286fC023BC39380232290b5E6B8Cb4510c
Name	FLEXIMINE
Token Tracker	FLEXIMINE (FXM)
Decimals	18
Supply	4,000,000,000
Platform	BNBCHAIN
compiler	v0.6.12+commit.27d51765
Contract Name	Token
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/ address/0x0cc0e8286fc023bc39380232290b5e6b8cb4510c#code
Payment Tx	Corporate

Main Contract Assessed
Contract Name

Name	Contract	Live
FLEXIMINE	0x0cC0E8286fC023BC39380232290b5E6B8Cb4510c	Yes

TestNet Contract Assessed
Contract Name

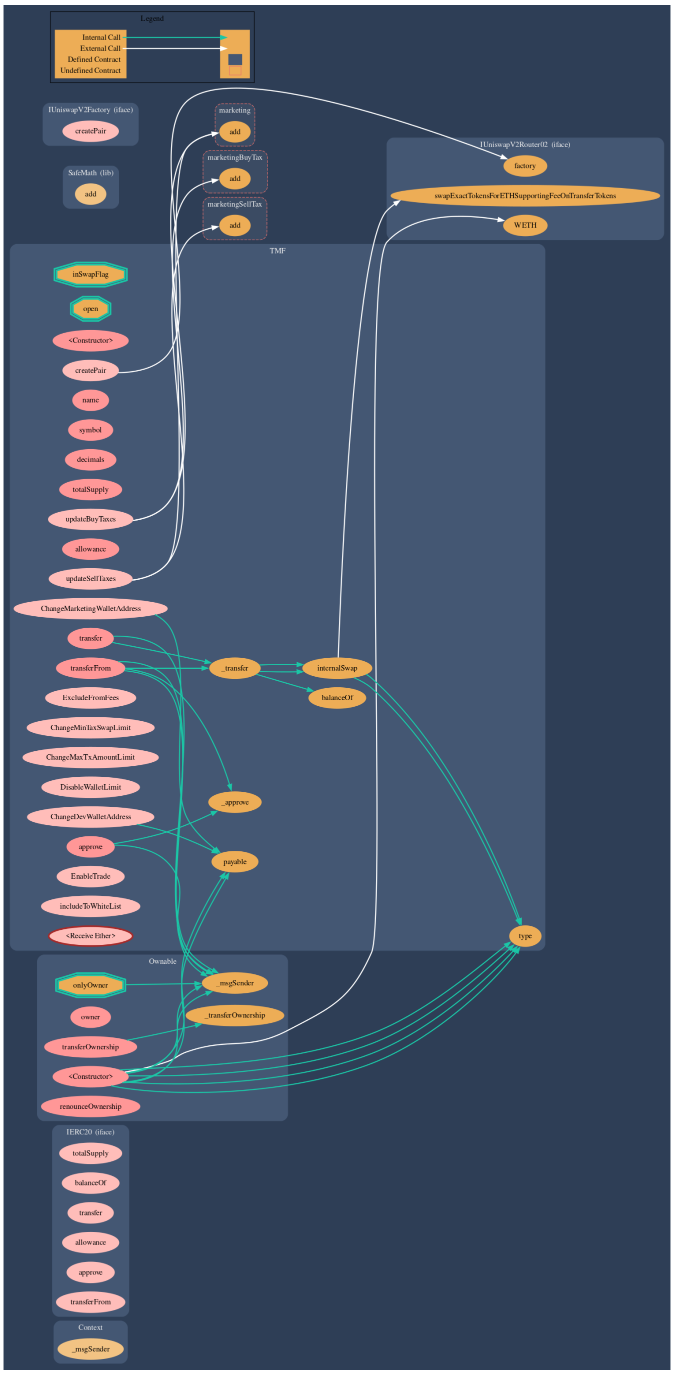
Name	Contract	Live
FLEXIMINE	0x39e83eB8be72F20d685AeA2a49bc947A70523c5C	Yes

Solidity Code Provided

SolID	File Sha-1	FileName
FXM	668b9d77251a99d557e4f94ee147b7ae0a5ec57f	FXM.sol
FXM		
FXM		
FXM		
FXM		
FXM		

Call Graph

The contract for FLEXIMINE has the following call graph structure.



Smart Contract Vulnerability Checks

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	FXM.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	FXM.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	FXM.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	FXM.sol	L: 57 C: 11
SWC-104	Pass	Unchecked Call Return Value.	FXM.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	FXM.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	FXM.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	FXM.sol	L: 0 C: 0
SWC-108	Pass	State variable visibility is not set..	FXM.sol	L: 0 C: 0
SWC-109	Pass	Uninitialized Storage Pointer.	FXM.sol	L: 0 C: 0
SWC-110	Medium	Assert Violation.	FXM.sol	L: 658 C: 25
SWC-111	Pass	Use of Deprecated Solidity Functions.	FXM.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	FXM.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	FXM.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-114	Pass	Transaction Order Dependence.	FXM.sol	L: 0 C: 0
SWC-115	Pass	Authorization through tx.origin.	FXM.sol	L: 0 C: 0
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	FXM.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	FXM.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	FXM.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	FXM.sol	L: 0 C: 0
SWC-120	Pass	Potential use of block.number as source of randonmness.	FXM.sol	L: 0 C: 0
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	FXM.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	FXM.sol	L: 0 C: 0
SWC-123	Pass	Requirement Violation.	FXM.sol	L: 0 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	FXM.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	FXM.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	FXM.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	FXM.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	FXM.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	FXM.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U +202E).	FXM.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	FXM.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	FXM.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	FXM.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	FXM.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	FXM.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	FXM.sol	L: 0 C: 0

We scan the contract for additional security issues using MYTHX and industry-standard security scanning tools.

Smart Contract Vulnerability Details

SWC-103 - Floating Pragma.

CWE-664: Improper Control of a Resource Through its Lifetime.

References:

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with 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.

Remediation:

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

References:

Ethereum Smart Contract Best Practices - Lock pragmas to specific compiler version.

Smart Contract Vulnerability Details

SWC-110 - Assert Violation

CWE-670: Always-Incorrect Control Flow Implementation

Description:

The Solidity assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement. A reachable assertion can mean one of two things:

A bug exists in the contract that allows it to enter an invalid state;
The assert statement is used incorrectly, e.g. to validate inputs.

Remediation:

Consider whether the condition checked in the assert() is actually an invariant. If not, replace the assert() statement with a require() statement.If the exception is indeed caused by unexpected behaviour of the code, fix the underlying bug(s) that allow the assertion to be violated.

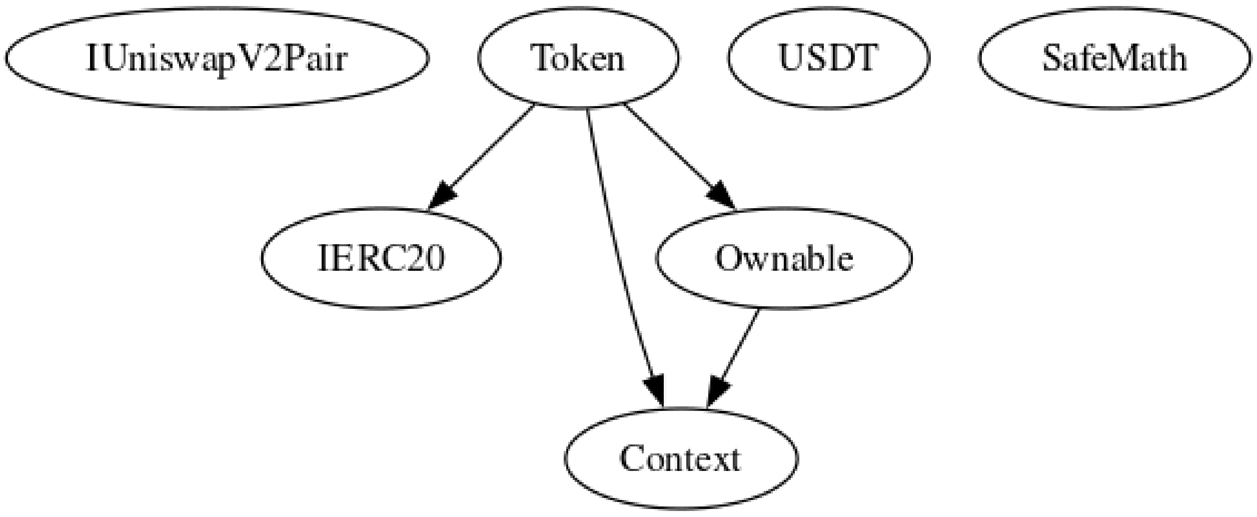
References:

The use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM.

Inheritance

The contract for FLEXIMINE has the following inheritance structure.

The Project has a Total Supply of 4,000,000,000



Privileged Functions (onlyOwner)

Please Note if the contract is Renounced none of this functions can be executed.

Function Name	Parameters	Visibility
renounceOwnership		Public
transferOwnership	address newOwner	Public

FXM-01 | Potential Sandwich Attacks.

Category	Severity	Location	Status
Security	● Low	FXM.sol:	Detected

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction being attacked) a transaction to sell the asset. The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- swapExactTokensForETHSupportingFeeOnTransferTokens()
- addLiquidityETH()

Remediation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

Referrences:

What Are Sandwich Attacks in DeFi — and How Can You Avoid Them?.

FXM-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	🟢 Low	FXM.sol: L: 380 C: 14	Detected

Description

The given input is missing the check for the non-zero address.

The given input is missing the check for the onlyOwners need to be revisited for require..

Remediation

We advise the client to add the check for the passed-in values to prevent unexpected errors as below:

```
    ...
     require(receiver != address(0), "Receiver is the zero address");
    ...
    ...
    require(value X limitation, "Your not able to do this function");
   ...
```

We also recommend customer to review the following function that is missing a required validation. onlyOwners need to be revisited for require..

FXM-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	🟢 Low	FXM.sol: L: 368 C: 14	Detected

Description

Detected missing events for critical arithmetic parameters. There are functions that have no event emitted, so it is difficult to track off-chain changes.The linked code does not create an event for the transfer.

Remediation

Emit an event for critical parameter changes. It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

FXM-07 | State Variables could be Declared Constant.

Category	Severity	Location	Status
Coding Style	🟢 Low	FXM.sol: L: 433-436	Detected

Description

Constant state variables should be declared constant to save gas.

```
_name
_symbol
_decimal
```

Remediation

Add the constant attribute to state variables that never changes.

https://docs.soliditylang.org/en/latest/contracts.html#constant-state-variables

FXM-14 | Unnecessary Use Of SafeMath

Category	Severity	Location	Status
Logical Issue	● Medium	FXM.sol: L: 0 C: 0	Detected

Description

The SafeMath library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations
will automatically revert in case of integer overflow or underflow.
library SafeMath {
An implementation of SafeMath library is found.
using SafeMath for uint256;
SafeMath library is used for uint256 type in contract.

Remediation

We advise removing the usage of SafeMath library and using the built-in arithmetic operations provided by the
Solidity programming language

Project Action

FXM-19 | Centralization Privileges of.

Category	Severity	Location	Status
	🔴 Medium	FXM.sol: L: 393 C: 14,L: 385 C: 14,L: 341 C: 14,L: 306 C: 14,L: 299 C: 14,L: 269 C: 14	Detected

Description

Centralized Privileges are found on the following functions.

Remediation

undefined

Project Action

Technical Findings Summary

Classification of Risk

Severity	Description
🔴 Critical	Risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟠 High	Risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟡 Medium	Risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform
🟢 Low	Risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the Project, but they may be less efficient than other solutions.
ℹ️ Informational	Errors are often recommended to improve the code's style or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

Findings

Severity	Found	Pending	Resolved
🔴 Critical	1	0	0
🟠 High	0	0	0
🟡 Medium	2	2	0
🟢 Low	3	2	0
ℹ️ Informational	0	0	0
Total	6	4	0

Social Media Checks

Social Media	URL	Result
Twitter	https://twitter.com/layersyncai	Pass
Other	https://medium.com/@layersyncai	Pass
Website	https://layersyncai.io	Pass
Telegram	https://t.me/layersyncai	Pass

We recommend to have 3 or more social media sources including a completed working websites.

Social Media Information Notes:

Auditor Notes: undefined

Project Owner Notes:

ASSURE DEFI | OFFICIAL PARTNER

Assessment Results

Score Results

Review	Score
Overall Score	86/100
Auditor Score	85/100

Review by Section	Score
Manual Scan Score	40
SWC Scan Score	33
Advance Check Score	13

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project most pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below.

Audit Passed

Assessment Results

Important Notes:

- Reentrancy: Implement checks-effects-interactions pattern in buyWithEth.ı
- DoS with Unexpected Revert: Consider using a pull-over-push pattern for Ether transfers.ı
- Centralization Risks: Review and potentially decentralize onlyOwner functions.ı
- Floating Pragma: Lock pragma to a specific compiler version.ı
- Outdated Compiler: Update to the latest Solidity version with all security patches.ı
- Oracle Reliance: Implement checks for oracle reliability and manipulation resistance.ı
- ERC20 Compliance: Ensure transfer and transferFrom return a boolean.ı
- Public Functions: Review access controls, especially for the burn function.ı
- Slippage Control: Add slippage protection or max price in buyWithEth.ı
- Hardcoded Addresses: Consider making USDT and Uniswap pair addresses configurable.ı
- Event Emission: Emit events for all state-changing external functions.ı

- Input Validation: Add require statements to validate function inputs.ı
- Price Manipulation: Protect calculateUsdtFromEth against potential manipulation.ı
- Fallback Function: Add limits or conditions to fallback and receive functions.ı
- Gas Limitations: Optimize or limit the size of the statistics array to prevent out-of-gas errors.ı
- Commented Code: Remove or clarify the purpose of commented code.ı
- Circuit Breaker: Implement a circuit breaker/emergency stop pattern to pause contract functions if needed.ı
- Tokenomics: Analyze and document the impact of buyWithUsdt and buyWithEth on token price and supply.ı
- Code Optimization: Review for any unnecessary code or gas inefficiencies.ı
- Function Visibility: Specify visibility for functions, especially where default is not appropriate.ı
- Contract Size: Check the contract size to avoid deployment issues related to maximum contract size limits.ı
- Testing: Ensure comprehensive testing including unit tests, integration tests, and test coverage analysis.ı
- Formal Verification: Consider formal verification of the contract's critical logic.ı
- External Contract Interactions: Audit and monitor the

contracts that are interacted with, like USDT and Uniswap V2 pair.ı

- Code Documentation: Improve code comments for better clarity and maintainability.ı
- Historical Price Data: Verify how historical price data is used and ensure it cannot be exploited.ı
- Contract Modularity: Break down the contract into smaller modules for easier management and auditing.ı
- Liquidity Considerations: Assess the effects of large buys or sells on liquidity and price impact.ı
- Admin Key Security: Ensure safe storage and management of admin keys.ı
- Concluding, the contract should undergo a thorough review and testing process to address these concerns before being considered secure for deployment.

Auditor Score =85
Audit Passed

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that actagainst the nature of decentralization, such as explicit ownership or specialized access roles incombination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimalEVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on howblock.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functionsbeing invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that mayresult in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to makethe codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code,such as a constructor assignment imposing different require statements on the input variables than a setterfunction.

Coding Best Practices

ERC 20 Conding Standards are a set of rules that each developer should follow to ensure the code meet a set of creterias and is readable by all the developers.

Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocation for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided ‘as is, and Assure Defi is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.

ASSURE DEFI™
THE VERIFICATION **GOLD STANDARD**