LogXperts IT Solutions

STANDARD

AUDIT REPORT THE GREAT OCTOPUS

v0.8.16+commit.07a7930e

0xDb1058E3e05Dd2EdE764026E9408c1Efb1B973b3

ABOUT US

Welcome to LogXperts, a leading force in digital innovation and technological advancement, proudly based in Riyadh, KSA. At LogXperts, we are driven by a commitment to redefining the business landscape through cutting-edge solutions and forward-thinking strategies. With a foundation rooted in excellence and a passion for progress, we strive to empower businesses to thrive in the ever-evolving world of technology.

Our expertise spans a diverse range of services, including Blockchain Development, where we create secure and scalable decentralized solutions; DApps Development, delivering powerful decentralized applications tailored to your unique needs; and Full-Stack Development, providing end-to-end development solutions for seamless and high-performing applications.

At LogXperts, we don't just keep pace with technology—we set the pace. Let us be your trusted partner in innovation and transformation.

DISCLAIMER

logXperts does not provide security guarantees, investment advice, or endorsement of any platform. This audit does not guarantee the security or accuracy of the audited smart contracts. Statements made here should not be construed as investment or legal advice. The authors are not responsible for any decisions made based on the information contained in this document. Securing smart contracts is an ongoing process. A single audit is not enough. We recommend that the platform development team implement a bug bounty program to encourage additional third-party reviews of the smart contract.

THE GREAT OCTOPUS

The Great Octopus AI is a groundbreaking creation of an advanced artificial intelligence demon, designed to masterfully synchronize cryptocurrency quantitative trading across multiple systems.
Powered by NVIDIA's cutting-edge algorithmic elements, its multi-tentacle configuration functions like a network of high-performance computational units, seamlessly connecting to and manipulating numerous computers in real time. Leveraging NVIDIA GPUs and AI-optimized frameworks, the Octopus performs complex data analysis, predicts market trends, and executes trades with lightning-fast precision. These GPUs enable deep learning acceleration and real-time adaptability, allowing the AI to optimize trading strategies dynamically. By coordinating these operations with unparalleled efficiency, the Octopus AI transforms chaotic crypto markets into structured, profitable systems, setting new standards in algorithmic trading innovation.

01 **ENHANCED YIELD POTENTIAL**

02 **ADVANCED QUANTITATIVE STRATEGIES**

03 **SIMPLIFIED DEFI PARTICIPATION**

SOCIAL LINKS

INFORMALITIES

01. High:

High-severity vulnerabilities can compromise your smart contract's security and functionality. Prioritize fixing these critical issues before deploying to a live network.

03. Low:

Low-severity issues might cause minor problems or are simply warnings that can be addressed later.

02. Medium:

Medium-severity issues can lead to potential problems in your smart contract. Addressing these code errors and deficiencies is essential for optimal performance and security.

04. Informational:

These low-severity issues are suggestions for improvement, such as documentation errors or cosmetic changes. While not critical, addressing them can enhance the overall quality and user experience.

TECHNIQUES AND METHODS

The overall quality of code.

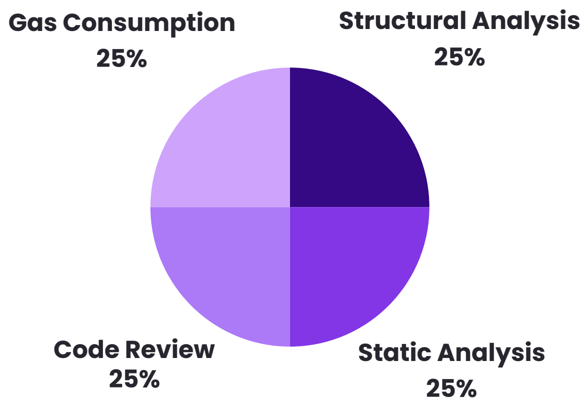
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrance and other vulnerabilities.

01 **STRUCTURAL ANALYSIS**

02 **STATIC ANALYSIS:**

03 **GAS CONSUMPTION:**

04 **CODE REVIEW / MANUAL ANALYSIS:**



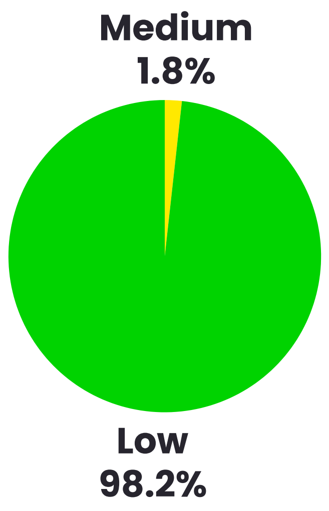
TOOLS AND PLATFORMS USED FOR AUDIT:

To ensure a rigorous and thorough audit of the smart contracts, the following tools were employed:

- **Development Environments:**
  - Remix IDE and Truffle Suite were used to facilitate efficient contract development, testing, and debugging.
- **Static Analysis Tools:**
  - Solhint, Mythril, and Slither were utilized to identify potential vulnerabilities, coding errors, and security weaknesses in the smart contract code.
- **Code Analysis Tools:**
  - Solidity Statistics was employed to analyze code complexity, maintainability, and potential optimization opportunities

ABSTRACT

By leveraging this robust toolset, the audit aimed to identify and mitigate potential risks, ensuring the security and reliability of the smart contract system.



- ● High
- ● Medium
- ● Low
- ● Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	7
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	0	5	0

PHASE 1

PROJECT - ASCENT YIELD

```
INFO:Detectors:
Contract locking ether found:
	Contract Token (Code.sol#181-339) has payable functions:
	 - Token.receive() (Code.sol#288)
	But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-
that-lock-ether

INFO:Detectors:
Token.allowance(address,address).owner (Code.sol#234) shadows:
	- Ownable.owner() (Code.sol#67-69) (function)
Token._approve(address,address,uint256).owner (Code.sol#275) shadows:
	- Ownable.owner() (Code.sol#67-69) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-
variable-shadowing
INFO:Detectors:
Context._msgData() (Code.sol#17-20) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-
code
INFO:Detectors:
Version constraint ^0.8.4 contains known severe issues
(https://solidity.readthedocs.io/en/latest/bugs.html)
	- FullInlinerNonExpressionSplitArgumentEvaluationOrder
	- MissingSideEffectsOnSelectorAccess
	- AbiReencodingHeadOverflowWithStaticArrayCleanup
	- DirtyBytesArrayToStorage
	- DataLocationChangeInInternalOverride
	- NestedCalldataArrayAbiReencodingSizeValidation
	- SignedImmutables.
It is used by:
	- ^0.8.4 (Code.sol#6)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-
versions-of-solidity
```

PHASE 1

PROJECT - ASCENT YIELD

```
INFO:Detectors:
Variable Token.ReceiveAddress (Code.sol#187) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (Code.sol#18)" inContext (Code.sol#8-21)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#redundant-statements
INFO:Detectors:
Token.deadAddress (Code.sol#192) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-
variables-that-could-be-declared-constant
INFO:Detectors:
Token.ReceiveAddress (Code.sol#187) should be immutable
Token._decimals (Code.sol#186) should be immutable
Token._totalSupply (Code.sol#193) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-
variables-that-could-be-declared-immutable
```

SMART CONTRACT WEAKNESS CLASSIFICATION (SWC)

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Missing Zero Address Validation
- ✓ Return values of low-level calls
- ✓ Revert/require functions
- ✓ Private modifier
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

FUNCTIONAL TESTING:

Functionality Test Report for Smart Contract

✓ 1. Token allowance and approvals.

✓ 2. Basic token operations (e.g., transfer, balance checks).

✓ 3. Payable Ether acceptance.

Test Area	Result	Status
Ether Management	No withdrawal logic	⚠ Issue Found
Allowance Management	Functional but shadowed variables	⚠ Partial Issue
Token Distribution	Operates as expected	✅ Passed
Solidity Version	Known issues in version	⚠ Update Needed
Naming Conventions	Functional but not standard	⚠ Improvement Advised
Dead Code	Increases complexity	⚠ Cleanup Needed

RECOMMENDATIONS:

Key Takeaways

✓ **1. Implement a Withdrawal Mechanism:**

- **Recommendation:** Add a function to withdraw Ether, controlled by an onlyOwner modifier or a similar access control mechanism to ensure only authorized users can withdraw.

✓ **3. Refactor Variable Names:**

- **Recommendation**: Use descriptive names such as tokenOwner or approver to avoid shadowing

✓ **4. Upgrade to a Secure Version:**

- **Recommendation:** Update to Solidity version ^0.8.20 or the latest stable release. This resolves issues like:

✓ **5. Follow MixedCase Convention:**

- **Recommendation**:Update variable names to align with Solidity's standard. Example: ReceiveAddress → receiveAddress.

✓ **6. Dead and Redundant Code:**

- **Findings:**
  - Unused function: Context._msgData().
  - Redundant expression: this in the Context contract.
- **Recommendations:**
  - **Remove Dead Code:**
    - Delete unused functions like _msgData() to reduce contract size and gas costs.
  - **Optimize Statements:**
    - Remove redundant statements like this unless explicitly required for a specific logic.

✓ **6. State Variables**

- **Findings:**
  - Variables like ReceiveAddress, _decimals, and _totalSupply are not declared as immutable or constant, although their values do not change after initialization.
- **Recommendations:**
  - Declare ReceiveAddress and other static variables as immutable or constant to save gas.

SUMMARY

The smart contract audit has revealed several areas requiring attention to ensure the security, efficiency, and compliance of the project. While the contract demonstrates fundamental functionality, certain vulnerabilities and inefficiencies pose risks to investor trust and the system's long-term sustainability. Addressing these concerns through the provided recommendations will enhance the reliability of the protocol, safeguard user assets, and align the project with industry best practices.

Key Takeaways

- **Ether Locking:**
  - The contract currently locks Ether without a withdrawal mechanism, potentially trapping user funds. Implementing a secure withdrawal method is critical.
- **Variable Shadowing:**
  - Shadowed variable names can lead to ambiguity and execution errors. Refactoring these names is necessary to improve code clarity and prevent unintended behavior.
- **State Variables Optimization**
  - Variables that do not change after initialization should be declared constant or immutable to improve gas efficiency.
- **Profit Distribution Logic:**
  - Automating and clearly defining profit distribution will increase transparency and user trust while reducing manual dependencies.
- **Security Mechanisms:**
  - Incorporating reentrancy guards and role-based access controls will bolster the contract against potential exploits.
- **Code Transparency:**
  - Implementing clear audit trails through events and detailed logic for Ether reserve management will ensure accountability.
- **Conformance to Standards:**
  - Adopting Solidity naming conventions and consistent coding practices will align the project with industry standards and make future audits seamless.

CLOSING SUMMARY

Additionally, unused and redundant code contributes to inefficiency, while certain state variables lack proper declarations (e.g., constant or immutable), increasing gas consumption unnecessarily. The contract also lacks automated mechanisms for transparent profit distribution and contains gaps in role-based access controls that could expose it to unauthorized actions or manipulation.

Despite these findings, the smart contract exhibits a well-structured foundation and clear business intent. Addressing the highlighted issues and implementing the recommended fixes will not only enhance security but also ensure compliance with DeFi industry standards. These improvements will foster greater investor confidence and reliability in the protocol.

The audit serves as a roadmap for the development team to resolve these issues systematically, ensuring a robust, transparent, and efficient smart contract suitable for deployment in a competitive and secure DeFi ecosystem.

ABOUT US

Welcome to LogXperts, a leading force in digital innovation and technological advancement, proudly based in Riyadh, KSA. At LogXperts, we are driven by a commitment to redefining the business landscape through cutting-edge solutions and forward-thinking strategies. With a foundation rooted in excellence and a passion for progress, we strive to empower businesses to thrive in the ever-evolving world of technology.
Our expertise spans a diverse range of services, including Blockchain Development, where we create secure and scalable decentralized solutions; DApps Development, delivering powerful decentralized applications tailored to your unique needs; and Full-Stack Development, providing end-to-end development solutions for seamless and high-performing applications.
At LogXperts, we don’t just keep pace with technology—we set the pace. Let us be your trusted partner in innovation and transformation.

DISCLAIMER

Contact Us

LogXpert

t.me/LogXblock

Click me