VIBRANIUM AUDITS

Security Assessment

FireFlies Token

Vibranium Audits Verified on Oct 30th, 2024

Summary

Finding

DISCLAIMER

Vibranium audits verified on Nov 15th 2024

FireFlies

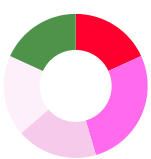
The security assessment was prepared by Vibranium audits , the leader in Web3 security

Executive Summary

TYPES	ECOSYSTEM	METHODS
Token	Ethereum / EVM	Formal Verification, Manual review

LANGUAGE	Codebase
Solidity	https://sepolia.etherscan.io/address/0xc510281b2f584fff3f6cd6398258346315b8b34c#code updated version : https://sepolia.etherscan.io/address/0xef1a779e40ad84c0cd0cf42112540104d228eb81#code

Vulnerability Summary



6	0	1	4	1	0
Total Findings	Critical	High	Medium	Minor	Informational

High	Resolved	The function does not verify if the contract has sufficient ETH and token balances before adding liquidity.
Medium	Resolved	Sending tokens to 0xdead does not remove them from total supply (_tTotal), creating potential misuse or ambiguity.
Medium	Resolved	The owner has excessive privileges, creating a single point of failure and reducing trust in the system.
Minimal	Resolved	Hardcoded tax values and redundant operations during transfers increase gas costs and reduce flexibility.
Medium	Resolved	No safeguards against sandwich attacks or price manipulation during liquidity provisioning.
Minor	Resolved	Redundant calculations in the _transfer function unnecessarily increase gas usage.

Codebase | **FireFlies**

Address

https://sepolia.etherscan.io/address/0xc510281b2f584fff3f6cd6398258346315b8b34c#code

updated version : https://sepolia.etherscan.io/address/0xef1a779e40ad84c0cd0cf42112540104d228eb81#code

Approach & Methods | FireFlies

This report has been prepared for FireFlies to identify potential issues and vulnerabilities in the source code of the project, including smart contracts and the token minting mechanism. A comprehensive examination has been conducted, utilizing Static Analysis and Manual Review techniques to ensure compliance with best security practices.

The auditing process pays special attention to the following considerations:

Testing the smart contracts against common and uncommon attack vectors such as reentrancy attacks, overflow vulnerabilities, and unauthorized access.

Assessing the codebase for compliance with industry best practices, ensuring code security, scalability, and maintainability.

Ensuring contract logic meets the specifications and requirements of the FireFlies project, specifically for token minting, access control, and transaction handling.

Cross-referencing contract structure and functionality against similar projects in the blockchain ecosystem to ensure adherence to standard practices.

Performing a line-by-line manual review of the entire codebase to identify any hidden or obscure vulnerabilities that automated tools may miss.

The security assessment identified vulnerabilities across a range of severity levels, from Critical to Informational. We recommend addressing these findings promptly to enhance the overall security of the FireFlies project. Key recommendations include:

Testing smart contracts against both common and uncommon attack vectors, ensuring robustness against real-world threats.

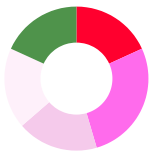
Implementing secure coding practices to enhance the quality and security of the codebase.

Expanding unit tests to cover all possible use cases and edge cases, especially for the token minting functions.

Improving code documentation and comments for better readability, particularly for critical functions involving token minting and transfers.

Ensuring transparency in privileged activities, such as the minting process, and implementing additional safeguards for authorization.

FireFlies | Finding



6	0	1	4	1	0
Total Findings	Critical	High	Medium	Minor	Informational

This report provides a detailed audit of the FireFlies project's codebase, focusing on the token contract . The objective of this audit is to uncover potential security vulnerabilities, optimize code efficiency, and identify areas for improvement to strengthen the project's security and reliability.
In total, six vulnerabilities were identified, categorized by severity as follows: 1 High , 4 Medium and 1 Minor.

ID	Title	Category	Severity	Status
FF-L1	Missing Balance Checks in openTrading	Validation Issues	High	Resolved
FF-L2	Improper Burn Mechanism	Supply Management	Medium	Resolved
FF-L3	Centralized Control and Trust Issues	Access Control	Medium	Resolved
FF-L4	Inefficient Taxation Logic	Gas Optimization	Medium	Resolved
FF-L5	Potential Token Price Manipulation	Market Manipulation Risks	Medium	Resolved
FF-L6	Gas Inefficiencies in _transfer	Gas Optimization	Minimal	Resolved

FF-L1 | Missing Balance Checks in openTrading

Category	Severity	Location	Status
Validation Issues	High	openTrading function	Resolved

Description

The openTrading function lacks essential validation to ensure that the contract holds sufficient ETH and tokens before attempting to add liquidity. This oversight can result in significant operational and financial problems:

1. Liquidity Addition Failure:
   - If the contract does not have enough tokens or ETH, the Uniswap addLiquidityETH function will revert. This failure prevents trading from being enabled and wastes gas fees.
2. Financial Loss:
   - If users or the owner assume the function will succeed but it fails, it can result in wasted deployment funds and additional costs for troubleshooting and retries.

Mitigation

1. Pre-Execution Balance Checks:
   - Add validation checks to ensure the contract has sufficient ETH and tokens before calling the addLiquidityETH function.
2. Explicit Error Messages:
   - Use descriptive require statements to provide clear error messages if balances are insufficient.

```
uint256 contractTokenBalance = balanceOf(address(this));
 uint256 contractETHBalance = address(this).balance;
 require(contractTokenBalance > 0, "Insufficient token balance in contract");
 require(contractETHBalance > 0, "Insufficient ETH balance in contract");
```

FF-L2 | Improper Burn Mechanism

Category	Severity	Location	Status
Supply Management	Medium	transfer function	Resolved

Description

The contract uses a burn mechanism that sends tokens to the hardcoded _burnAddress (0xdead). While this is a widely recognized convention for burning tokens, it has notable shortcomings:

1. Tokens Are Not Truly Removed from Supply:
   - The total supply (_tTotal) is not reduced when tokens are sent to _burnAddress. This creates an inaccurate representation of the circulating supply.
2. Potential Misuse:
   - If someone were to gain control of the private key for 0xdead (unlikely but theoretically possible), they could retrieve the tokens sent to that address, undermining the intent of burning.
3. Transparency Concerns:
   - The absence of a proper burn() function makes it harder for users to verify how many tokens are permanently removed from circulation, leading to trust issues.

Mitigation

Implement a Proper Burn Mechanism:

- Introduce a burn() function that reduces the total supply (_tTotal) directly and ensures transparency.

OR

- Use address(0) for Irrecoverable Token Removal:

Transfer tokens to address(0) for true burning since it is universally recognized as the null address and tokens sent there are irrecoverable.

FF-L3 | Centralized Control and Trust Issues

Category	Severity	Location	Status
Access Control	Medium	setExcludedFromFee updateReservePotAddress onlyOwner	Resolved

Description

The contract grants excessive privileges to the owner through the onlyOwner modifier, which can lead to trust and security concerns:

1. Owner Privileges:
   - The owner can exclude any address from fees (setExcludedFromFee).
   - The owner can update the _reservePotAddress, potentially redirecting funds to an arbitrary address.
2. Single Point of Failure:
   - If the owner account is compromised or misused, these powers can be exploited to harm the system or users. For example:
     - The owner could exclude themselves or malicious addresses from fees to manipulate trading.
     - The _reservePotAddress could be updated to an address controlled by an attacker, misappropriating tax funds.
3. Lack of Decentralization:
   - Excessive reliance on a single owner contradicts the decentralized ethos and can undermine trust among token holders and investors.

Mitigation

Introduce Multi-Signature Wallets:

- Use a multi-signature wallet for the owner account to reduce the risk of unauthorized changes. Multi-signature wallets require multiple parties to authorize critical actions, enhancing security.

FF-L4 | Inefficient Taxation Logic

Category	Severity	Location	Status
Gas Optimization	Medium	_buyTax and _sellTax	Resolved

Description

The taxation logic in the _transfer function is rigid and inefficient, leading to unnecessary gas costs and reduced flexibility:
Hardcoded Tax Rates:

- _buyTax and _sellTax are hardcoded as constants, preventing the owner or governance from adjusting tax rates to respond to market or community needs.

Mitigation

Making Tax Rates Configurable:
adding updateTaxRates(uint256 newBuyTax, uint256 newSellTax) external onlyOwner
to change Taxes

FF-L5 | Potential Token Price Manipulation

Category	Severity	Location	Status
Market Manipulation Risks	Medium	openTrading function	Resolved

Description

The contract is vulnerable to token price manipulation during liquidity provisioning in the openTrading function and subsequent interactions with the Uniswap liquidity pool.
Specifically:

1. Sandwich Attacks:
   - Malicious actors can front-run and back-run transactions that involve adding liquidity or large token transfers, manipulating the token price for profit.
2. Lack of Slippage Protection:
   - The addLiquidityETH function is called with amountTokenMin and amountETHMin set to 0, leaving the contract exposed to executing trades or adding liquidity at highly unfavorable rates if the token price is manipulated.

Mitigation

Add Slippage Protection:
Specify meaningful amountTokenMin and amountETHMin values when calling the addLiquidityETH function to ensure the transaction only executes at acceptable price ranges.

FF-L6 | Gas Inefficiencies in _transfer

Category	Severity	Location	Status
Gas Optimization	Minimal	_transfer function	Resolved

Description

The _transfer function has redundant calculations and operations that increase gas costs unnecessarily:

1. Redundant Tax Calculation:
   - The tax amount is calculated multiple times, which could be consolidated to save gas.
2. Separate Operations for Tax Handling:
   - Splitting the tax into burnAmount and reserveAmount and updating balances separately introduces additional computational overhead.
3. Inefficient Balance Updates:
   - Token balance adjustments for the sender and recipient could be optimized for better gas efficiency.

Mitigation

Consolidate Tax Calculations:

- Combine tax calculations into a single operation to avoid redundancy.

Batch Update Balances:

- Reduce the number of operations by batching balance updates for the sender, recipient, and tax destinations.

Streamline Burn and Reserve Logic:

- Minimize the number of operations when processing burnAmount and reserveAmount.

DISCLAIMER

VIBRANIUM AUDITS

This security assessment has been prepared by Vibranium Audits for the FireFlies project to identify potential vulnerabilities in the project's codebase. Vibranium Audits applied its expertise in Web3 and smart contract security to perform a detailed analysis of the project, utilizing a combination of manual code review, automated tools, and industry best practices.

This report represents Vibranium Audits' assessment as of the verification date (November 15th, 2024). The findings and recommendations provided here in reflect the state of the project at the time of the audit. As blockchain technology and associated security risks continuously evolve, we strongly recommend ongoing monitoring, upgrades, and reassessments to maintain a robust security posture.

Vibranium Audits assumes no responsibility for any losses, damages, or consequences, whether financial or otherwise, resulting from the use, misuse, or reliance on this report. This document is intended solely for informational purposes and should not be construed as financial, legal, or investment advice, nor as an endorsement or guarantee of the project.

Vibranium Audits Securing the Web3 World

Vibranium Audits is a blockchain security company that was founded in 2021 by professors from the University of Greenwich and cyber-security engineers from ITI Capital. As pioneers in the field, Vibranium Audits utilizes best-in-class Formal Verification and AI technology to secure and monitor blockchains, smart contracts, and Web3 apps.