Smart contract security audit FarmingServiceVault

v.1.0

Table of Contents

1.0 Introduction

1.1 Project engagement

During January of 2022, FarmingServiceVault engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. FarmingServiceVault provided CTDSec with access to their code repository and whitepaper.

1.2 Disclaimer

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the network's fast-paced and rapidly changing environment, we at CTDSec recommend that FarmingServiceVault team put in place a bug bounty program to encourage further and active analysis of the smart contract.

2.0 Coverage

2.1 Target Code and Revision

For this audit, we performed research, investigation, and review of the FarmingServiceVault contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code files are considered in-scope for the review:

Source:

https://etherscan.io/address/0x643239d5d7F05eD6D268c371aBF5eF694bc64E5f#code

2.2 Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

№	Issue description.	Checking status
1	Compiler warnings.	PASSED
2	Race conditions and Reentrancy. Cross-function race conditions.	PASSED
3	Possible delays in data delivery.	PASSED
4	Oracle calls.	PASSED
5	Front running.	LOW ISSUES
6	Timestamp dependence.	PASSED
7	Integer Overflow and Underflow.	PASSED
8	DoS with Revert.	PASSED
9	DoS with block gas limit.	PASSED
10	Methods execution permissions.	PASSED
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	PASSED
12	The impact of the exchange rate on the logic.	PASSED
13	Private user data leaks.	PASSED
14	Malicious Event log.	PASSED
15	Scoping and Declarations.	PASSED
16	Uninitialized storage pointers.	PASSED

17	Arithmetic accuracy.	PASSED
18	Design Logic.	PASSED
19	Cross-function race conditions.	PASSED
20	Safe Zeppelin module.	PASSED
21	Fallback function security.	PASSED
22	Overpowered functions / Owner privileges	MEDIUM ISSUES

3.0 Security Issues

3.1 High severity issues [0]

No high severity issues found.

3.2 Medium severity issues [1]

1. Centralization:

Owner has authority over the next functions:

Any compromise to the owner account may allow the hacker to take advantage of this and change the above significant states of the contract.

setCanSwap(): Owner can enable/disable swapping.

setTradingEnabled(): Owner can enable/disable trading.

setTreasuryWallet(): Owner can change the treasury wallet.

setMFCWallet(): Owner can change the marketing wallet.

excludeFromFee(): Owner can exclude address from fees.

includeToFee(): Owner can include address to fees.

excludeFromReflection(): Owner can exclude address from reflections.

setTeamFee(): Owner can change the team fee (max 25%).

setTaxFee(): Owner can change tax fee (max 25%).

Recommendation:

-Add a time lock on privileged operations.

-Use a multisig wallet to prevent SPOF on the Private Key.

-Introduce DAO mechanism for owner functions (will add transparency and user involvement).

3.3 Low severity issues [1]

1. Sandwich attack

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 backrunning (after the transaction being attacked) a transaction to sell the asset, this can happen on large input amounts.

Recommendation:

It's better to set a reasonable minimum output amount, instead of 0 based on token price when you call uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens.

```
function swapTokensForEth(uint256 tokenAmount) private lockTheSwap {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapRouter.WETH();
    _approve(address(this), address(uniswapRouter), tokenAmount);
    uniswapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp
    );
}
```

2. Improved function naming

The function ‘setMFCWallet’ may not be properly understood.

Recommendation:

We recommend changing to ‘setMarketingWallet’ or ‘setFSVWallet’.

```
364     function setMFCWallet(address payable _w2) external onlyOwner {
365         w2 = _w2;
366         _isExcludedFromFee[w2] = true;
367     }
```

4.0 Summary of the audit

Overall contract is safe with the only problem of having a medium degree of centralization which must be protected through the recommendations presented. No liquidity / locking tokens provided.