FoxEscrow Smart Contract Final Audit Report

Overview

FoxEscrow

Locked tokens cannot be traded or used to transact before a specific date, they were initially popularized to prevent large sell-offs in Initial Coin Offerings (ICO's). FoxEscrow Contracts allow for the trustless trading of locked tokens. It creates an Over The Counter (OTC) that brings two parties to trade peer to peer. It solves the solution of using trusted parties, hacky ways and manual processes that were normally used to sell locked tokens, hence allowing safe escrow and transfer of locked tokens. Users selling locked tokens create an offer by deploying a contract specifying amount of token selling, token desired and amount of tokens desired e.g. amount of stablecoin; via Factory Contract. User then deposits the locked tokens to the contract and anyone interested in the token can buy.

Scope of Audit

The scope of this audit was to analyze **FoxEscrow smart contract**'s codebase for quality, security, and correctness.

FoxEscrow Contracts:
https://github.com/FarmersOnlyFi/fox-escrow-contracts/tree/main/src

Branch: main

Commit: 8bdc94e43662d1bfc8b106c58acef6b5b0e2b692

Fixed In:

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of **FoxEscrow smart contracts,** care was taken to ensure:

- 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/HRC20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the Smart contract is structured in a way that will not result in future problems.

Static Analysis

Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually

analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

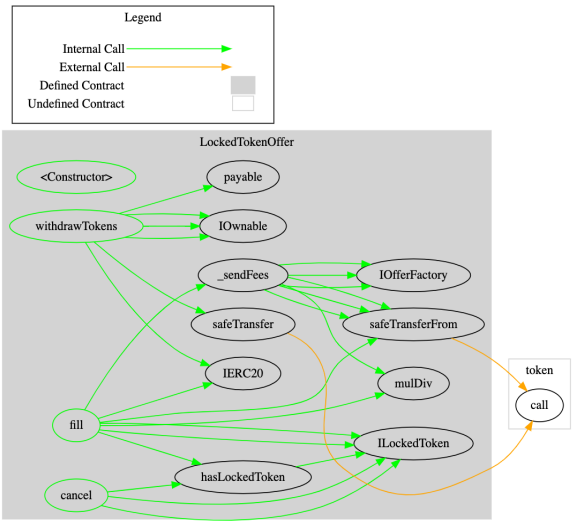
Gas Consumption

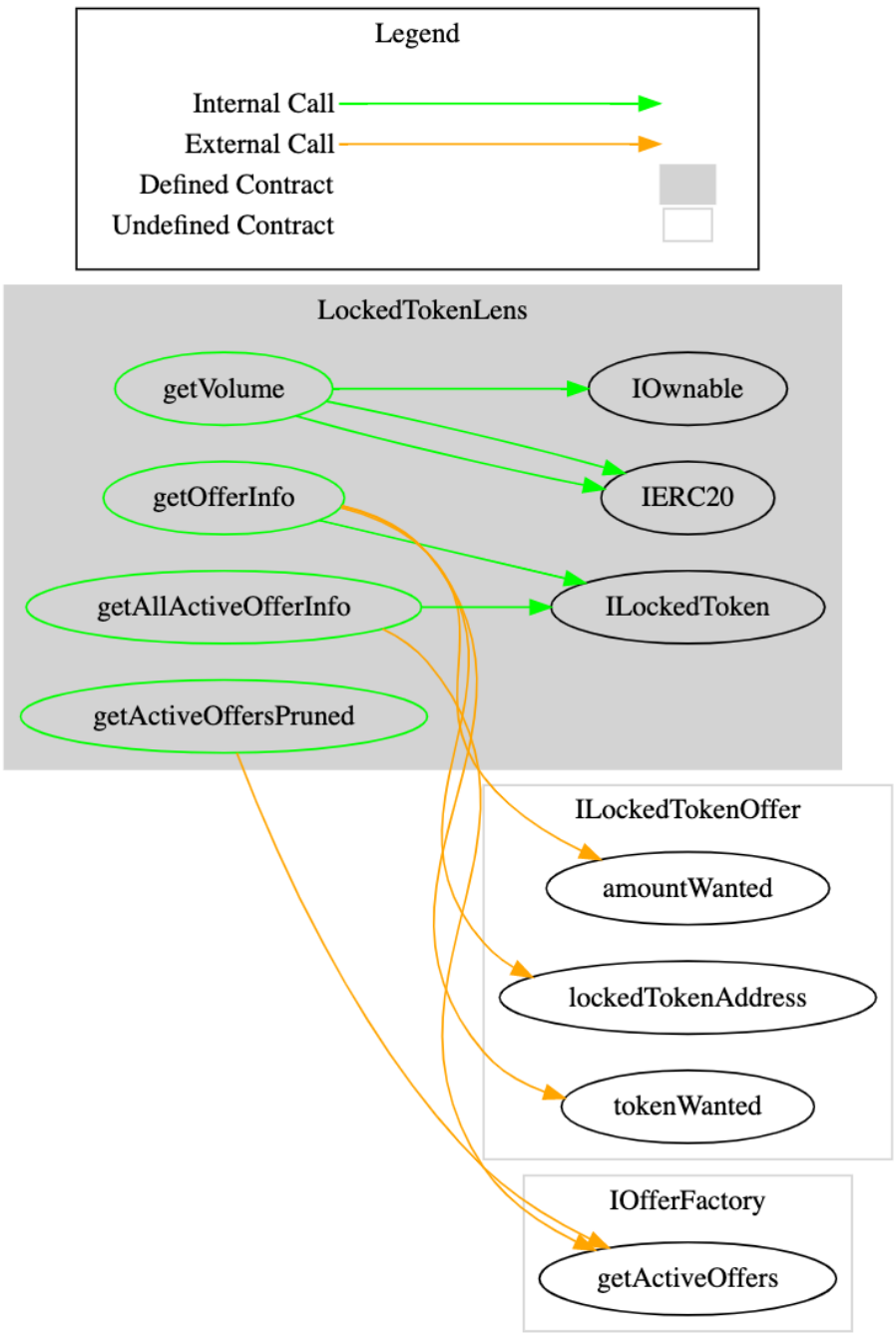
In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

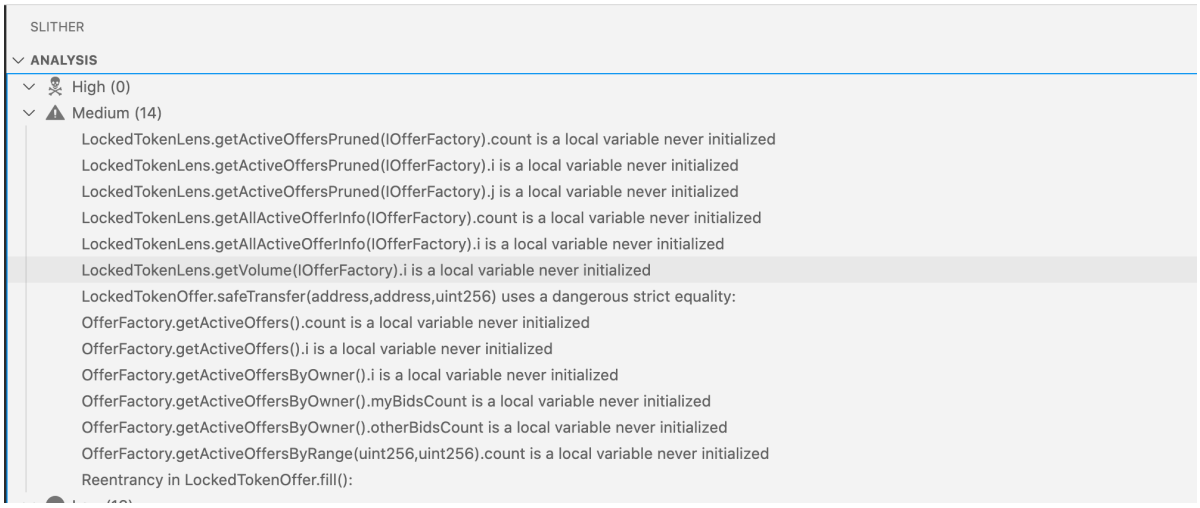
Slither, MythX, Truffle, Remix, Ganache, Solidity Metrics

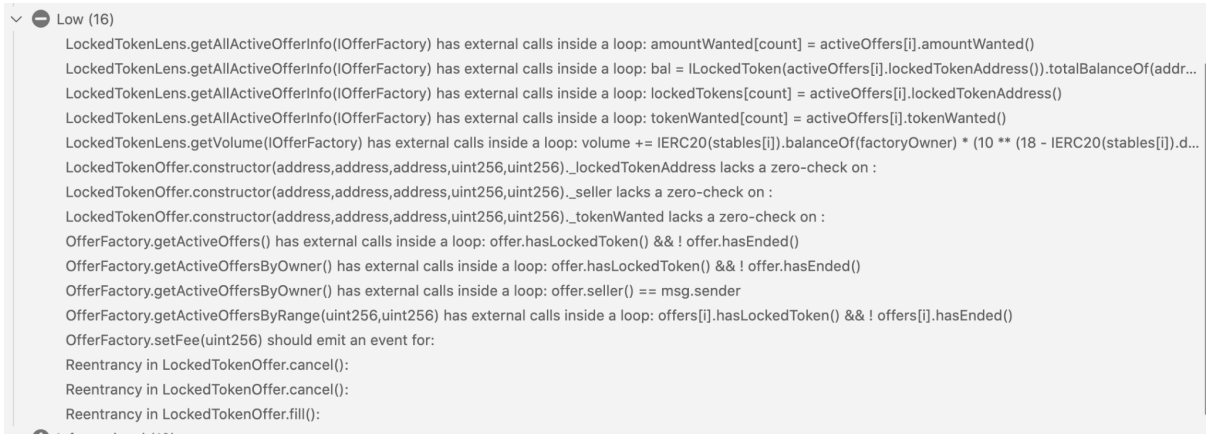
Call Graphs




Legend
Internal Call
External Call
Defined Contract
Undefined Contract
LockedTokenLens
getVolume
IOwnable
IERC20
getOfferInfo
getAllActiveOfferInfo
ILockedToken
getActiveOffersPruned
ILockedTokenOffer
amountWanted
lockedTokenAddress
tokenWanted
IOfferFactory
getActiveOffers

Slither Analysis Results





MythX Analysis Results

Issues

These were the issues detected in the scan. You can use the toggles to filter by severity and display or hide ignored issues.

Severity **Low (1)** **Medium (0)** **High (0)**

Ignored Issues Hidden (0)

ID	Severity	Name	File	Location
SWC-103	Low	A floating pragma is set.	offerfactory.sol	L: 3 C: 0

Issues

These were the issues detected in the scan. You can use the toggles to filter by severity and display or hide ignored issues.

Severity **Low (1)** **Medium (0)** **High (0)**

Ignored Issues Hidden (0)

ID	Severity	Name	File	Location
SWC-103	Low	A floating pragma is set.	lockedtokenoffer.sol	L: 3 C: 0

Issues

These were the issues detected in the scan. You can use the toggles to filter by severity and display or hide ignored issues.

Severity **Low (3)** **Medium (0)** **High (0)**

Ignored Issues Hidden (0)

ID	Severity	Name	File	Location
SWC-103	Low	A floating pragma is set.	lockedtokenlens.sol	L: 3 C: 0
SWC-123	Low	Requirement violation.	lockedtokenlens.sol	L: 39 C: 50
SWC-123	Low	Requirement violation.	lockedtokenlens.sol	L: 7 C: 0

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

TYPE	HIGH	MEDIUM	LOW	INFORMATIONAL
Open	0	0	0	0
Acknowledged	0	0	4	4
Closed	0	1	1	6

Functional Testing Results

Complete functional testing report has been attached below:

Some of the tests performed are mentioned below: Visit Link

OfferFactory

- ✅ should be able to create a new Offer contract
- ✅ should be able to add support for locked tokens
- ✅ should have an owner
- ✅ should be able to have addresses for fees and various parameters changeable by owner

LockedTokenOffer

- ✅ should be able to cancel an open offer
- ✅ should be able to fill any open offer by anyone willing to participate

LockedTokenLens

- ✅ should be able to get information on a selected offer
- ✅ should be able to get volume
- ✅ should be able to get information on all active offers

Tests

```
  Contract: OfferFactory
    OfferFactory
      deployment
        ✓ tracks initial parameters correctly, seller, factory, lockTokenAddress, am
ountWanted, fee  (126ms)
      setFee()
        ✓ should be rejected if called by anyone else not owner (296ms)
        ✓ owner can setFee in valid range (146ms)
Is setting zero fee desired behaviour???
        ✓ owner can setFee zero Fee (83ms)
        ✓ should not propagate fee (1974ms)
        ✓ should be rejected if fee is above MAX_FEE (63ms)
      setEscrowFeeAddress()
        ✓ should be rejected if called by anyone else not owner (63ms)
        ✓ should be rejected if zero address is set for escrowFee address (63ms)
        ✓ should set newEscrowAddress and event, if non-zero address set by owner (9
1ms)
      setDevAddress()
        ✓ should be rejected if called by anyone else not owner (70ms)
        ✓ should be rejected if zero address is set for escrowFee address (63ms)
        ✓ should set newEscrowAddress and event, if non-zero address set by owner (8
0ms)
      addLockedTokenSupport()
        ✓ should be rejected if called by anyone else not owner (398ms)
Is setting zero address locked token desired behaviour???
        ✓ owner can add zero Address token as supported Token (433ms)
        ✓ owner can add lockedToken and event is emitted and supportedTokens mapping
 (2588ms)
      createOffer()
        ✓ should be rejected if lockedToken first not supported (385ms)
        ✓ any user can create offer for supported token, emits event (1786ms)
        ✓ pushes offer to offers array (1843ms)


  18 passing (57s)
```

Issues Found

High Severity Issues

None

Medium Severity Issues

1. Possibility of reentrancy

External call was detected in the cancel() function and fill() function, assuming that malicious code might execute. Even if that external contract is not malicious, malicious code can be executed by any contracts it calls.

```
function cancel() public {
        require(hasLockedToken(), "no Locked Token balance");
        require(msg.sender == seller);
        uint256 balance =
ILockedToken(lockedTokenAddress).totalBalanceOf(address(this));
        ILockedToken(lockedTokenAddress).transferAll(seller);
        hasEnded = true;
        emit OfferCanceled(seller, balance);
    }
```

Remedy

It is recommended adding nonReentrant() for the aforementioned functions to avoid Reentrancy weaknesses.

Auditor's Response: Added Reentrancy Guard

Status : FIXED.

Low Severity Issues

1. Missing value verification

Certain functions lack a safety check in the value, the values that are coming from the arguments should be verified, otherwise, the contract's functionality might be affected or lead to work in unexpected ways. Consider the functions below:

```
function setFee(uint256 _fee) public onlyOwner {
        require(_fee < MAX_FEE, 'Fee to high');
        fee = _fee;
    }
```

In the above function fee can be set as 0 leading to unexpected behavior unless if intended.

Remedy

It is recommended to check that (fee != 0). Consider not only fixing the specific instances mentioned above, but also reviewing the entire codebase to make sure every input that needs some verification is checked.

Auditor's Response:

Status : Acknowledged.

2. Missing address verification

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible, tokens may be burned in perpetuity or code may behave unexpectedly.

```
function setFee(uint256 _fee) public onlyOwner {
```

```
        require(_fee < MAX_FEE, 'Fee to high');
        fee = _fee;
    }
```

In the above function fee can be set as 0 leading to unexpected behavior unless if intended.

Remedy

It's recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Auditor's Response:

Status : Acknowledged .

3. Ownership

OfferFactory is Ownable with the owner having certain privileges like withdrawing lost tokens and ether, ability to change fee amount, dev and escrow fee addresses at any time, any number of times, and renounce ownership(leaving contracts without an owner). This poses centralization and other risks.

Remedy

It's recommended to ensure the owner is not a single entity, consider using a Multisig for the owner. It is recommended to prevent renounceOwnership from being called or limit how and when it can be called.

Auditor's Response:

Status : Acknowledged.

4. Critical address changes

Certain functions change critical addresses like escrow fee addresses, owner addresses. If these changes are made without making sure the new addresses are accessible for example the controller possesses private keys, this can lead to problems for the project. For example fees going to an inaccessible address escrow address with lost keys, which becomes like burning the fees. Function below allows the owner to change addresses without checks if accessible.

```
function setEscrowFeeAddress(address _newAddress) public onlyOwner {
    require(_newAddress != address(0), 'Bad address');
    escrowMultisigFeeAddress = _newAddress;

    emit EscrowFeeAddressUpdate(_newAddress, msg.sender);
}

function setDevAddress(address _newAddress) public onlyOwner {
    require(_newAddress != address(0), 'Bad address');
    devAddress = _newAddress;

    emit DevAddressUpdate(_newAddress, msg.sender);
}
```

Remedy

Changing critical addresses in contracts should be a two-step process where the first transaction (from the old/current address) registers the new address (i.e. grants ownership) and the second transaction (from the new address) replaces the old address with the new one (i.e. claims ownership). This gives an opportunity to recover from incorrect addresses mistakenly used in the first step. If not, contract functionality might become inaccessible. Consider using Claimable contracts from OpenZeppelin

Auditor's Response:

Status : Acknowledged.

5. **Costly functions with loops**

There are various getter functions which rely on looping through arrays to return data. Although the functions are not as unsafe, risky or costly as looping over dynamic arrays they still pause the challenge. Consider the function below:

```
   function getActiveOffersByOwner() public view returns (LockedTokenOffer[] memory,
LockedTokenOffer[] memory) {
       LockedTokenOffer[] memory myBids = new LockedTokenOffer[](offers.length);
       LockedTokenOffer[] memory otherBids = new LockedTokenOffer[](offers.length);

       uint256 myBidsCount;
       uint256 otherBidsCount;
       for (uint256 i; i < offers.length; i++) {
           LockedTokenOffer offer = LockedTokenOffer(offers[i]);
           if (offer.hasLockedToken() && !offer.hasEnded()) {
               if (offer.seller() == msg.sender) {
                   myBids[myBidsCount++] = offers[i];
               } else {
                   otherBids[otherBidsCount++] = offers[i];
               }
           }
       }
       return (myBids, otherBids);
   }
```

To get all your active offers as a user, the function also has to loop over everyones offers to collect your offers. Consider if there are lots of offers, especially if a malicious actor may create a large number of offers with very small token value deposits. This will require reading from storage over a very large array increasing cost of calling the function, or chances of function consuming too much gas affecting the project.

Remedy

It's recommended where possible a users own offers be stored in their own data structure for example `mapping(address => LockedTokenOffer[]`

Auditor's Response: Changed to be More Efficient

Status : Partially Fixed.

Informational

1. **Floating pragma**

Contracts OfferFactory, LockedTokenOffer and LockedTokenLens all make use of pragma ^0.8.11 which allows for solidity compiler versions from 0.8.11 to the version just before 0.9.0. This can result in different versions being used for testing and production. Additionally Contract Ownable not only uses floating pragma ^0.8.0 but this can result in a different version used from other contracts.

Remedy

It is recommended to deploy contracts using the same compiler version/flags with which they have been tested. The solidity version must be fixed by locking the pragma by avoiding using ^. Consider using an earlier stable version like pragma 0.8.4

Auditor's Response:

Status : Acknowledged.

2. **Natspec**

The code is lacking commenting. Comments inline, particularly in Natspec format help to clarify what the code does.

Remedy

It is recommended to use Natspec, a form of comments in Solidity that provides rich documentation for functions, return variables and more.

E.g
```
/// @notice Returns the amount of leaves the tree has.
/// @dev Returns only a fixed number.
```

Auditor's Response:

Status : Acknowledged.

3. Missing test cases

Some functions have not been tested especially those returning values from the blockchain like getActiveOffers(), getActiveOffersByOwner() in OfferFactory.sol

Remedy

It is recommended to write test cases for all the functions. Any existing tests done if failures must be resolved. Tests will help determine if the code is working in the expected way. Unit tests, functional tests and integration tests should have been performed to achieve good test coverage across the entire codebase.

Auditor's Response:

Status : Acknowledged.

4. State variable ordering gas savings

The order and size in which storage variables are declared in a Solidity file impacts the number of storage slots that can be used. In the contract LockedTokenOffer the bool variable may end up occupying its own storage slot.

```
address public immutable factory;
address public immutable seller;
address public immutable lockedTokenAddress;
address public immutable tokenWanted;
uint256 public immutable amountWanted;
uint256 public immutable fee; // in bps
bool public hasEnded = false;
```

Remedy

It is recommended to take advantage of Solidity storage packing and order smaller sized variables like addresses taking 20 bytes, bool taking 1 byte next to each other. If space permits they can be packed into a single storage slot. Consider moving `bool public hasEnded = false;` to the top of the variables list.

Auditor's Response: Variables Reordered

Status : Fixed.

5. No error messages in require functions

There are few places in the entire codebase where the require() statement does not contain any error message. As error messages are intended to notify users about failing conditions, they should provide enough information so that appropriate corrections can be made to interact with the system. Below is a non-exhaustive list of identified instances:

`require(msg.sender == IOwnable(factory).owner());` in LockedTokenOffer => withdrawTokens() function

`require(msg.sender == seller);` in LockedTokenOffer => cancel() function

`require(msg.sender == seller);` in LockedTokenOffer => cancel() function

Remedy

Lack of error messages greatly damage the overall user experience, thus lowering the system's quality. Consider not only fixing the specific instances mentioned above, but also reviewing the entire codebase to make sure every error message is informative and user-friendly.

Auditor's Response: Require message Added

Status : Fixed.

6. Public functions that can be made external

Public functions that are not called within the contract can be made external. External functions cost less than public functions.

`function cancel() public` in LockedTokenOffer.sol

`function fill() public` in LockedTokenOffer.sol

`function getVolume(IOfferFactory factory) public view returns (uint256 sum)` in LockedTokenOffer.sol

`function getOfferInfo(ILockedTokenOffer offer)public view returns` in LockedTokenOffer.sol

`function getActiveOffersPruned(IOfferFactory factory) public view returns (ILockedTokenOffer[] memory)` in LockedTokenOffer.sol

```
function getAllActiveOfferInfo(IOfferFactory factory) public view
 returns (
           address[] memory lockedTokens,
           address[] memory offerAddresses,
           uint256[] memory lockedBalances,
           address[] memory tokenWanted,
           uint256[] memory amountWanted
       )
```
in LockedTokenOffer.sol

`function addLockedTokenSupport(address _lockedToken, bool _supported) public onlyOwner` In OfferFactory.sol

`function createOffer(address _lockedTokenAddress, address _tokenWanted, uint256 _amountWanted) public returns (LockedTokenOffer)` in OfferFactory.sol

`function getActiveOffersByOwner() public view returns (LockedTokenOffer[] memory, LockedTokenOffer[] memory)` in OfferFactory.sol

`function getActiveOffers() public view returns (LockedTokenOffer[] memory)` in OfferFactory.sol

`function getActiveOffersByRange(uint256 start, uint256 end) public view returns (LockedTokenOffer[] memory)` in OfferFactory.sol

Remedy

It is recommended to make the above mentioned functions external to save on gas costs. Consider not only fixing the specific instances mentioned above, but also reviewing the entire codebase to make sure every error message is informative and user-friendly.

Auditor's Response: Relevant Public Functions made External

Status : Fixed.

7. Consider code reuse

When code is repeated in functions it may lead to difficulty in following code,code maintainability issues, higher costs due to large size code base.

```
function getActiveOffers() public view returns (LockedTokenOffer[] memory) {
        LockedTokenOffer[] memory activeOffers = new LockedTokenOffer[](offers.length);
        uint256 count;
        for (uint256 i; i < offers.length; i++) {
            LockedTokenOffer offer = LockedTokenOffer(offers[i]);
            if (offer.hasLockedToken() && !offer.hasEnded()) {
                activeOffers[count++] = offer;
            }
        }
        return activeOffers;
    }
```

The looping logic in the getActiveOffers() function is almost similar to that in getActiveOffersByRange() below and to some extent getActiveOffersByOwner()

```
function getActiveOffersByRange(uint256 start, uint256 end) public view returns
(LockedTokenOffer[] memory) {
        LockedTokenOffer[] memory activeOffers = new LockedTokenOffer[](end - start);
        uint256 count;
        for (uint256 i = start; i < end; i++) {
            if (offers[i].hasLockedToken() && !offers[i].hasEnded()) {
                activeOffers[count++] = offers[i];
            }
        }
        return activeOffers;
    }
}
```

Remedy

It may be possible to refactor the code into an internal function flexible enough to be reused in all the functions where needed.

Auditor's Response: Some Changes Made

Status : Partially Fixed.

8. Missing events

The missing event makes it difficult to track off-chain and impact off-chain monitoring and incident response functionality. An event should be emitted for significant transactions calling the following functions:

```
function setFee(uint256 _fee) public onlyOwner
```

Remedy

Critical updates need to emit events. We recommend emitting an event to log the update of the above functions.

Auditor's Response:

Status : Acknowledged.

9. Unindexed event parameters

Events without indexed parameters may lead to challenges for off-chain tooling that are expecting indexed events. Solidity events can have up to 3 parameters indexed.

```
event OfferFilled(address buyer, uint256 lockedTokenAmount, address token, uint256
tokenAmount);
event OfferCanceled(address seller, uint256 lockedTokenAmount);
event OfferCreated(address offerAddress, address lockedTokenAddress, address
tokenWanted, uint256 amountWanted);
event LockedTokenSupportUpdate(address lockedTokenAddress, bool supported);
```

```
event EscrowFeeAddressUpdate(address newAddress, address caller);
event DevAddressUpdate(address newAddress, address caller);
```

Remedy

It is recommended to make the above mentioned event parameters indexed. This allows them to be accessed faster as they are included in log bloom filters. Consider not only fixing the specific instances mentioned above, but also reviewing the entire codebase to make sure that events are indexed appropriately.

Auditor's Response: Event Parameter Indexed

Status : Fixed.

10. Spelling and grammar

Incorrect spellings may impact code readability, code understanding or be misinterpreted leading to unexpected behaviors. For example, the code below "Fee to high" must be "Fee too high". Someone on reading output may interpret the incorrect message as fee going somewhere

```
function setFee(uint256 _fee) public onlyOwner {
    require(_fee < MAX_FEE, 'Fee to high');
    fee = _fee;
}
```

Remedy

Fix all spelling and grammar and ensure require strings can be easily understood without misinterpretation. Consider not only fixing the specific instances mentioned above, but also reviewing the entire codebase to make sure proper spelling and grammar is used.

Auditor's Response: Fixes Made

Status : Fixed.

Closing Summary:

Some issues of Medium, Low and Informational severity were found in the Initial Audit. At the end most of the issues have been fixed and acknowledged by the FoxEscrow Team.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of **FoxEscrow.** This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the FoxEscrow team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.