AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

LuxVault

DATED : 29 Oct 24'

MANUAL TESTING

Centralization – Enabling Trades

Severity: High

Function: EnableTrading

Status: Open

Overview:

The EnableTrading function permits only the contract owner to activate trading capabilities. Until this function is executed, no investors can buy, sell, or transfer their tokens. This places a high degree of control and centralization in the hands of the contract owner.

```
function enableTrading() external onlyOwner {
    if (tradingEnabled) revert TradingAlreadyEnabled();

    tradingEnabled = true;

    emit TradingEnabled();
}
```

Suggestion:

To reduce centralization and potential manipulation, consider one of the following approaches:

•Automatically enable trading after a specified condition, such as the completion of a presale, is met.

•If manual activation is still desired, consider transferring the ownership of the contract to a trustworthy, third-party entity like a certified "PinkSale Safu" developer. This can give investors more confidence in the eventual activation of trading capabilities, mitigating concerns of potential bad-faith actions by the original owner.

MANUAL TESTING

Centralization – The owner can Blacklist Wallet.

Severity: High

Function: blacklist

Status: Open

Overview:
The owner can blacklist multiple wallets.

```
function blacklist(address account, bool isBlacklisted) external onlyOwner {
    blacklisted[account] = isBlacklisted;

    emit BlacklistUpdated(account, isBlacklisted);
}
```

Suggestion:
There should be a locking period so that the wallet cannot be locked in an indefinite period.

AUDIT SUMMARY

Project name – LuxVault

Date: 29 Oct, 2024

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: **High-risk major flag**

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	2	2	1	1
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:
A line by line code review has been performed by audit ace team.

2- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.

3- Slither :
The code has undergone static analysis using Slither.

Testnet version:
The tests were performed using the contract deployed on the BSC Testnet, which can be found at the following address:

Token Information

Token Address:
0xf581400cf99C9b6b478aDED8981ecef157eEc820

Name: LuxVault

Symbol: LUXV

Decimals: 18

Network: Ether Scan

Token Type: ERC-20

Owner: 0xBD96a9f98904794985923324C86F3d041361eD69

Deployer:
0xBD96a9f98904794985923324C86F3d041361eD69

Token Supply: 200,000,000

Checksum: bd5f6c909f027445e506cbfd02e81182

TOKEN OVERVIEW

Buy Fee: 0-3%

Sell Fee: 0-3%

Transfer Fee: 0-0%

Fee Privilege: Owner

Ownership: Owned

Minting: Yes

Max Tx: No

Blacklist: Yes

AUDIT METHODOLOGY

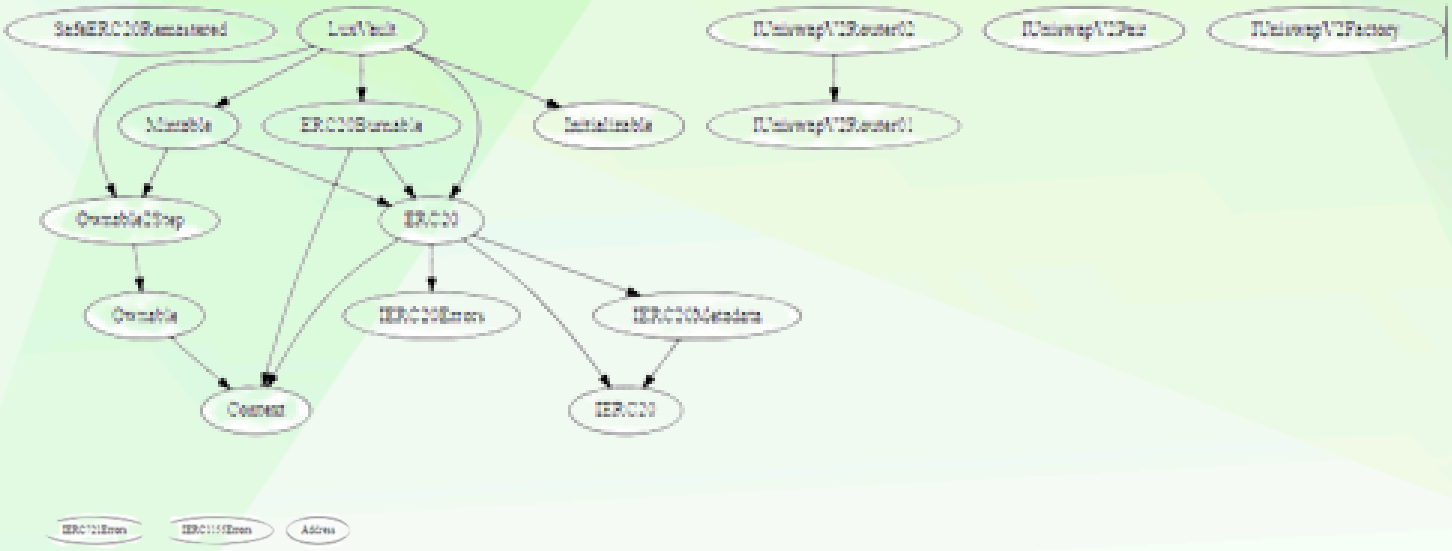
The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

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

INHERITANCE TREE



POINTS TO NOTE

- The owner can set AMM.
- The owner can recover foreignERC20.
- The owner can update the Swap threshold.
- The owner can blacklist.
- The owner can Enable trading.
- The owner can update the marketing address setup.
- The owner can recover the token.
- The owner can set the max sell amount and update the max wallet amount.

STATIC ANALYSIS

```
INFO:Detectors:
Address._revert(bytes) (Address.sol#146-158) uses assembly
        - INLINE ASM (Address.sol#151-154)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Version constraint ^0.8.20 contains known severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
        - VerbatimInvalidDeduplication
        - FullInlinerNonExpressionSplitArgumentEvaluationOrder
        - MissingSideEffectsOnSelectorAccess.
It is used by:
        - ^0.8.20 (Address.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
INFO:Detectors:
Version constraint ^0.8.20 contains known severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
        - VerbatimInvalidDeduplication
        - FullInlinerNonExpressionSplitArgumentEvaluationOrder
        - MissingSideEffectsOnSelectorAccess.
It is used by:
        - ^0.8.20 (IERC20.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
INFO:Detectors:
Version constraint >=0.5.0 contains known severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
        - DirtyBytesArrayToStorage
        - ABIDecodeTwoDimensionalArrayMemory
        - KeccakCaching
        - EmptyByteArrayCopy
        - DynamicArrayCleanup
        - ImplicitConstructorCallvalueCheck
        - TupleAssignmentMultiStackSlotComponents
        - MemoryArrayCreationOverflow
        - privateCanBeOverridden
        - SignedArrayStorageCopy
        - ABIEncoderV2StorageArrayWithMultiSlotElement
        - DynamicConstructorArgumentsClippedABIV2
        - UninitializedFunctionPointerInConstructor
        - IncorrectEventSignatureInLibraries
        - ABIEncoderV2PackedStorage.
It is used by:
        - >=0.5.0 (IUniswapV2Factory.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

STATIC ANALYSIS

```
INFO:Detectors:
LuxVault.constructor() (LuxVault.sol#151-184) uses literals with too many digits:
        - updateMaxWalletAmount(500000000 * (10 ** decimals()) / 10) (LuxVault.sol#172)
LuxVault.constructor() (LuxVault.sol#151-184) uses literals with too many digits:
        - updateMaxSellAmount(50000000 * (10 ** decimals()) / 10) (LuxVault.sol#174)
LuxVault.constructor() (LuxVault.sol#151-184) uses literals with too many digits:
        - updateMaxTransferAmount(100000000 * (10 ** decimals()) / 10) (LuxVault.sol#175)
LuxVault.constructor() (LuxVault.sol#151-184) uses literals with too many digits:
        - _mint(supplyRecipient,2000000000 * (10 ** decimals()) / 10) (LuxVault.sol#182)
LuxVault.constructor() (LuxVault.sol#151-184) uses literals with too many digits:
        - Mintable(4000000000) (LuxVault.sol#154)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
The following unused import(s) in LuxVault.sol should be removed:
        -import "./IUniswapV2Pair.sol"; (LuxVault.sol#52)

        -import "./IUniswapV2Router01.sol"; (LuxVault.sol#53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-imports
INFO:Detectors:
Mintable.maxSupply (Mintable.sol#10) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutabl
e
INFO:Slither:LuxVault.sol analyzed (19 contracts with 94 detectors), 60 result(s) found
```

```
INFO:Detectors:
LuxVault._update(address,address,uint256) (LuxVault.sol#411-486) uses a Boolean constant improperly:
        -false || _marketingPending > 0 (LuxVault.sol#454)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#misuse-of-a-boolean-constant
INFO:Detectors:
LuxVault._update(address,address,uint256) (LuxVault.sol#411-486) performs a multiplication on the result of a division:
        - fees = amount * totalFees[txType] / 10000 (LuxVault.sol#434)
        - _marketingPending += fees * marketingFees[txType] / totalFees[txType] (LuxVault.sol#437)
LuxVault._update(address,address,uint256) (LuxVault.sol#411-486) performs a multiplication on the result of a division:
        - fees = amount * totalFees[txType] / 10000 (LuxVault.sol#434)
        - _liquidityPending += fees * liquidityFees[txType] / totalFees[txType] (LuxVault.sol#439)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
LuxVault._addLiquidity(uint256,uint256) (LuxVault.sol#288-292) ignores return value by routerV2.addLiquidityETH{value: c
oinAmount}(address(this),tokenAmount,0,0,address(0),block.timestamp) (LuxVault.sol#291)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

Result => A static analysis of contract's source code has been performed using slither,

No major issues were found in the output

CLASSIFICATION OF RISK

Severity	Description
Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
Critical	0
High-Risk	2
Medium-Risk	2
Low-Risk	1
Gas Optimization / Suggestions	1

MANUAL TESTING

Centralization – Enabling Trades

Severity: High

Function: EnableTrading

Status: Open

Overview:

The EnableTrading function permits only the contract owner to activate trading capabilities. Until this function is executed, no investors can buy, sell, or transfer their tokens. This places a high degree of control and centralization in the hands of the contract owner.

```
function enableTrading() external onlyOwner {
    if (tradingEnabled) revert TradingAlreadyEnabled();

    tradingEnabled = true;

    emit TradingEnabled();
}
```

Suggestion:

To reduce centralization and potential manipulation, consider one of the following approaches:

•Automatically enable trading after a specified condition, such as the completion of a presale, is met.

•If manual activation is still desired, consider transferring the ownership of the contract to a trustworthy, third-party entity like a certified "PinkSale Safu" developer. This can give investors more confidence in the eventual activation of trading capabilities, mitigating concerns of potential bad-faith actions by the original owner.

MANUAL TESTING

Centralization – The owner can Blacklist Wallet.

Severity: High

Function: blacklist

Status: Open

Overview:
The owner can blacklist multiple wallets.

```
function blacklist(address account, bool isBlacklisted) external onlyOwner {
    blacklisted[account] = isBlacklisted;

    emit BlacklistUpdated(account, isBlacklisted);
}
```

Suggestion:
There should be a locking period so that the wallet cannot be locked in an indefinite period.

MANUAL TESTING

Centralization – The owner can regain ownership.

Severity: Medium

Function: pendingOwner

Status: Open

Overview:

The owner can regain ownership after transferring it with the following steps:

1. Call lock function to set previous owner to the own address
2. Call unlock function to get ownership back
3. Transfer/renounce ownership
4. Call unlock function to get ownership back

```
abstract contract Ownable2Step is Ownable {
address private _pendingOwner;

event OwnershipTransferStarted(address indexed previousOwner, address indexed newOwner);

/**
* @dev Returns the address of the pending owner.
*/
function pendingOwner() public view virtual returns (address) {
      return _pendingOwner;
}

/**
* @dev Starts the ownership transfer of the contract to a new account. Replaces the pending transfer if there is one.
* Can only be called by the current owner.
*/
```

MANUAL TESTING

```
function transferOwnership(address newOwner) public virtual override
onlyOwner {
      _pendingOwner = newOwner;
      emit OwnershipTransferStarted(owner(), newOwner);
}

/**
* @dev Transfers ownership of the contract to a new account (`newOwner`)
and deletes any pending owner.
* Internal function without access restriction.
*/
function _transferOwnership(address newOwner) internal virtual override {
      delete _pendingOwner;
      super._transferOwnership(newOwner);
}

/**
* @dev The new owner accepts the ownership transfer.
*/
function acceptOwnership() public virtual {
      address sender = _msgSender();
      if (pendingOwner() != sender) {
         revert OwnableUnauthorizedAccount(sender);
      }
      _transferOwnership(sender);
}
};
```

Suggestion: Set the previous ownership back to address zero after using the unlock function.

MANUAL TESTING

Centralization – Missing Require Check.

Severity: Medium

Function: marketing address setup

Status: Open

Overview:
The owner can set any arbitrary address, excluding zero addresses. This is not recommended because if the owner sets the address to the contract address, then the Eth will not be sent to that address, the transaction will fail, and this will lead to a potential honeypot in the contract.

```
function marketingAddressSetup(address _newAddress) public onlyOwner {
    if (_newAddress == address(0)) revert
InvalidTaxRecipientAddress(address(0));

    marketingAddress = _newAddress;
    excludeFromFees(_newAddress, true);
    _excludeFromLimits(_newAddress, true);

    emit WalletTaxAddressUpdated(1, _newAddress);
}
```

Suggestion: It is recommended that the address should not be able to set as a contract address.

MANUAL TESTING

Centralization – Missing Events

Severity: Low

Subject: Missing Events

Status: Open

Overview:

They serve as a mechanism for emitting and recording data onto the blockchain, making it transparent and easily accessible.

```
function recoverToken(uint256 amount) external onlyOwner {
    uint256 maxRecoverable = balanceOf(address(this)) - getAllPending();
    if (amount > maxRecoverable) revert InvalidAmountToRecover(amount,
maxRecoverable);

    _update(address(this), msg.sender, amount);
}

function recoverForeignERC20(address tokenAddress, uint256 amount)
external onlyOwner {
    if (tokenAddress == address(this)) revert InvalidToken(tokenAddress);

IERC20(tokenAddress).safeTransfer(msg.sender, amount);
```

MANUAL TESTING

Optimization

Severity: Optimization

Subject: Remove unused code.

Status: Open

Overview:
Unused variables are allowed in Solidity, and they do. not pose a direct security issue. It is the best practice. though to avoid them.

```
function safeTransfer(IERC20 token, address to, uint256 value) internal {
_callOptionalReturn(token, abi.encodeCall(token.transfer, (to, value)));
  }

  /**
   * @dev Transfer `value` amount of `token` from the calling contract to `to`. If `token` returns no value,
   * non-reverting calls are assumed to be successful.
   */
  function safeTransfer_noRevert(IERC20 token, address to, uint256 value) internal returns (bool) {
      return _callOptionalReturnBool(token, abi.encodeCall(token.transfer, (to, value)));
  }

  /**
   * @dev Transfer `value` amount of `token` from `from` to `to`, spending the approval given by `from` to the
   * calling contract. If `token` returns no value, non-reverting calls are assumed to be successful.
   */
  function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
      _callOptionalReturn(token, abi.encodeCall(token.transferFrom, (from, to, value)));
  }
```

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace