SALUS SECURITY

AUG 2023

CODE SECURITY ASSESSMENT

DAY OF DEFEAT

Overview

Project Summary

- Name: Day of Defeat
- Platform: BNB Smart Chain
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Day of Defeat
Version	v6
Type	Solidity
Dates	August 07 2023
Logs	July 04 2023; July 13 2023; July 14 2023; July 25 2023; July 31 2023; August 07 2023

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	5
Total Low-Severity issues	5
Total informational issues	10
Total	22

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	DoS by front-running the transactions that set a new pool	High	Denial of Service	Acknowledged
2	Use of testnet addresses	High	Configuration	Resolved
3	Incorrect function calls to V3 routers	Medium	Denial of Service	Resolved
4	The absence of decimal checks could lead to incorrect supply control	Medium	Numerics	Resolved
5	SwapAmount should be cached before calculating the tax	Medium	Business Logic	Resolved
6	Transfers unrelated to liquidity pools may be blocked due to inappropriate checks	Medium	Denial of Service	Resolved
7	Centralization risk	Medium	Centralization	Mitigated
8	Transfer can be blocked when _poolCA is not set	Low	Business Logic	Resolved
9	Precision loss in prize tax calculation	Low	Numerics	Resolved
10	Incomplete update of the excluded lists and allowances	Low	Business Logic	Resolved
11	Insecure access control for _hasLimits()	Low	Access Control	Resolved
12	No need to authenticate the caller in view functions	Low	Data Validation	Acknowledged
13	Use of floating pragma	Informational	Configuration	Resolved
14	Missing two-step transfer ownership pattern	Informational	Business logic	Resolved
15	Inconsistent use of router address in event emission	Informational	Logging	Resolved
16	Unexpected revert if the total supply is 0	Informational	Code Quality	Resolved
17	Defined but unused events	Informational	Logging	Resolved
18	Inappropriate error message	Informational	Code Quality	Resolved

19	Missing zero address check	Informational	Data Validation	Resolved
20	Redundant variables	Informational	Redundancy	Resolved
21	Redundant code	Informational	Redundancy	Resolved
22	Gas Optimization Suggestions	Informational	Gas Optimization	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. DoS by front-running the transactions that set a new pool

Severity: High	Category: Denial of Service

Target:
- DOD_Token_2_1_03.sol
- LUSD_Token_1_0.sol

Description

When setting a new liquidity pool in the DOD_Token_2_1_03 contract, it will first check the pool's existence. If the pair already exists or if isLiqPool[lpCA] is true, the transaction will be reverted. However, if a pair does not exist, an attacker can front-run to create the target new pair. In this scenario, although isLiqPool[lpCA] is false, lpCA will never be added.

DOD_Token_2_1_03.sol:L314-L316

```
lpCA = IFactoryV2( IDexRouterV2(V2ROUTER).factory() ).getPair( _LPTargetCoinCA,
address(this) );
require(lpCA == address(0) && !isLiqPool[lpCA], "StudioL_Token: Pair already exists!");
lpCA = IFactoryV2( IDexRouterV2(V2ROUTER).factory() ).createPair( _LPTargetCoinCA,
address(this) );
```

Similar to the above, in the Legacy_Stable_1_0 contract, it only verifies whether isLiqPool[lpCA] is false before calling IFactoryV2.createPair or IFactoryV3.createPool. However, if the pair or the pool exists, the call to IFactoryV2.createPair or IFactoryV3.createPool will fail due to the inclusion of an existence check.

LUSD_Token_1_0.sol:L267-L275

```
if(_V2orV3) {
    lpCA = IFactoryV2( factoryCAperDex[_router] ).getPair( _LPTargetCoinCA,
address(this) );
    require(!isLiqPool[lpCA], "StudioL_Token: Pair already exists!");
    lpCA = IFactoryV2( factoryCAperDex[_router] ).createPair( _LPTargetCoinCA,
address(this) );
} else {
    lpCA = IFactoryV3( factoryCAperDex[_router] ).getPool( _LPTargetCoinCA,
address(this), 3000 );
    require(!isLiqPool[lpCA], "StudioL_Token: Pool already exists!");
    lpCA = IFactoryV3( factoryCAperDex[_router] ).createPool( _LPTargetCoinCA,
address(this), 3000 );
}
```

Recommendation

Consider modifying the require statement from `lpCA == address(0) && !isLiqPool[lpCA]` to `!isLiqPool[lpCA]` in the DOD_Token_2_1_03 contract, and executing the creation only when a pair or a pool does not exist.

Status

This issue has been acknowledged by the team.

2. Use of testnet addresses

Severity: High	Category: Configuration
Target: - DOD_Token_2_1_03.sol	

Description

The constants V2ROUTER and V3ROUTER have been set to testnet addresses in the DOD_Token_2_1_03 contract.

DOD_Token_2_1_03.sol:L66-L69

```
address constant public V2ROUTER = 0xD99D1c33F9fC3444f8101754aBC46c52416550D1;
// Uniswap V2 = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, PCS V3 Mainnet =
0x10ED43C718714eb63d5aA57B78B54704E256024E, Testnet =
0xD99D1c33F9fC3444f8101754aBC46c52416550D1
address constant public V3ROUTER = 0x9a489505a00cE272eAa5e07Dba6491314CaE3796;
// Uniswap V3 = 0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45, PCS V3 Mainnet =
0x13f4EA83D0bd40E75C8222255bc855a974568Dd4, Testnet =
0x9a489505a00cE272eAa5e07Dba6491314CaE3796
```

Recommendation

Consider configuring the correct addresses for the routers before deploying to the mainnet.

Status

This issue has been resolved by the team. The V2 router can be updated by the owner.

3. Incorrect function calls to V3 routers

Severity: Medium	Category: Denial of Service
Target: - LUSD_Token_1_0.sol	

Description

In the syncNativeANDStablePairs function, if _syncNativeCoin is true and there is a V3 pool with pairedIsNative equal to true, the transaction will be reverted since there is no getAmountsOut function in the V3 router.

LUSD_Token_1_0.sol:L461-L480

```
if ( _syncNativeCoin && LiqPool.pairedIsNative ) {

        uint256 ourPathCoin0Balance = IERC20(pathCoin0).balanceOf(poolAddr);
        uint256 ourLUSDBalance = _tOwned[poolAddr];

        amounts = IDexRouterV2(LiqPool.dexCA).getAmountsOut(ourPathCoin0Balance,
refPath);

        if( ourLUSDBalance > amounts[1] ) { // ourLUSDBalance is over

            _supplyControl(poolAddr, ( ourLUSDBalance - amounts[1] ), false); // Match
the stablecoin amount

        } else if( ourLUSDBalance < amounts[1]) { // ourLUSDBalance is short

            _supplyControl(poolAddr, ( amounts[1] - ourLUSDBalance ), true); // Match the
stablecoin amount
        }
        if( LiqPool.V2orV3 ) {
            IPairV2(poolAddr).sync();
        }
        emit PoolSynced( _msgSender(), LiqPool.pairedCoinCA, poolAddr, true,
block.timestamp );
}
```

Recommendation

Consider implementing a validation step before retrieving the amount. For V3 routers, consider using Quoter.quoteExactInput() to get the amount out received for a given exact input swap.

Status

This issue has been resolved by the team. The team chose to only use v2Router.

4. The absence of decimal checks could lead to incorrect supply control

Severity: Medium	Category: Numerics
Target: - LUSD_Token_1_0.sol	

Description

In the syncNativeANDStablePairs function, the current balances of token0 and token1 in the pool are checked before the minting or burning process. Then, a comparison is made to determine the required amount of LUSD tokens to be minted or burned.

LUSD_Token_1_0.sol:L483-L492

```
uint256 pairedCoinBalance = IERC20(LiqPool.pairedCoinCA).balanceOf(poolAddr);

if( pairedCoinBalance > _tOwned[poolAddr] ) {

    _supplyControl(poolAddr, ( pairedCoinBalance - _tOwned[poolAddr] ), true);

} else if( pairedCoinBalance < _tOwned[poolAddr] ){

    _supplyControl(poolAddr, ( _tOwned[poolAddr] - pairedCoinBalance ), false);
}
```

LUSD has 18 decimals, but if the corresponding token has a different number of decimals (for example, USDT which has 6 decimals on the ETH network or other), minting or burning may result in an unbalanced state, leading to significant deviation in the price of LUSD.

Recommendation

It is necessary to compare decimals before performing calculations involving token amounts.

Status

This issue has been resolved by the team. The pairedCoinBalance will be normalized before calculation if the decimals of pairedCoinCA are not equal to 18.

5. SwapAmount should be cached before calculating the tax

Severity: Medium	Category: Business Logic
Target: - DOD_Token_2_1_03.sol	

Description

The contractSwap() does not cache the swapAmout parameter to a temporary variable. As a result, the updated swapAmount is used for the second tax calculation and all subsequent calculations, leading to incorrect tax calculation results. The vulnerability arises because the swapAmount is reduced after each tax calculation, which affects the subsequent tax calculations.

The code below illustrates the calculation of the first two taxes, with the referralTax utilizing the updated swapAmount.

DOD_Token_2_1_03.sol:L633-L651

```
if(stakingTax > 0) {
    transferBalance = ( ( swapAmount * stakingTax ) / totalTax );
    _tOwned[ stakingPool ] += transferBalance;
    _tOwned[ address(this) ] -= transferBalance;
    emit Transfer( address(this), stakingPool, transferBalance );

    swapAmount -= transferBalance;
    transferBalance = 0;
}

if(referralTax > 0) {
    transferBalance = ( ( swapAmount * referralTax ) / totalTax );
    _tOwned[ referralPool ] += transferBalance;
    _tOwned[ address(this) ] -= transferBalance;
    emit Transfer( address(this), referralPool, transferBalance );

    swapAmount -= transferBalance;
    transferBalance = 0;
}
```

Recommendation

Consider caching the swapAmount to a separate variable before the tax calculations. This will ensure that the original swapAmount value is used for all tax calculations.

Status

This issue has been resolved by the team.

6. Transfers unrelated to liquidity pools may be blocked due to inappropriate checks

Severity: Medium	Category: Denial of Service
Target: - DOD_Token_2_1_03.sol	

Description

Because _poolCA is a state variable, when the transfer is not related to liquidity pools (i.e. other is set to true), the value of _poolCA is the value set in the previous transfer related to liquidity pools. As a result, if _hasLimits(from, to) is true and the trading is disabled for the previously set pool, the transaction will be reverted.

DOD_Token_2_1_03.sol:L552-L572

```
if (isLiqPool[from]) {
    buy = true;
    _poolCA = from;
} else if (isLiqPool[to]) {
    sell = true;
    _poolCA = to;
} else {
    other = true;
}
uint8 index = searchLiqPool(_poolCA);
LPool memory poolInfo = liqPoolList[ address(this) ][index];

if( _hasLimits(from, to) ) {
    if(poolInfo.tradingPauseTime != 0) {
       if( getRemainingPauseTimeInSecs(poolInfo.poolCA) == 0 ) {
            liqPoolList[ address(this) ][index].tradingEnabled = true;
            liqPoolList[ address(this) ][index].tradingPauseTime = 0;
       }
    }
    require(liqPoolList[ address(this) ][index].tradingEnabled, "StudioL_Token: Trading not enabled!");
}
```

Recommendation

It is recommended to check whether the trading is enabled only when buy or sell is true.

Status

This issue has been resolved by the team.

7. Centralization risk

Severity: Medium	Category: Centralization
Target: - DOD_Token_2_1_03.sol - LUSD_Token_1_0.sol	

Description

The DOD_Token_2_1_03 and Legacy_Stable_1_0 contracts have privileged accounts.

The owner of the DOD_Token_2_1_03 contract can modify all the configurations and is granted permission to manipulate all the DOD tokens in the V2ROUTER and all the assets in the DOD_Token_2_1_03 contract.

When the Legacy_Stable_1_0 contract is deployed, the initial supply of LUSD tokens are sent to the _operator. The operator can distribute LUSD tokens without the consensus of the community. Meanwhile, the operator can modify the critical configurations and is granted permission to manipulate all the LUSD tokens in pools and routers, as well as all the assets in the contract.

Should the owner’s or operator’s private key be compromised, an attacker can sweep all the funds in the contract or change the configuration as desired. If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been mitigated by the team. The owner account will be transferred to a multi-sig account upon the launch.

8. Transfer can be blocked when _poolCA is not set

Severity: Low	Category: Business Logic
Target: - DOD_Token_2_1_03.sol - LUSD_Token_1_0.sol	

Description

The _poolCA in the DOD_Token_2_1_03 contract and poolAddr in the Legacy_Stable_1_0 contract are state variables. They are only set after the first transfer associated with liquidity pools. If _poolCA (poolAddr) is not defined, the searchLiqPool function will return type(uint8).max and result in an out-of-bounds exception in the following line.

DOD_Token_2_1_03.sol:L552-L562

```
if (isLiqPool[from]) {
    buy = true;
    _poolCA = from;
} else if (isLiqPool[to]) {
    sell = true;
    _poolCA = to;
} else {
    other = true;
}
uint8 index = searchLiqPool(_poolCA);
LPool memory poolInfo = liqPoolList[ address(this) ][index];
```

LUSD_Token_1_0.sol:L563-L571

```
if ( isLiqPool[from] ) {
        buy = true;
        poolAddr = from;
} else if (isLiqPool[to]) {
        sell = true;
        poolAddr = to;
}
uint8 index = searchLiqPool(poolAddr);
LPool memory LiqPool = liqPoolList[ address(this) ][index];
```

Recommendation

It is recommended to include a check and avoid performing pool-related operations for regular transfers.

Status

This issue has been resolved by the team.

9. Precision loss in prize tax calculation

Severity: Low	Category: Numerics
Target: - DOD_Token_2_1_03.sol	

Description

When calculating the prize tax, half of the prize tax is burned, while the other half goes to the treasury. However, if transferBalance is odd, 1 wei will be lost. There will be a mismatch between the sum of _tOwned and the total supply.

DOD_Token_2_1_03.sol:L654-L657

```
transferBalance = ( (swapAmount * prizeTax) / totalTax );
_tOwned[ DEAD ] += (transferBalance / 2); // half of the prize tax gets burned to DEAD
_tOwned[ transitionCollector ] += (transferBalance / 2); // half of the prize tax goes
to LStable treasury
_tOwned[ address(this) ] -= transferBalance;
```

Recommendation

Consider following the example below.

```
uint256 transferBalanceHalf = transferBalance / 2;
_tOwned[DEAD] += transferBalanceHalf;
_tOwned[transitionCollector] += (transferBalance - transferBalanceHalf);
_tOwned[address(this)] -= transferBalance;
```

Status

This issue has been resolved by the team.

10. Incomplete update of the excluded lists and allowances

Severity: Low	Category: Business Logic
Target: - DOD_Token_2_1_03.sol - LUSD_Token_1_0.sol	

Description

It is recommended to revoke the privileges of the previous accounts when transferring ownership or updating the wallet or pool accounts, even though there are functions available to update these settings.

Recommendation

Consider following the example below.

LUSD_Token_1_0.sol:L636-L637

```
_operator = newOperator;
_isExcludedFromLimits[newOperator] = true;
```

to

```
_isExcludedFromLimits[_operator] = false;
_operator = newOperator;
_isExcludedFromLimits[newOperator] = true;
```

Status

This issue has been resolved by the team.

11. Insecure access control for _hasLimits()

Severity: Low	Category: Access Control
Target: - DOD_Token_2_1_03.sol - LUSD_Token_1_0.sol	

Description

The _hasLimits function checks whether the transfer sender has limits using tx.origin. Using tx.origin to authenticate is generally not a good practice since it can be abused by malicious contracts when users are interacting with them.

DOD_Token_2_1_03.sol:L529-L538

```
function _hasLimits(address from, address to) internal view returns (bool) {
        return from != owner()
        && to != owner()
        && tx.origin != owner()
        && !_isExcludedFromLimits[from]
        && !_isExcludedFromLimits[to]
        && to != DEAD
        && to != address(0)
        && from != address(this);
}
```

LUSD_Token_1_0.sol:L421-L428

```
function _hasLimits(address from, address to) internal view returns (bool) {
        return tx.origin != _operator
        && !_isExcludedFromLimits[from]
        && !_isExcludedFromLimits[to]
        && to != DEAD
        && to != address(0)
        && from != address(this);
}
```

Recommendation

Consider changing tx.origin to msg.sender.

Status

This issue has been resolved by the team.

12. No need to authenticate the caller in view functions

Severity: Low	Category: Data Validation
Target: - LUSD_Token_1_0.sol	

Description

There is no need to authenticate the caller in the getAllLiqPoolsData and getTreasury function.

LUSD_Token_1_0.sol:L397

```
function getAllLiqPoolsData() external view onlyOperator returns (LPool[] memory)
```

LUSD_Token_1_0.sol:L440

```
function getTreasury() external view onlyOperator returns (address)
```

Recommendation

Consider removing the onlyOperator modifier.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

13. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.0;
pragma solidity >=0.8.0;
pragma solidity >=0.8.0 <0.9.0;
```

The Day of Defeat codebase uses a floating compiler version.

Using a floating pragma statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been resolved by the team.

14. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - DOD_Token_2_1_03.sol - LUSD_Token_1_0.sol	

Description

The Legacy_Stable_1_0 contract uses a custom function setNewOperator which is a simple mechanism to transfer the ownership not supporting a two-step transfer ownership pattern. This simpler mechanism can be useful for quick tests, but projects with production concerns are likely to outgrow it. Transferring ownership is a critical operation and this could lead to transferring it to an inaccessible wallet or renouncing the ownership, e.g. mistakenly.

The DOD_Token_2_1_03 contract inherits from the Ownable contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

It is recommended to implement a two-step transfer of ownership mechanism where the ownership is transferred and later claimed by a new owner to confirm the whole process and prevent lockout. This functionality is implemented in the Ownable2Step contract provided by OpenZeppelin.

Status

This issue has been resolved by the team.

15. Inconsistent use of router address in event emission

Severity: Informational	Category: Logging
Target: - DOD_Token_2_1_03.sol	

Description

In the setNewLiquidityPool function, the address lpCA is obtained via V2ROUTER. However, there is an issue in the event emission where V3ROUTER is mistakenly used instead of V2ROUTER. It leads to a mismatch between the address used for emitting an event and the actual address used in the function.

DOD_Token_2_1_03.sol:L314-L316

```
lpCA = IFactoryV2( IDexRouterV2(V2ROUTER).factory() ).getPair( _LPTargetCoinCA,
address(this) );
require(lpCA == address(0) && !isLiqPool[lpCA], "StudioL_Token: Pair already exists!");
lpCA = IFactoryV2( IDexRouterV2(V2ROUTER).factory() ).createPair( _LPTargetCoinCA,
address(this) );
```

DOD_Token_2_1_03.sol:L334

```
emit NewLPCreated(V3ROUTER, lpCA, _LPTargetCoinCA);
```

Recommendation

It is recommended to replace "V3ROUTER" with "V2ROUTER" when emitting the event.

Status

This issue has been resolved by the team.

16. Unexpected revert if the total supply is 0

Severity: Informational	Category: Code Quality
Target: - LUSD_Token_1_0.sol	

Description

Calls to the totalSupply and decimals functions will be reverted when _tTotal is equal to 0.

LUSD_Token_1_0.sol:L166-L167

```
function totalSupply() external view override returns (uint256) { if (_tTotal == 0) {
revert(); } return _tTotal; }
function decimals() external view override returns (uint8) { if (_tTotal == 0) {
revert(); } return _decimals; }
```

Recommendation

It is recommended that totalSupply() returns 0 and decimals() returns the actual decimals when _tTotal is equal to 0 in order to comply with the ERC20 standard.

Status

This issue has been resolved by the team.

17. Defined but unused events

Severity: Informational	Category: Logging
Target: - DOD_Token_2_1_03.sol	

Description

In the DOD_Token_2_1_03 contract, the ETHWithdrawn and StuckTokensWithdrawn events are defined but not used.

DOD_Token_2_1_03.sol:L48-L49

```
event ETHWithdrawn(address Withdrawer, address Recipient, uint256 ETHamount);
event StuckTokensWithdrawn(address Withdrawer, address Recipient, uint256 TokenAmount);
```

Recommendation

According to the definition, consider emitting the event in the corresponding part of the rescueStuckAssets function.

Status

This issue has been resolved by the team. The team has removed the above two events.

18. Inappropriate error message

Severity: Informational	Category: Code Quality
Target: - LUSD_Token_1_0.sol	

Description

Since there is both mint and burn logic in the _supplyControl function, the error message only mentions the burning logic, which is inappropriate.

LUSD_Token_1_0.sol:L506

```
require(account != address(0), "ERC20: burn from the zero address");
```

Recommendation

It is recommended to set a more appropriate error message.

Status

This issue has been resolved by the team.

19. Missing zero address check

Severity: Informational	Category: Data Validation

Target:
- DOD_Token_2_1_03.sol
- LUSD_Token_1_0.sol

Description

It is considered a security best practice to verify addresses against the zero address in the constructor or setting. However, this precautionary step is absent for the variables highlighted below.

1. DOD_Token_2_1_03.sol:L154-L183

```
constructor (
        address _genesis,
        address _marketing,
        address _transition
    ) {
        ...

        marketingWallet = _marketing;
        _isExcludedFromFees[ _marketing ] = true;
        _isExcludedFromLimits[ _marketing ] = true;

        transitionCollector = _transition;
        _isExcludedFromFees[ _transition ] = true;
        _isExcludedFromLimits[ _transition ] = true;

        ...
}
```

2. DOD_Token_2_1_03.sol:L366-L376

```
function setNewCEXLiquidityPool(address lpCA) external onlyOwner {

        liqPoolList[ address(this) ].push( LPool( lpCA, address(0), false, true, 0, 0,
0, 0 ) );

        ...
}
```

3. DOD_Token_2_1_03.sol:L477-L489

```
function setStakingAndReferralPool(address _staking, address _referral) external
onlyOwner {

        emit StakingPoolUpdated(_msgSender(), stakingPool, _staking);

        stakingPool = _staking;
        ...

        emit ReferralPoolUpdated(_msgSender(), referralPool, _referral);
        referralPool = _referral;
        ...
}
```

4. DOD_Token_2_1_03.sol:L491-L502

```
function setMarketingWalletAndTransitionCollector(address _marketing, address
```

```
_collector) external onlyOwner {

        emit MarketingWalletUpdated(_msgSender(), marketingWallet, _marketing);
        marketingWallet = _marketing;
        ...

        emit PrizeTaxCollectorUpdated(_msgSender(), transitionCollector, _collector);
        transitionCollector = _collector;
        ...
}
```

5. DOD_Token_2_1_03.sol:L688-L692

```
function setSacrificeCA(address _ca) external onlyOwner {
        require(sacrificeCA != _ca, "StudioL: already set to the desired address");
        emit SacrificeCASet( _msgSender(), sacrificeCA, _ca);
        sacrificeCA = _ca;
}
```

6. DOD_Token_2_1_03.sol:L698-L702

```
function setReferralCA(address _ca) external onlyOwner {
        require(referralCA != _ca, "StudioL: already set to the desired address");
        emit ReferralCASet( _msgSender(), referralCA, _ca);
        referralCA = _ca;
}
```

7. LUSD_Token_1_0.sol:L289-L311

```
function setDexRouterTradingStatus(address _router, bool _switch, uint32
pauseTimeInSecs) external onlyOperator {
        if(_switch) {
            require(!enableAggregateDex[_router], "StudioL_Token: trading already
enabled.");

            enableAggregateDex[_router] = true;

            ...

            emit DexRouterEnabled(_msgSender(), _router, block.number, block.timestamp);
        } else {
            ...
        }
}
```

8. LUSD_Token_1_0.sol:L433-L438

```
function setTreasury(address _treasury) external onlyOperator {
        require(treasury != _treasury, "StudioL_Token: Already set to the desired
value");
        emit TreasurySet( _msgSender(), treasury, _treasury);
        treasury = _treasury;
        _isExcludedFromLimits[treasury] = true;
}
```

9. LUSD_Token_1_0.sol:L633-L646

```
function setNewOperator(address newOperator) external onlyOperator {
        require(newOperator != _operator, "StudioL_Token: This address is already the
operator!");
        emit OperatorTransferred(_operator, newOperator);
        _operator = newOperator;
        _isExcludedFromLimits[newOperator] = true;

        ...
}
```

Recommendation

Consider adding zero-address checks for the above-mentioned variables.

Status

This issue has been resolved by the team.

20. Redundant variables

Severity: Informational	Category: Redundancy
Target: - DOD_Token_2_1_03.sol - LUSD_Token_1_0.sol	

Description

Redundant variables in the contract may serve no essential purpose in the contract's functionality, consume unnecessary storage space and increase the gas cost of transactions and deployment. The variables mentioned below are not used in contracts or some parts of them are redundant.

1. DOD_Token_2_1_03.sol:L59

```
uint256 private _circSupply;
```

2. DOD_Token_2_1_03.sol:L68

```
address constant public V3ROUTER = 0x9a489505a00cE272eAa5e07Dba6491314CaE3796;
```

There is no V3 pool created in the DOD_Token_2_1_03 contract.

3. DOD_Token_2_1_03.sol:L74

```
mapping (address => LPool[]) private liqPoolList;
```

The contract only uses address(this) to assign to the mapping key. Consider using LPool[] instead of this mapping.

4. LUSD_Token_1_0.sol:L63

```
uint16 constant DIVISOR = 10000;
```

5. LUSD_Token_1_0.sol:L132

```
address private governor;
```

6. LUSD_Token_1_0.sol:L86

```
mapping(address => uint256) private dexRoutersIndex;
```

7. LUSD_Token_1_0.sol:L108

```
struct LPool {
        ...
        bool launched;
        ...
}
```

Recommendation

Consider removing redundant variables or replacing redundant parts of variables with simpler forms.

Status

This issue has been resolved by the team.

21. Redundant code

Severity: Informational	Category: Redundancy

Target:
- DOD_Token_2_1_03.sol
- LUSD_Token_1_0.sol

Description

DOD_Token_2_1_03.sol:L207

```
function totalSupply() external pure override returns (uint256) { if (genesisTotalSupply == 0) { revert(); } return genesisTotalSupply; }
```

DOD_Token_2_1_03.sol:L215

```
function decimals() external pure override returns (uint8) { if (genesisTotalSupply == 0) { revert(); } return _decimals; }
```

The totalSupply() and decimals() functions contain unnecessary checks for the genesisTotalSupply constant. The genesisTotalSupply is already defined as a non-zero constant, making the additional checks redundant.

DOD_Token_2_1_03.sol:L285

```
function isContract(address account) internal view returns (bool) {
```

The isContract function is never called in this contract.

DOD_Token_2_1_03.sol:L512

```
function setExcludedFromFees(address account, bool _switch) external onlySacrificeAuthorized returns (bool) {
    _isExcludedFromFees[account] = _switch;
    return true;
}
```

In the setExcludedFromFees function, the return statement is unnecessary since the function consistently returns a value of true.

DOD_Token_2_1_03.sol:L529-L538

```
function _hasLimits(address from, address to) internal view returns (bool) {
    return from != owner()
        && to != owner()
        && tx.origin != owner()
        && !_isExcludedFromLimits[from]
        && !_isExcludedFromLimits[to]
        && to != DEAD
        && to != address(0)
        && from != address(this);
}
```

LUSD_Token_1_0.sol:L505

```
function _hasLimits(address from, address to) internal view returns (bool) {
    return tx.origin != _operator
        && !_isExcludedFromLimits[from]
        && !_isExcludedFromLimits[to]
        && to != DEAD
```

```
        && to != address(0)
        && from != address(this);
}
```

The _hasLimits function contains a redundant condition. When from is equal to address(this), the !_isExcludedFromLimits[from] check would already evaluate to false.

LUSD_Token_1_0.sol:L505

```
function _supplyControl(address account, uint256 amount, bool mintORburn) private
returns (bool) {
```

In the _supplyControl function, the return statement is unnecessary since the function consistently returns a value of true.

LUSD_Token_1_0.sol:L501

```
function supplyControl(address account, uint256 amount, bool mintORburn) external
onlyTreasury nonReentrant returns (bool) {
```

In the supplyControl function, the nonReentrant modifier can be removed due to the absence of external calls, and there is no requirement to import ReentrancyGuard.sol.

LUSD_Token_1_0.sol:L140-L143

```
_operator = _msgSender();

_isExcludedFromLimits[_msgSender()] = true;
_isExcludedFromLimits[_operator] = true;
```

LUSD_Token_1_0.sol:L256-L259

```
_allowances[ _msgSender() ][_router] = type(uint256).max;
_allowances[_router][ _msgSender() ] = type(uint256).max;
_allowances[ _operator ][_router] = type(uint256).max;
_allowances[_router][ _operator ] = type(uint256).max;
```

In these assignment operations, it can be observed that _msgSender() is effectively the same as the _operator variable, leading to redundant assignment operations.

Recommendation

Consider removing the redundant code.

Status

This issue has been resolved by the team.

22. Gas Optimization Suggestions

Severity: Informational	Category: Gas Optimization

Target:
- DOD_Token_2_1_03.sol
- LUSD_Token_1_0.sol

Description

Some changes can be made to improve gas consumption:

DOD_Token_2_1_03.sol:L277

```
function multiSendTokens(address[] memory accounts, uint256[] memory amountsInWei)
```

DOD_Token_2_1_03.sol:L396

```
function pauseTradeByPool(address[] memory poolCA, bool pauseAllPools, uint32
pauseTimeInSecs)
```

LUSD_Token_1_0.sol:L224

```
function multiSendTokens(address[] memory accounts, uint256[] memory amountsInWei)
```

These functions only require read access of the input data, using calldata instead of memory can access function arguments directly from the input data, without the need to create a separate copy in memory.

DOD_Token_2_1_03.sol:L298-302, L306-L310

```
_isExcludedFromFees[V2ROUTER] = true;
_isExcludedFromLimits[V2ROUTER] = true;
...
_allowances[ address(this) ][V2ROUTER] = type(uint256).max;
_allowances[V2ROUTER][ address(this) ] = type(uint256).max;

_allowances[ address(this) ][V3ROUTER] = type(uint256).max;
_allowances[V3ROUTER][ address(this) ] = type(uint256).max;
```

It is recommended to move these constant assignments to the constructor function.

LUSD_Token_1_0.sol:L457-L499

```
function syncNativeANDStablePairs(bool _syncNativeCoin, bool _syncStableCoin) public {
        for(uint8 i = 0; i < liqPoolList[ address(this) ].length; i++) {
                ...
                amounts = IDexRouterV2(LiqPool.dexCA).getAmountsOut(ourPathCoin0Balance,
refPath);
                ...
                if( pairedCoinBalance > _tOwned[poolAddr] ) {
                ...
        }
}
```

Highlighted variables above can be stored in temporary variables for further processing because reading in storage will cost more than reading in memory.

Recommendation

Consider applying the gas optimizations where needed.

Status

This issue has been resolved by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files provided by the client:

File	SHA-1 hash
LUSD_Token_1_0.sol	fe78cc60d533d5b2168aea39bb4b795de9c50d4d
interface/IDexRouterV2.sol	5cf63b36afdcf89c52870285bf7c9d9fdff3f6d0
interface/IPairV2.sol	4e7007fd092ea01fac269ea5864bbb6db8f33c86
interface/IFactoryV2.sol	893de6e7d1ba380b67dd3695bb287fbeb7cb8118
interface/IFactoryV3.sol	65ec834742812e09c55b33860769ddbeed9e524f
DOD_Token_2_1_03.sol	335311840c97f5070cdf8118644f48b912345fdc

After resolving the issues, the contracts are flattened and deployed to the addresses listed below:

Contract	Address
Legacy_Stable_1_02	0x7e18dabfb0eC86C08749a85752d45Bcf6B4aceFf
DOD_Token_2_1_05	0x0e9729a1Db9E45FF08F64E6C4342Be3921e993e0