verichains

SECURITY AUDIT OF

LUNA RUSH SMART CONTRACTS

Public Report

Jan 12, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Jan 12, 2022. We would like to thank the Luna Rush for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Luna Rush Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Luna Rush Smart Contracts

Luna Rush is an idle RPG game based on blockchain technology. You can fight other players, team up with friends, win a tournament and earn money with your strategy and luck.

Summon your Warriors, TRAIN them to become powerful heroes, or convert them into Spirit material for EVOLVING.

Luna Rush is also a multiplayer RPG NFT GAME that lets the user engage in the combat arena and profit from battles.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Luna Rush Smart Contracts.

It was conducted on commit 340c97354a73253389a43267adfd29e62672da17 from git repository *https://bitbucket.org/lunarush/contracts*.

The latest version of the following files were made available in the course of the review:

SHA256 SUM	FILE
8b32ccf259d4a5b97d1f645ffa54d35deb5ad386e97decdf964d7a8198c570f7	ILUWADesign.sol
33a32094b2d386d57ab48f6b5fa5b58b3c8ba8867c56ea082a43c34c070dea57	ILUWAStats.sol
145471c739c35986bbd90f68e6b86197fb18065ad17cbc3a28abd90557106a22	ILUWAToken.sol
31f686d4866eb99d732f22054d10b811a8070c09410f33f62dffbd44ccfc30d4	LUWABox.sol
8dcddc7444938d39a8c4811effbb46bfb1245025c6247769ebbd92e9e9c98a87	LUWABoxDetails.sol
b2767e53ba351557dede8d2d7aa02853504cce39bd814e4fdba7eb29b56e958f	LUWADesign.sol
32ae067f4f17836613c37003df698a33c9fe8dfc51e21ad76d8de5bf19535cb0	LUWADetails.sol
c7520a6da774295ff500bb13baccf90b4e1c979521fe1becd7e3956ec969b757	LUWAToken.sol
c9e49b247c6c1bb37542b5b9ab2c536274fe928e6dd00c395737426a6d76600f	Utils.sol
6405a854c8e307ca0af0f3ac35f6caf6130dec2bed87f2891e0da3c58a328acb	HikariStats.sol
c5dc7b1a485ecff750802c995d00e016f540610143358bf369807335472ffee7	HitomiStats.sol
226b6c24f93d3cf2b6504ae3f48247318b73805099a3d9bfce914c7f5895b99e	KaiyoStats.sol

SHA256 SUM	FILE
642f317971a17d63ee0f322128692b53d3083303ae4825443ad7ad9e6fbb7466	KinuStats.sol
64d1fd3c95b97e174de0c678e0e82d39b7f68f16b4c908852c24e88e2953c4a7	MatsukoStats.sol
bd14b32423d4956f9cfef290f6526228b50c4f8d0c79ac5569caf68373868427	NishiStats.sol
03b7e1660a0139c505eef4947ac60728bd240296500e9b4d3e48595a7485ed0c	ReikoStats.sol
73921ed00e99a6a0797a87ce23a4a6bd830cf6ddbdcf15d0817f0512dbb97b40	SumikoStats.sol
95df05ad07faf9b494a334628f3e1d937702831484a11baea0f14a0b7e928d3c	UmiStats.sol
5241ce4c231af3978818d64bde33153510b6ebd9e2a314758a997ec2d79aa5bf	YoutaStats.sol
9bb1f34ba2f88b556e54ce77d7cac7100627b0ec2b6a505973c4a9ed5057dc10	YuukaStats.sol

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

Luna Rush Smart Contracts contains 3 main contracts: LUWAToken, LUWABox and LUWADesign.

2.1.1. LUWAToken contract

This is the upgradable NFT contract in the Luna Rush Smart Contracts, which extends ERC721Upgradeable, AccessControlUpgradeable, PausableUpgradeable, UUPSUpgradeable and OwnableUpgradeable contracts. With OwnableUpgradeable, by default, Token Owner is contract deployer but he can transfer ownership to another address at any time. AccessControlUpgradeable allows the contract to implement role-based access control mechanisms which let DEFAULT_ADMIN_ROLE (contract deployer by default) sets any role for anyone.

Users can purchase NFT (only Small Box and Big Box type) with LUS token. Box can be opened after 5 blocks from the time of purchase by calling processTokenRequests and will become a NFT hero with random rarity. NFT hero can be transferred between addresses, each address can not hold more NFT than the limit set in LUWADesign.

Addresses with DESIGNER_ROLE can mint any boxes (any type) to anyone by calling safeMint.

2.1.2. LUWABox contract

This is the upgradable box selling contract in the Luna Rush Smart Contracts, which extends ERC721Upgradeable, AccessControlUpgradeable, PausableUpgradeable, UUPSUpgradeable and OwnableUpgradeable contracts. With OwnableUpgradeable, by default, Token Owner is contract deployer but he can transfer ownership to another address at any time. AccessControlUpgradeable allows the contract to implement role-based access control mechanisms which let DEFAULT_ADMIN_ROLE (contract deployer by default) sets any role for anyone. DEFAULT_ADMIN_ROLE can withdraw all LUS tokens and BUSD tokens from this contract.

This contract is used to sell Mega Box (by BUSD tokens) which requires users to hold an amount of LUS tokens to have permission to buy. The holding amount is 500 tokens by default but can be changed anytime by DESIGNER_ROLE. This sale can be disabled/enabled anytime by DESIGNER_ROLE. There is also a sale for whitelisted users (whitelist is set by DESIGNER_ROLE), this sale can not be disabled and does not require users to hold LUS.

Users can choose to open box to get a NFT hero (after 5 blocks from the time of opened by calling processTokenRequests) or sell their unopened box to marketplace where others can buy box by LUS with a small marketplace tax.

2.1.3. LUWADesign contract

This is the upgradable NFT logic contract in the Luna Rush Smart Contracts, which extends AccessControlUpgradeable and UUPSUpgradeable contracts. AccessControlUpgradeable allows the contract to implement role-based access control mechanisms which let DEFAULT_ADMIN_ROLE (contract deployer by default) sets any role for anyone.

This contract hold all the logic of NFT like drop rate of each box type, mint cost, upgrade cost, hero stats, hero limit, RNG which can be updated by DESIGNER_ROLE at any time. It also supports bring hero offchain (deposit heroes to the game) and onchain (withdraw heroes back and update their stats). Offchain heroes can not be transferred.

2.2. Findings

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

Luna Rush fixed the code according to Verichains's draft report in commit e3593aefd7abbbdedcf45124d366c51bb03168fd.

2.2.1. LUWABox.sol - Front running in buy function CRITICAL

Attackers can list an item by sale with low price and wait for a user buy it with buy. While waiting, attackers listen for pending transactions and when a user buy that item, they send two transactions deactiveSale and sale with higher gas price (higher gas price transaction is usually mined first) than the buy transaction to relist the item to a with a higher price. The result is user buy the item with more money than he saw in the marketplace.

For example: Attacker put an item with 1 LUS in marketplace, user see it and make a transaction to buy it with 1 LUS. Attacker listen to pending transactions and know that someone is buying the item with 1 LUS and send two transactions to relist the item with price of 1000 LUS with higher gas price and get mined before the buy transaction. The result is user lost 1000 LUS to attacker for that item.

```
function sale(uint256 tokenId, uint256 boxPriceLUS) external {
    address to = msg.sender;
    require(ownerOf(tokenId) == to, "Token not owned");
    LUWABoxDetails.BoxDetails storage boxDetail = tokenDetails[tokenId];
    require(boxDetail.is_opened == 0, "Box already opened");
    boxDetail.price = boxPriceLUS * (10**18);
```

```
    boxDetail.on_market = 1;
    // Market hole token for sale
    transferFrom(to, address(this), tokenId);
    emit Sale(to, tokenId, boxDetail.price);
}
function deactiveSale(uint256 tokenId) external {
    address to = msg.sender;
    LUWABoxDetails.BoxDetails storage boxDetail = tokenDetails[tokenId];
    require(boxDetail.owner_by == to, "Token not owned");
    require(boxDetail.is_opened == 0, "Box already opened");
    require(boxDetail.on_market == 1, "Box already off chain");
    boxDetail.on_market = 0;
    tokenDetails[tokenId] = boxDetail;
    this.approve(to, tokenId);
    transferFrom(address(this), to, tokenId);
    emit DeactiveSale(to, tokenId, boxDetail.price);
}
function buy(uint256 tokenId) external {
    address to = msg.sender;
    require(tokenIds[to].length + 1 <= boxLimit, "User limit reached");
    LUWABoxDetails.BoxDetails memory boxDetail = tokenDetails[tokenId];
    require(boxDetail.is_opened == 0, "Box already opened");
    require(boxDetail.on_market == 1, "Box not on chain for marketplace");
    require(
        boxDetail.price <= coinToken.balanceOf(to),
        "User need hold enough LUS to buy this box"
    );
    // Total fee
    uint256 fee = marketFee(boxDetail.price);
    // Fee for market
    coinToken.transferFrom(to, address(this), fee);
    // Fee for the owner
    coinToken.transferFrom(to, boxDetail.owner_by, boxDetail.price - fee);
    // Market hole token for sale
    //require(1<= 0, "Box already opened here");
    this.approve(to, tokenId);
    transferFrom(address(this), to, tokenId);
    emit Buy(to, tokenId, boxDetail.price, boxDetail.owner_by);
}
```

RECOMMENDATION

Adding price parameter to buy function to revert when box price is higher than the price user want to buy.

```
function buy(uint256 tokenId, uint256 price) external {
    address to = msg.sender;
    require(tokenIds[to].length + 1 <= boxLimit, "User limit reached");
    LUWABoxDetails.BoxDetails memory boxDetail = tokenDetails[tokenId];
    require(boxDetail.is_opened == 0, "Box already opened");
    require(boxDetail.on_market == 1, "Box not on chain for marketplace");
    require(price >= boxDetail.price, "Buy price is too low");
    ...
}
```

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged and fixed by the Luna Rush team.

2.2.2. LUWABox.sol - Mint more than maxBox LOW

In mint and whitelistMint functions, require(tokenIdCounter.current() <= maxBox, "Box sold out"); requires not to mint more than maxBox, but it's still can mint pass the maxBox.

```
function mint(uint256 count) external {
    require(count > 0, "No token to mint");
    require(tokenIdCounter.current() <= maxBox, "Box sold out");
    // Check limit.
    address to = msg.sender;
    require(tokenIds[to].length + count <= boxLimit, "User limit reached"…
  );
    ...
}
function whitelistMint(uint256 count) external onlyRole(WHITELIST_ROLE) {
    require(count > 0, "No token to mint");
    require(tokenIdCounter.current() <= maxBox, "Box sold out");
    address to = msg.sender;
    // Check limit.
    require(tokenIds[to].length + count <= boxLimit, "User limit reached"…
  );
    ...
}
```

RECOMMENDATION

Adding + count to the require.

```
function mint(uint256 count) external {
    require(count > 0, "No token to mint");
    require(tokenIdCounter.current()  + count <= maxBox, "Max box reached…
  ");
    // Check limit.
    address to = msg.sender;
    require(tokenIds[to].length + count <= boxLimit, "User limit reached"…
  );
    ...
}

function whitelistMint(uint256 count) external onlyRole(WHITELIST_ROLE) {
    require(count > 0, "No token to mint");
    require(tokenIdCounter.current()  + count <= maxBox, "Box sold out");…

    address to = msg.sender;
    // Check limit.
    require(tokenIds[to].length + count <= boxLimit, "User limit reached"…
  );
    ...
}
```

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged and fixed by the Luna Rush team.

2.2.3. LUWABox.sol - Redundant approve INFORMATIVE

In deactiveSale and buy functions, this.approve(to, tokenId); is redundant because it is unnecessary while transfer by owner and ERC721Upgradeable.__transfer already clear approvals from previous owner.

```
function deactiveSale(uint256 tokenId) external {
    address to = msg.sender;
    LUWABoxDetails.BoxDetails storage boxDetail = tokenDetails[tokenId];
    require(boxDetail.owner_by == to, "Token not owned");
    require(boxDetail.is_opened == 0, "Box already opened");
    require(boxDetail.on_market == 1, "Box already off chain");
    boxDetail.on_market = 0;
    tokenDetails[tokenId] = boxDetail;
    this.approve(to, tokenId);
    transferFrom(address(this), to, tokenId);
    emit DeactiveSale(to, tokenId, boxDetail.price);
```

```
}
function buy(uint256 tokenId) external {
    address to = msg.sender;
    require(tokenIds[to].length + 1 <= boxLimit, "User limit reached");
    LUWABoxDetails.BoxDetails memory boxDetail = tokenDetails[tokenId];
    require(boxDetail.is_opened == 0, "Box already opened");
    require(boxDetail.on_market == 1, "Box not on chain for marketplace");
    require(
        boxDetail.price <= coinToken.balanceOf(to),
        "User need hold enough LUS to buy this box"
    );
    // Total fee
    uint256 fee = marketFee(boxDetail.price);
    // Fee for market
    coinToken.transferFrom(to, address(this), fee);
    // Fee for the owner
    coinToken.transferFrom(to, boxDetail.owner_by, boxDetail.price - fee);
    // Market hole token for sale
    //require(1<= 0, "Box already opened here");
    this.approve(to, tokenId);
    transferFrom(address(this), to, tokenId);
    emit Buy(to, tokenId, boxDetail.price, boxDetail.owner_by);
}
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(ERC721Upgradeable.ownerOf(tokenId) == from, "ERC721: transfer…
   of token that is not own");
    require(to != address(0), "ERC721: transfer to the zero address");
    _beforeTokenTransfer(from, to, tokenId);
    // Clear approvals from the previous owner
    _approve(address(0), tokenId);
    _balances[from] -= 1;
    _balances[to] += 1;
    _owners[tokenId] = to;
    emit Transfer(from, to, tokenId);
}
```

RECOMMENDATION

Removing the approval.

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged by the Luna Rush team.

2.2.4. LUWABox.sol - Redundant storage variable reassign INFORMATIVE

When contract add a local storage variable inside function, assignments from storage to this local storage variable only assign a reference so if contract modify it, the changes will reflect to the storage. No need to reassign to storage.

For example, in deactiveSale function, boxDetail is a pointer pointing to tokenDetails[tokenId], so boxDetail.on_market = 0 is the same as tokenDetails[tokenId].on_market = 0 and there is no need of tokenDetails[tokenId] = boxDetail;.

```
function deactiveSale(uint256 tokenId) external {
    address to = msg.sender;
    LUWABoxDetails.BoxDetails storage boxDetail = tokenDetails[tokenId];
    require(boxDetail.owner_by == to, "Token not owned");
    require(boxDetail.is_opened == 0, "Box already opened");
    require(boxDetail.on_market == 1, "Box already off chain");
    boxDetail.on_market = 0;
    tokenDetails[tokenId] = boxDetail;
    this.approve(to, tokenId);
    transferFrom(address(this), to, tokenId);
    emit DeactiveSale(to, tokenId, boxDetail.price);
}
```

RECOMMENDATION

Removing storage variable reassign in the whole contract.

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged and fixed by the Luna Rush team.

2.2.5. LUWABox.sol - Redundant code INFORMATIVE

In openBox function, contract sets boxDetail.on_market = 0 is redundant because it's already 0 by require(boxDetail.on_market == 0, "Box not off chain for open");.

```
function openBox(uint256 tokenId) external {
    address to = msg.sender;
    LUWABoxDetails.BoxDetails storage boxDetail = tokenDetails[tokenId];
```

```
    require(boxDetail.owner_by == to, "Token not owned");
    require(boxDetail.is_opened == 0, "Box already opened");
    require(boxDetail.on_market == 0, "Box not off chain for open");
    boxDetail.is_opened = 1;
    boxDetail.on_market = 0;
    tokenDetails[tokenId] = boxDetail;
    // Call LUWAToken to random hero
    luwaToken.openBox(to, 1, boxDetail.box_type);
    emit OpenBox(to, tokenId);
}
```

RECOMMENDATION

Removing redundant assign.

```
function openBox(uint256 tokenId) external {
    address to = msg.sender;
    LUWABoxDetails.BoxDetails storage boxDetail = tokenDetails[tokenId];
    require(boxDetail.owner_by == to, "Token not owned");
    require(boxDetail.is_opened == 0, "Box already opened");
    require(boxDetail.on_market == 0, "Box not off chain for open");
    boxDetail.is_opened = 1;
    // Call LUWAToken to random hero
    luwaToken.openBox(to, 1, boxDetail.box_type);
    emit OpenBox(to, tokenId);
}
```

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged and fixed by the Luna Rush team.

2.2.6. LUWABox.sol - Hardcode decimals for tokens INFORMATIVE

The contract is hardcoding decimals for tokens.

```
boxPriceBUSD = 2 * (10**18);
boxDetail.price = (1000 * (10**18));
boxDetail.price = boxPriceLUS * (10**18);
```

RECOMMENDATION

Consider using IERC20Metadata to get token decimals for calculating or add constant variables for LUSDecimal and BUSDDecimal.

```
uint public constant LUS_DECIMALS = 10 ** 18;
uint public constant BUSD_DECIMALS = 10 ** 18;
boxPriceBUSD = 2 * BUSD_DECIMALS;
boxDetail.price = 1000 * LUS_DECIMALS;
boxDetail.price = boxPriceLUS * LUS_DECIMALS;
```

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged and fixed by the Luna Rush team.

2.2.7. LUWABox.sol, LUWADesign.sol - Gas optimize INFORMATIVE

In external function, use memory argument will force Solidity to copy it to memory which will cost more gas than using from calldata especially when passing large readonly array.

RECOMMENDATION

Consider changing memory to calldata in external function for gas saving.

For example:

```
// function setWhitelist(address[] memory addresses) external onlyRole(DE…
  SIGNER_ROLE) {
function setWhitelist(address[] calldata addresses) external onlyRole(DES…
  IGNER_ROLE) {
    for (uint256 i = 0; i < addresses.length; ++i) {
        _setupRole(WHITELIST_ROLE, addresses[i]);
    }
}
```

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged and fixed by the Luna Rush team.

2.2.8. LUWADesign.sol - heroEncoded is not verified HIGH

In ownerOnChain function, Users can update their NFT's stats to anything if enableOwnerOnChain is enabled somehow.

```
function ownerOnChain(uint256 tokenId, uint256 heroEncoded) external {
    require(enableOwnerOnChain == 1, "Function was not support now");
    address owner = msg.sender;
    require(nftToken.ownerOf(tokenId) == owner, "Token not owned");
    LUWADetails.Details memory details = LUWADetails.decode(heroEncoded);
    details.is_onchain = LUWADetails.ON_CHAIN;
```

```
    luwaDetails[tokenId] = details.encode();
    emit OwnerOnChain(tokenId, luwaDetails[tokenId]);
}
```

RECOMMENDATION

Removing this function and add it back with verify proof from backend server when it's ready.

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged and fixed by the Luna Rush team.

2.2.9. LUWAToken.sol - Attackers can choose best seed to mint CRITICAL

In processTokenRequests, if block.number - 256 > request.targetBlock, seed will be 0 and seed will be recalculated by blockhash of block in the past which is already known. Attackers can now control seed and calculate the best result for them.

```
function processTokenRequests() external {
    ...
    CreateTokenRequest storage request = requests[i - 1];
    uint256 targetBlock = request.targetBlock;
    require(block.number > targetBlock, "Target block not arrived");
    uint256 seed = uint256(blockhash(targetBlock));
    if (seed == 0) {
        targetBlock =
        (block.number & maskFirst248Bits) +
        (targetBlock & maskLast8Bits);
        if (targetBlock >= block.number) {
            targetBlock -= 256;
        }
        seed = uint256(blockhash(targetBlock));
    }
    ...
}
```

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged and fixed by the Luna Rush team.

2.2.10. LUWAToken.sol - Redundant variable nextSeed INFORMATIVE

In createToken, nextSeed variable is assigned only, not use for anything.

```
function createToken(
    address to,
    uint256 count,
    uint256 rarity,
    uint256 seed,
    uint256 boxType
) internal {
    uint256 nextSeed;
    for (uint256 i = 0; i < count; ++i) {
        uint256 id = tokenIdCounter.current();
        uint256 tokenSeed = uint256(keccak256(abi.encode(seed, id)));
        nextSeed =  design.createRandomToken(tokenSeed, id, rarity, boxTy…
  pe);
        tokenIdCounter.increment();
        tokenDetails[id] = id;
        _safeMint(to, id);
        emit TokenCreated(to, id, id);
    }
}
```

RECOMMENDATION

Consider removing nextSeed variable.

```
function createToken(
    address to,
    uint256 count,
    uint256 rarity,
    uint256 seed,
    uint256 boxType
) internal {
    for (uint256 i = 0; i < count; ++i) {
        uint256 id = tokenIdCounter.current();
        uint256 tokenSeed = uint256(keccak256(abi.encode(seed, id)));
        design.createRandomToken(tokenSeed, id, rarity, boxType);
        tokenIdCounter.increment();
        tokenDetails[id] = id;
        _safeMint(to, id);
        emit TokenCreated(to, id, id);
    }
}
```

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged by the Luna Rush team.

2.2.11. LUWAToken.sol, LUWABox.sol - Unused PausableUpgradeable INFORMATIVE

The contracts extend PausableUpgradeable and have pause/unpause functions but they are not used anywhere.

```
function pause() public onlyRole(PAUSER_ROLE) {
    _pause();
}

function unpause() public onlyRole(PAUSER_ROLE) {
    _unpause();
}
```

RECOMMENDATION

Consider removing unused PausableUpgradeable or implement a logic to pause the contract.

UPDATES

- *Dec 30, 2021*: This issue has been acknowledged by the Luna Rush team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 31, 2021*	Public Report	Verichains Lab
1.1	*Jan 12, 2022*	Public Report	Verichains Lab

Table 2. Report versions history