SAFETIN AUDIT

LUX

December 9th, 2022

TABLE OF CONTENTS

AUDIT SUMMARY

This report was written for LUX in order to find flaws and vulnerabilities in the LUX project's source code, as well as any contract dependencies that weren't part of an officially recognized library.
A comprehensive examination has been performed, utilizing Static Analysis, Manual Review, and LUX Deployment techniques. The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors
- Assessing the codebase to ensure compliance with current best practices and industry standards
- Ensuring contract logic meets the specifications and intentions of the client
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Through line-by-line manual review of the entire codebase by industry expert

AUDIT OVERVIEW

PROJECT SUMMARY

Project name	LUX
Description	On December 11th 2022, 1000 digital collectibles will be put out for purchase at 1K$. Each of these collectibles is attached to a hotel stay and to one of the 6 bespoke travel experiences crafted for this occasion. Gift your loved ones (or yourself) the best and most surprising Christmas present yet, and discover what's inside on Christmas day.
Platform	…
Language	Solidity
Codebase	Lux.sol

FINDINGS SUMMARY

Vulnerability	Total	Resolved
● Critical	0	0
● Major	0	0
● Medium	0	0
● Minor	0	0
● Informational	7	0

AUDIT FINDINGS

Code	Title	Severity
VAR-1	State variables are declared with default values	● Informational
MINT-1	Toggle function for minting could be consolidated	● Informational
FUNT-1	Public getter functions are being declared for state variables	● Informational
PAY-1	Payments could be accepted in stablecoin if fixed price is required	● Informational
INDEX-1	ERC721A standard has an index that iterates on mints	● Informational
INTER-1	Checks effects interactions best practice	● Informational
MESS-1	Message value could be used to send funds in mint function	● Informational

VAR-1 | State variables are declared with default values

Description

State variables such as index, _isMintOn, uri, and mintPartnershipCount are declared with values that are the same as their defaults. When variables are set, they will automatically be set to their default value which in the case of a uint will be 0, a bool will be false, and a string will be an empty string. Due to this reason, if the values for index, _isMintOn, uri, and mintPartnershipCount could be initialized to defaults without declaration.

Recommendation

Declare the types and names for index, _isMintOn, uri, and mintPartnershipCount but do not declare the default values:

```
uint index;
bool _isMintOn;
string uri;
uint mintPartnershipCount;
```

MINT-1 | Toggle function for minting could be consolidated

Description

The toggles functions used to change the value of _isMintOn could be consolidated into a single function that sets the value to the opposite of the current value. This will minimally reduce the gas cost of other write functions depending on the placement of the function in the selector logic but it will minimize code used for a similar purpose.

Recommendation

Consolidate the stopMint and enableMint functions into a single function:

```
function toggleMint() external onlyOwner {
        _isMintOn = !_isMintOn;
}
```

FUNT-1 | Public getter functions are being declared for state variables

Description

Functions have been created to fetch the values for state variables such as index, price, and _isMintOn. By declaring the state variables that are desired to be visible as public, these functions can be removed. Declaring getter functions may be required for interoperability with external contracts but there is no clear reference to this based on the information shared.

PAY-1 | Payments could be accepted in stablecoin if fixed price is required

Description

The NFT price is set in terms of Ether, which due to volatility may require multiple updates from the admin to ensure that it aligns with a desired dollar sale price. If there is a fixed price that the project would like to accept for each of the NFTs then the mint function could be edited to solely accept USDC. However, this would only be recommended if the price that each NFT is being sold at must be in a tightly fixed range in dollar terms.

To do this, the IERC20 interface could be imported using OpenZeppelin library and the mint function would check that the balanceOf() the msg.sender is more than the price. In the example of USDC, if the amount accepted was 100 dollars then the fixed value would be 100000000 (as USDC token is set to 6 decimal places). These funds could then be sent by the contract to the mainWallet, the additional step that would be required on a front-end would be the approval step for the fixed price of the NFT.

INDEX-1 | ERC721A standard has an index that iterates on mints

Description

The standard being used for the collection has a public getter function called totalSupply(). This function returns the currentIndex minus the burnCounter minus the startTokenId - this means there is an active log of the index factoring in burnt tokens. Alternatively, there is also a function called _nextTokenId(), which returns the _currentIndex being iterated upon by the standard.

In theory, this means that the index that is iterated on in the mint and mintPartnership functions could be discarded in exchange for one of the standard existing functions.

INTER-1 | Checks effects interactions best practice

Description

The best practice approach is to check conditions, update effects in the existing contract, and then interact externally. As the non-reentrant modifier is being used, this will prevent any re-entrancy risk in the mint function. However, it would be best practice to iterate the index before the external call is made to send funds to the MAIN_WALLET.

If there are any issues with the funds being sent or tokens being minted then the function will revert and all changes to the contract state will be returned to how they were prior to the contract call.

Recommendation

Iterate the index before the external transaction sending Ethereum:

```
index += quantity;
(bool success, ) = payable(MAIN_WALLET).call{value: address(this).balance)(“ ”);
require(success, “Transfer Failed.”);
```

MESS-1 | Message value could be used to send funds in mint function

Description

Currently, the address balance is sent through the call function in the mint function. However, as there are no receive() or fallback() functions, the contract would never have more Ether than the msg.value amount that is being sent in the mint function - due to the lack of a receive() or fallback() function. Therefore, the value field could use msg.value for the amount being sent.

Recommendation

Change the value field to the msg.value in the mint function:

```
(bool success, ) = payable(MAIN_WALLET).call{value: msg.value)("");
require(success, "Transfer Failed.");
```

Supporting Comments

Assuming the logic, which handles the assignment of the random prizes will be handled off-chain as there is no reference to this in the contract. To ensure that NFTs can not be sniped in an advantageous way by buyers, the project can handle reward assignment off-chain via a web2 platform, release the metadata for all NFTs are the whole collection is minted (a provenance hash approach can be used to prove there is no tampering), or offset the starting index for the metadata linked to each token ID.

If the reward functionality is off-chain then most of the points above should be ignored, as assignment would be through a web2 platform. However, if prizes were going to be assigned on-chain then different approaches for using provenance hashes and offset starting indexes can be found in projects such as Crypto Coven or Bored Ape Yacht Club.

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement.

This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without Safetin's prior written consent.This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Safetin to perform a security assessment.

This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance. This report should not be used in any way

Safetin security assessment to make decisions around investment or involvement with any particular project.

This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Safetin's position is that each company and individual are responsible for their own due diligence and continuous security. Safetin's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or fun.