SALUS SECURITY

FEB 2024

CODE SECURITY ASSESSMENT

LONGTOTEM

Overview

Project Summary

- Name: longtotem
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/LONGTOTEM/longtotem-protocol
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	longtotem
Version	v2
Type	Solidity
Dates	Feb 02 2024
Logs	Feb 02 2024; Feb 02 2024

Vulnerability Summary

Total High-Severity issues	3
Total Medium-Severity issues	1
Total Low-Severity issues	2
Total informational issues	4
Total	10

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 **4**

1.1 About SALUS 4

1.2 Audit Breakdown 4

1.3 Disclaimer 4

Findings **5**

2.1 Summary of Findings 5

2.2 Notable Findings 6

1. Lack of access control in LongFactory.setMintPrice() 6
2. Lack of access control in LongFactory.setStartEnd() 7
3. Incorrect mint amount in LongFactory.invest() 8
4. Centralization risks 10
5. Missing events for functions that change critical state 11
6. Problems with ProxyStrap 12

2.3 Informational Findings 13

7. Should not leave the implementation contract uninitialized 13
8. Incorrect compiler version specified in hardhat.config.js 14
9. Unbounded loop may lead to out-of-gas error 15
10. Redundant code 16

Appendix **17**

Appendix 1 - Files in Scope 17

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	Lack of access control in LongFactory.setMintPrice()	High	Access Control	Resolved
2	Lack of access control in LongFactory.setStartEnd()	High	Access Control	Resolved
3	Incorrect mint amount in LongFactory.invest()	High	Business Logic	Resolved
4	Centralization risks	Medium	Centralization	Acknowledged
5	Missing events for functions that change critical state	Low	Logging	Acknowledged
6	Problems with ProxyStrap	Low	Business Logic	Acknowledged
7	Should not leave the implementation contract uninitialized	Informational	Access Control	Acknowledged
8	Incorrect compiler version specified in hardhat.config.js	Informational	Configuration	Resolved
9	Unbounded loop may lead to out-of-gas error	Informational	Gas	Acknowledged
10	Redundant code	Informational	Redundancy	Acknowledged

2.2 Notable Findings

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

1. Lack of access control in LongFactory.setMintPrice()

Severity: High	Category: Access Control
Target: - contracts/long/LongFactory.sol	

Description

LongFactory.sol:L55-L57

```
function setMintPrice(uint256 _mintPrice) external {
    mintPrice = _mintPrice;
}
```

The setMintPrice function in the LongFactory contract is an external function without proper access control. Anyone can exploit this function to change the mint price.

An attacker can set the mint price to 0 by calling setMintPrice(0), then mint all the supplies from the LongFactory contract without incurring any fees. This could also result in other users being unable to mint since the MAX_SUPPLY in LongFactory has been reached.

Recommendation

It is recommended to add proper access control to the setMintPrice function. For example, the onlyOwner modifier can be included in the function.

Status

This issue has been resolved by the team with commit 841cb0d.

2. Lack of access control in LongFactory.setStartEnd()

Severity: High	Category: Access Control
Target: - contracts/long/LongFactory.sol	

Description

LongFactory.sol:L50-L53

```
function setStartEnd(uint256 _startAt, uint256 _endAt) external {
    startAt = _startAt;
    endAt   = _endAt;
}
```

The setStartEnd function in the LongFactory contract does not have proper access control, which means anyone can change the start time and end time for minting (i.e. invest()) in LongFactory.

An attacker can prevent a user from minting by frontrunning the invest() transaction and calling setStartEnd() to set endAt to 0.

Recommendation

It is recommended to add proper access control to the setStartEnd function. For example, the onlyOwner modifier can be included in the function.

Status

This issue has been resolved by the team with commit 841cb0d.

3. Incorrect mint amount in LongFactory.invest()

Severity: High	Category: Business Logic
Target: - contracts/long/LongFactory.sol	

Description

LongFactory.sol:L59-L92

```
function invest(uint256 amount, address _referralBy) external payable {
    ...

    uint256 mintable = min(amount, MAX_SUPPLY - totalMint);
    require(mintable > 0, "Mint over");
    uint256 mintFee = mintPrice * mintable;

    ...

    userDeposit[msg.sender] +=  mintFee;

    totalMint += mintable;
    IMintInList(longNFT).mintInList(msg.sender, amount);

    ...
}
```

The invest function in the LongFactory contract deducts fees from the user for a `mintable` number of LONG NFTs. However, this function mints an `amount` number of LONG NFTs to the user, which could be more than `mintable`. This issue allows users to mint more NFTs than they are paying for.

An example exploit scenario is as follows (here we ignore the block gas limit just to illustrate the gist of the impact):

1. Notice that the max supply in LongFactory is 250000, and the max supply in LongNFT is 500000
2. Right after the LongNFT and LongFactory are deployed and configured, an attacker calls invest(500000, address(0))
3. The invest function calculates the `mintable` to be 25000, so it charges the user mintPrice * 250000 as fees
4. Since `amount` is 500000, when the invest function calls IMintInList(longNFT).mintInList, it mints 500000 NFTs to the attacker, which is twice the amount they are paying for.

Recommendation

It is recommended to correct the mint amount in the invest function.

For example, change

```
IMintInList(longNFT).mintInList(msg.sender, amount);
```

to

```
IMintInList(longNFT).mintInList(msg.sender, mintable);
```

Status

This issue has been resolved by the team with commit 841cb0d.

4. Centralization risks

Severity: Medium	Category: Centralization
Target: - contracts/long/LongToken.sol - contracts/long/LongNFT.sol - contracts/long/LongFactory.sol	

Description

1. Centralization risk with initial token distribution
When the LongToken contract is deployed, all LONG tokens are sent to the deployer account. This account then has full control over the token distribution. If it is an EOA account, any compromise of its private key could drastically affect the project – for example, attackers could dump the price of LONG on the DEX if they gain access to the private key.

2. Centralization risk with the owner role in LongNFT
The LongNFT contract has a privileged owner role. The owner can:

- Set whitelist addresses by using setWhiteList(). The whitelist address can mint NFTs for free by using the mintInList() function. It's worth noting that the whitelist address is not required to be a contract, the owner can set an EOA account as a whitelist address.
- Set the BaseURI, TokenURI, and MysteryURI.
- Set the openMysteryBox state variable, which is used to determine the return value of tokenURI().
- Withdraw the ethers and tokens in the contract by using withdrawTokensSelf()

If the owner's private key is compromised, the attackers could free mint NFTs and change the URIs to compromise the project.

3. Centralization risk with contract upgradability
The LongNFT and LongFactory contracts are upgradable contracts. The proxy admin can upgrade the implementations of those contracts. Should the proxy admin's private key be compromised, the attacker can update the implementation to a malicious contract that drains all the token approved to the contract.

Recommendation

It is recommended to transfer all privileged accounts (e.g. initial token holder, owners, proxy admins) 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 acknowledged by the team.

5. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - contracts/long/LongNFT.sol - contracts/long/LongFactory.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes. Missing events do not promote transparency.

In the LongNFT contract, the following functions change critical state without emitting event:

- setVault()
- setTokenURI()
- setBaseURI()
- setMysteryURI()
- openBox()
- setWhiteList()

In the LongFactory contract, the following functions change critical state without emitting event:

- setStartEnd()
- setMintPrice()

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

6. Problems with ProxyStrap

Severity: Low	Category: Business Logic
Target: - contracts/strip/ProxyStrap.sol	

Description

There are two issues with the ProxyStrap contract.

1. The ProxyStrap is defined as an abstract contract, which means it cannot be deployed.
2. The ProxyStrap lacks the implementation of a constructor, which is responsible for constructing the TransparentUpgradeableProxy super contract. As a result, this contract can not be compiled.

Recommendation

It’s recommended to implement the contract correctly. It’s also recommended to document the intended use case for this contract.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

7. Should not leave the implementation contract uninitialized

Severity: Informational	Category: Access Control
Target: - contracts/long/LongNFT.sol - contracts/long/LongFactory.sol	

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized. It's recommended that you invoke the _disableInitializers() function in the constructor to automatically lock it, so that the implementation contract can not be initialized by malicious users.

Recommendation

Consider invoke the _disableInitializers() function in the constructor:

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}
```

Status

This issue has been acknowledged by the team.

8. Incorrect compiler version specified in hardhat.config.js

Severity: Informational	Category: Configuration
Target: - hardhat.config.js	

Description

hardhat.config.js:L18-L20

```
module.exports = {
  solidity: "0.7.3",
};
```

The hardhat.config.js file specifies the Solidity version as 0.7.3. However, this version does not match the versions specified in the pragma statements of the contracts.

When attempting to compile the project using Hardhat, the following error occurs.

```
Error HH606: The project cannot be compiled, see reasons below.

The Solidity version pragma statement in these files doesn't match any of the configured
compilers in your config. Change the pragma or configure additional compiler versions in
your hardhat config.

  * contracts/long/LongFactory.sol (0.6.12)
  * contracts/strip/ProxyStrap.sol (^0.6.12)
  * contracts/strip/AdminStrap.sol (^0.6.12)
  * contracts/long/LongToken.sol (^0.6.12)
  * contracts/long/LongNFT.sol (0.6.12)
```

Recommendation

Consider changing the solidity version in hardhat.config.js to 0.6.12.

Status

This issue has been resolved by the team with commit 841cb0d.

9. Unbounded loop may lead to out-of-gas error

Severity: Informational	Category: Gas
Target: - contracts/long/LongNFT.sol	

Description

There is no upper limit for the `amount` parameter in the mintInList() and burnBatch() functions. If the `amount` is a large number, these functions may revert because the gas used exceeds the block gas limit.

Recommendation

It’s recommended to warn the users about this issue. This way, if a user's call to mintInList (indirectly) or burnBatch fails due to a large `amount` parameter, they will be aware that it is because of an out-of-gas error. They can then choose to call the function multiple times with a smaller amount parameter.

Status

This issue has been acknowledged by the team.

10. Redundant code

Severity: Informational	Category: Redundancy
Target: - all	

Description

1. Redundant console library imported.

```
import "hardhat/console.sol";
```

The console.sol library is only used for testing purposes, and should be removed before deployment.

2. Redundant Ownable contract used.

The LongToken contract inherits the Ownable contract, but it doesn't have any owner-specific functions. Therefore, the use of the Ownable contract in LongToken is redundant and can be removed.

Recommendation

Consider removing the redundant codes.

Status

This issue has been acknowledged by the team. The team has removed the redundant console.sol libraries with commit 841cb0d.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file in commit 9e5a761:

File	SHA-1 hash
contracts/long/LongFactory.sol	09175af41f9fccc6535a1143f7444e6d8dfc7436
contracts/long/LongNFT.sol	6f502ec90ede488395a5a853d9a1934e6af40d19
contracts/long/LongToken.sol	38e8467b170819e71584e2a8ab8050a15fd8e93b
contracts/strip/AdminStrap.sol	77bbe053c838cbca90430e411f479829440f76ae
contracts/strip/ProxyStrap.sol	4b28a31dad0497f4e73b5991cbd1b663e22f6465