Mar 16th- Apr 11th , 2022

Lambda 256 Security Audit Report

version 1.1

Luniverse Bridge Smart Contract Audit

Audited by SOOHO

Contents

1. Introduction

As a beginning, we provide a disclaimer and describe how the SOOHO audit team approached the Luniverse bridge project methodologically during the security assessment period.

1.1 Disclaimer

- This document is based on a security assessment conducted by a blockchain security company SOOHO. This document describes the detected security vulnerabilities and also discusses the code quality and code license violations.

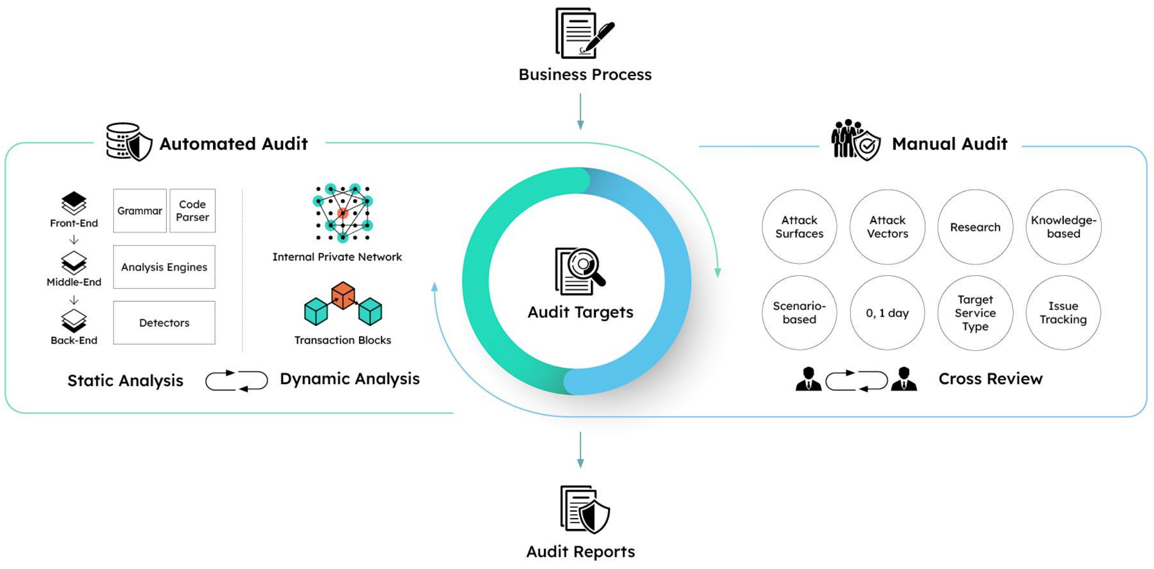
- This security assessment does not guarantee nor describe the usefulness of the code, the stability of the code, the suitability of the business model, the legal regulation of the business, the suitability of the contract, and the bug-free status. Audit document is used for discussion purposes only.

- SOOHO does not disclose any business information obtained during the review or save it through a separate media.

- SOOHO presents its best endeavors in smart contract security assessment.

1.2 Methodology

SOOHO conducts a more complete, continuous blockchain security assessment by applying two audit methodologies: Automated Audit and Manual Audit.



Automated audit ensures high quality of security assessment by finding various attacks quickly and precisely through cooperative analysis between static and dynamic analysis. Our static analyzer analyzes target codes through parsing grammar and verifies constraints through path finding and variable tracking. In dynamic analysis, emulation of target codes is executed in SOOHO's own test network along with fuzzing and concolic execution.

During the manual auditing process, our security experts verify the contract using various security and domain knowledge. Experts preferentially analyze codes with greater risk, check whether the codes are written under the client's intention and supervise access control management. Experts are complementing automated analysis by dealing with complex attack scenarios and recent security issues. We also provide more upgraded security audit reports through cross-review between security experts.

By conducting risk verification using various methodologies as above, SOOHO secures partners' contracts from 1-day to 0-day vulnerabilities.

Detected vulnerabilities are listed on the basis of the risk rating of vulnerability. The risk rating of vulnerability is set based on OWASP's Impact & Likelihood Risk Rating Methodology. Some issues were rated vulnerable aside from the corresponding model and the reasons are explained in the following results.

2. Assessment Results

In this section, we describe the overall structure of the Luniverse bridge project and provide the summary of findings. Details of detected vulnerabilities are given for improvement in the implementation of the Luniverse bridge project.

2.1 Analysis Target

Project	luniverse-bridge-contract-develop
# of Files	14
# of Target	14
# of Lines	1018
File Tree	luniverse-bridge-contract-develop/contracts

```
luniverse-bridge-contract-develop/contracts
├── Bridge.sol
├── ERC721Custody.sol
├── ExampleERC721.sol
├── IBridge.sol
├── ICustody.sol
├── IWERC721.sol
├── WERC721.sol
├── access
│   ├── Roles.sol
│   └── roles
│       ├── PauserRole.sol
│       └── SignerRole.sol
├── lifecycle
│   └── Pausable.sol
├── mapping
│   ├── IMapping.sol
│   └── Mapping.sol
└── utils
    └── Strings.sol
```

2.2 Summary

Severity	# of Findings
Critical	
High	
Medium	
Low	⬛ ⬛ ⬛
Note	⬛ ⬛

#	Description	Severity	Status
1	Boolean Operation code review	Low	Resolved
2	ChainCounterChecker code optimization	Note	Resolved
3	unlockAndReturnToken owner validation	Low	Resolved
4	Debug code removal	Note	Resolved
5	Note on usage of event Logs	Low	Resolved

Key Audit Points

The Luniverse Bridge project is a NFT bridge project based on ERC 721 protocol. Our audit team focused on reviewing whether the contract is constructed well following the standard protocol, gas optimization of the contract, or probability of unintended behaviors occurring, etc.

However, we did not take any internal hackings by administrators into account. Analyzes are about the functioning of the subject contract, given the safety of the system.

2.3 Findings

✓ (Resolved) Boolean Operation code review

File Name	Bridge.sol
File Location	contracts/Bridge.sol
File Hash	e81848a9d58bc0388dbafa0cd7bdeb6a

Line 171 ~ 174

```
require(
  _wrapperChainCounterChecker[wrapperChainCounter] = true,
  "Bridge: WrapperChainCounter is invalid"
);
```

Line 207 ~ 210

```
require(
  _originChainCounterChecker[originChainCounter] = true,
  "Bridge: OriginChainCounter is invalid"
);
```

Details

The require function is branched by assigning a statement, not with a boolean expression. Therefore, it always branches with the right operand value.

Mitigation

Change the assign statement to boolean expression in the first argument of the require function.

```
require(
  _originChainCounterChecker[originChainCounter] == true,
  "Bridge: OriginChainCounter is invalid"
);
```

Update

The condition argument is changed with a boolean operation correctly, in **#e81848a9**

✔ (Resolved) ChainCounterChecker code optimization

File Name	Bridge.sol
File Location	contracts/Bridge.sol
File Hash	e81848a9d58bc0388dbafa0cd7bdeb6a

Line 171 ~ 174

```
require(
  _wrapperChainCounterChecker[wrapperChainCounter] = true,
  "Bridge: WrapperChainCounter is invalid"
);
```

Line 207 ~ 210

```
require(
  _originChainCounterChecker[originChainCounter] = true,
  "Bridge: OriginChainCounter is invalid"
);
```

Details

ChainCounterCheck and ChainCounter both have the same context and the ChainCounterCheck array is not used elsewhere. Removal of the ChainCounterCheck array reduces the gas consumption by lessening SLOAD calls)

Suggestion

Replace the ChainCounterCheck related statements with counter variables.

```
require(
  wrapperChainCounter < _wrapperChainCounter,
  "Bridge: OriginChainCounter is invalid"
);
```

Update

The purpose of array check is to check whether or not same process working twice and the Check array is not monotonic true in #e81848a9. So, the comparison statement with check array and boolean value is more suitable for the require condition argument as in the previous code.

✔ (Resolved) unlockAndReturnToken owner validation

File Name	Bridge.sol
File Location	contracts/Bridge.sol
File Hash	e81848a9d58bc0388dbafa0cd7bdeb6a

Line 196 ~ 230

```
function unlockAndReturnToken(
  address originToken,
  address withdrawReceiver,
  bytes memory tokenData,
  bytes32 tokenType,
  uint256 originChainCounter
  )
    public
    onlySigner
    whenNotPaused
   {
    require(
      _originChainCounterChecker[originChainCounter] = true,
      "Bridge: OriginChainCounter is invalid"
    );
    require(
      originToWrappedToken(originToken) != address(0x0),
      "Bridge: TOKENADDRESS_NOT_MAPPED"
    );
  _unlockAndReturnToken(originToken, withdrawReceiver, tokenData, tokenType);
  address wrappedToken = originToWrappedToken(originToken);
  emit UnlockAndReturnToken(withdrawReceiver, originToken, wrappedToken, tokenData,
tokenType, _originChain, _wrapperChain);
}

function _unlockAndReturnToken(
  address originToken,
  address withdrawReceiver,
  bytes memory tokenData,
  bytes32 tokenType
  ) private {
    require(originToken != address(0), "Bridge: unlockAndReturnToken to zero contract
address");
    require(withdrawReceiver != address(0), "Bridge: unlockAndReturnToken to the zero
address");
  address custody = typeToCustody(tokenType);
  ICustody(custody).unlockToken(originToken, withdrawReceiver, tokenData);
}
```

(ERC721Custody.sol) Line 56 ~ 71

```
function unlockToken(
  address originToken,
  address withdrawReceiver,
  bytes calldata tokenData
)
  external
  onlyBridge
{
  uint256 tokenId = abi.decode(bytes(tokenData), (uint256));
  IERC721(originToken).safeTransferFrom(
      address(this),
      withdrawReceiver,
      tokenId
  );
  emit UnlockToken(bytes32(_tokenType), withdrawReceiver, originToken, tokenData);
}
```

Details

unlockToken behavior is only accessed by the signer privilege. Be careful when unlocking a token since there is no validation logic for checking an original owner of the token, which could result in transferring a token to a random user.

Acknowledgement

Luniverse dev team confirmed that owner verification is being made in the safeTransferFrom logic and reviewed token ownership.

✓	**(Resolved) Debug code removal**
File Name	ERC721Custody.sol
File Location	contracts/ERC721Custody.sol
File Hash	0790f1f58686c56c4b95ee99ba90ecfb

Line 8

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

Details

Remove Hardhat Debug code from the contract.

Details

In **#0790f1f5**, the debug codes are removed.

✔ (Resolved) Notes on usage of event logs

File Name	ERC721Custody.sol
File Location	contracts/ERC721Custody.sol
File Hash	0790f1f58686c56c4b95ee99ba90ecfb

Line 53

```
emit LockToken(bytes32(_tokenType), depositor, depositReceiver, originToken,
tokenData);
```

Line 106

```
emit BurnWrappedToken(bytes32(_tokenType), withdrawer, wrappedToken, tokenData);
```

Details

LockToken and BurnWarppedToken events are called by public functions. When tokenData is received, it is converted into token ID through abi.decode and used, but the emitted event stores tokenData instead of the converted token ID. Therefore, when performing tasks reading the event logs in off-chain, such as verifying owner in off-chain, it can mislead the decoded tokenData.

```
example 1)

tokenData = 0x0000000000000000000000000000000000000000000000000000000000000001
tokenId = 1

example 2)
tokenData =
0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000001
tokenId = 0
```

Suggestion

Log the same decoded value in event calls. In this contract, only token ID is extracted by tokenData so insert the token ID in the log events in order to avoid the decoding issue.

```
event BurnWrappedToken(
  bytes32 tokenType,
  address indexed withdrawer,
  address indexed wrappedToken,
  bytes tokenData,
  uint256 tokenId
);
```

```
function lockToken(
    address depositor,
    address depositReceiver,
    address originToken,
    bytes calldata tokenData
)
    external
    onlyBridge
{
    uint256 tokenId = abi.decode(bytes(tokenData),(uint256));
    IERC721(originToken).safeTransferFrom(
      depositor,
      address(this),
      tokenId
    );
    emit LockToken(bytes32(_tokenType), depositor, depositReceiver, originToken,
tokenData, tokenId);
}
```

Update

In **#0790f1f5**, the event now stores token Id which is the same as actual decoded value of tokenData.

2.4 Conclusion

The source code of the Luniverse Bridge contract developed by Lambda256 is easy to read and very well organized. We have to remark that contracts are well architected and all the additional features are implemented.

The detected vulnerabilities are as follows: **Total 5 issues (Low 3, Note 2) and SOOHO confirmed all the issues have been resolved by the team**. It is recommended to promote the stability of service through continuous code auditing and analyzing potential vulnerabilities.

About SOOHO

SOOHO with the motto of “Audit Everything, Automatically” researches and provides technology for a reliable blockchain ecosystem. SOOHO verifies vulnerabilities through the entire development life-cycle with Aegis, a vulnerability analyzer created by SOOHO, and open source analyzers.
SOOHO is composed of experts including Ph.D researchers in the field of automated security tools and white-hackers verifying contract codes and detecting vulnerabilities in depth. Professional experts in SOOHO secure partners’ contracts from known to zero-day vulnerabilities.

Contact us :)

Twitter:	SOOHO_AUDIT
E-mail:	audit@sooho.io
Website:	https://audit.sooho.io/