Mar 16th- Apr 11th , 2022

Lambda 256
Security Audit Report

version 1.1

Luniverse Bridge Smart Contract Audit

Audited by SOOHO

SOOHO

목차

1. 개요

'개요' 파트에서는 SOOHO 보안 감사 프로세스와 보안 감사에 사용된 방법론을 서술합니다. 분석 내용을 이해하시는 데에 참고하시길 바랍니다.

1.1 시작하기 전에

• 본 문서는 블록체인 보안 전문업체 SOOHO에서 진행한 취약점 검사를 바탕으로 작성한 문서로, 스마트 컨트랙트의 보안 취약점의 발견에 초점을 두고 있습니다. 추가적으로 코드 퀄리티 및 코드 라이센스 위반 사항 등에 대해서도 논의합니다.

• 본 문서는 코드의 유용성, 코드의 안정성, 비즈니스 모델의 적합성, 비즈니스의 법적인 규제, 계약의 적합성, 버그 없는 상태에 대해 보장하거나 서술하지 않습니다. 감사 문서는 논의 목적으로만 사용됩니다.

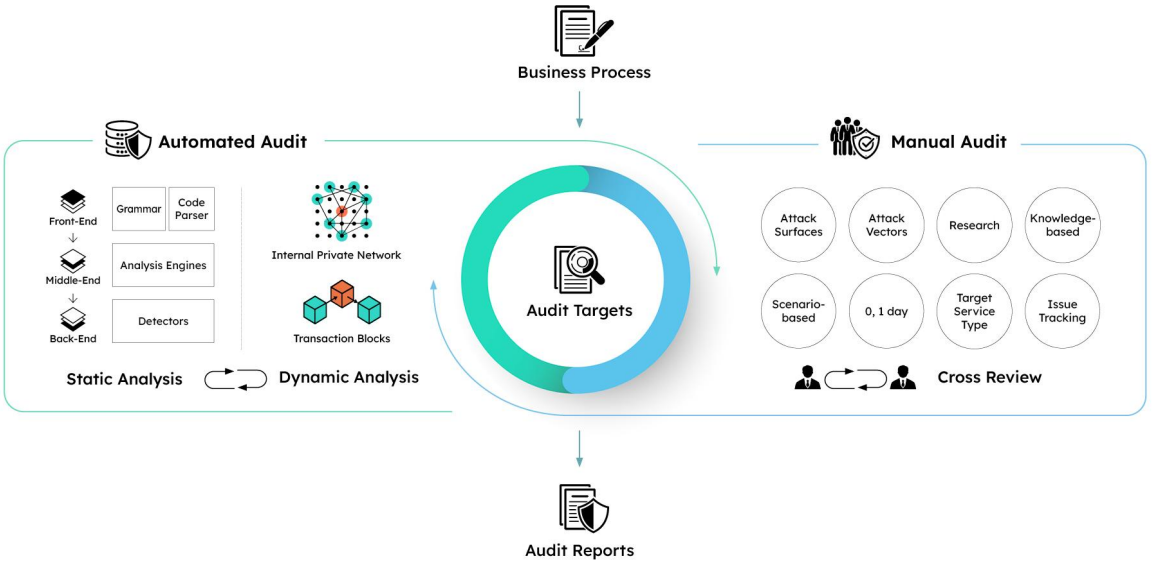
• SOOHO는 회사 정보가 대외비 이상의 성격을 가짐을 인지하고 사전 승인 없이 이를 공개하지 않습니다.

• SOOHO는 업무 수행 과정에서 취득한 일체의 회사 정보를 누설하거나 별도의 매체를 통해 소장하지 않습니다.

• SOOHO는 스마트 컨트랙트 분석에 최선을 다하였음을 밝히는 바입니다.

1.2 SOOHO 보안 감사 프로세스

SOOHO는 자동 분석(Automated Audit)과 수동 분석(Manual Audit)의 두 가지 감사 방법론을 적용하여 더욱 완벽한 블록체인 보안 감사를 진행합니다.



자동 분석은 정적 분석(Static analysis)과 동적 분석(Dynamic analysis) 사이 상호 협력적 분석을 통해 다양한 공격을 정확하고 빠르게 찾아내며 높은 감사 퀄리티를 보장합니다. SOOHO의 자체 분석기는 정적 분석을 통해 고객사 컨트랙트 코드의 문법을 분석(parsing)하여 의미 추론, 변수 추적, 경로 탐색을 진행함으로써 조건을 검증합니다. 정적 분석에서는 SOOHO 자체 테스트 네트워크에서 실제 동작을 통한 분석과 퍼징(Fuzzing), 콘콜릭 실행(Concolic Execution)을 통해 더욱 정교한 분석을 진행합니다.

수동 분석에서는 SOOHO의 보안 감사 전문가 그룹이 다양한 보안 및 도메인 지식을 활용하여 고객사의 프로젝트를 직접 검증합니다. 보다큰 리스크를 내포하는 코드를 중점적으로 분석하고, 파트너 사가 의도한 대로 코드가 작성되었는지 살피며 접근 권한의 관리가 올바르게 기능하고 있는지 검사합니다. 보안 감사 전문가들은 복잡한 공격 시나리오나 최근 보안 이슈를 처리하며 자동 분석을 상호 보완하여 감사의 완성도를 높입니다. 또한, 전문가 사이의 교차 검증을 통해 더욱 정교한 보안 감사 결과물을 제공합니다.

위와 같이 다양한 방법론을 이용하여 리스크 검증을 진행함으로써 파트너 사의 컨트랙트를 알려진 취약점(1-day)과 0-day 취약점의 위협으로부터 안전하게 만들어줍니다. 발견된 취약점은 심각도 척도를 기준으로 하여 등급이 매겨집니다. 심각성 척도는 OWASP의 Impact & Likelihood 기반 리스크 평가 모델을 기반으로 정해졌습니다.

2. 분석 내용

'분석 내용' 파트에서는 Luniverse Bridge 보안 감사 결과에 대한 전반적인 요약과 발견된 취약점의 구체적인 내용을 서술합니다. 발견된 모든 취약점에 대해 개선하는 것을 권장드리고 앞으로도 꾸준한 코드 감사를 통하여 서비스의 안정을 도모하시길 바랍니다.

2.1 분석 대상

Project	luniverse-bridge-contract-develop
# of Files	14
# of Target	14
# of Lines	1018
File Tree	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 분석 결과 요약

심각도 등급	취약점 수
Critical	
High	
Medium	
Low	■■■
Note	■■

#	설명	심각도 등급	상태
1	Boolean Operation 코드 리뷰	Low	Resolved
2	ChainCounterChecker 코드 최적화	Note	Resolved
3	unlockAndReturnToken 소유자 검증	Low	Resolved
4	Debug 코드 제거	Note	Resolved
5	Event 로그에 대한 주의	Low	Resolved

주요 감사 포인트

Luniverse Bridge 프로젝트는 ERC721 기반 NFT Bridge 프로젝트 입니다. SOOHO Audit은 해당 컨트랙트 코드가 표준에 잘 맞추어 작성되었는지, Gas의 낭비가 존재하지 않는지, 의도되지 않은 행동이 있는지를 중점적으로 감사하였습니다.

단, 관리자의 내부 해킹을 비롯해 컨트랙트 외부 서버 환경의 안정성은 고려하지 않았습니다. 본 보고서에서는 언급하지 않았지만 노드 자체에 대한 보안 검증과 외부 연동되는 서비스에 대해서도 검토하기를 제안합니다. 분석은 대상 프로젝트에 포함된 컨트랙트의 기능 안정성에 관한 것입니다.

2.3 분석 결과

✔ (Resolved) Boolean Operation 코드 리뷰

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

require 함수의 조건 부분을 Boolean 형태가 아닌 Assign 형태로 지정하여, 우측 피연산자의 값을 기준으로 분기 조건이 결정되게 됩니다.

Mitigation

require 함수의 조건 인자로 비교 연산을 넣어 원래 의도로 수정합니다.

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

Update

#e81848a9 에서 조건 인자에 대해 비교 연산으로 수정되었습니다.

✔ (Resolved) ChainCounterChecker 코드 최적화

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

ChainCounterChecker의 경우 ChainCounter와 항상 동일하게 설정되고 다른곳에서 사용되지 않기 때문에 ChainCounter만을 사용한다면 SLOAD 호출 수를 줄여 가스를 절약 할 수 있습니다.

Suggestion

Counter 값만을 비교하여 Checker의 접근을 하지 않고 원래 의도대로 동작 할 수 있도록 제안합니다.

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

Update

#e81848a9 에서 Checker 배열을 통해 ChainCounter Index에 대한 로직 (ex.clearWrapperChain CounterCheck)이 추가되었으며 단순히 단조 증가 값에 대한 비교가 아닌 반영에 대한 체크로 확인이되어 기존 방식처럼 배열 인덱스에 대한 비교 연산이 적절함을 확인하였습니다.

✔ (Resolved) unlockAndReturnToken 소유자 검증

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

unlockToken의 행위는 Signer 권한의 행위로 권한이 없는 사람은 접근이 불가능한 행위 입니다. 하지만 contract 상에서 unlock 대상 토큰에 대한 기존 소유자 검증 로직이 없어 잘못된 토큰 전송이 가능하기 때문에 이 행위에 대한 호출 시 주의가 필요합니다.

Acknowledgement

개발팀에서는 safeTransferFrom 내부 로직에서 소유자에 대한 검증이 이루어지고 있는 것과 토큰 소유 권한에 대해 확인하였습니다.

✔ (Resolved) Debug code 제거

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

Line 8

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

Details

Hardhat Debug 코드를 제거합니다.

Update

#0790f1f5 에서 디버그 코드가 제거 되었습니다.

✔ (Resolved) Event 로그에 대한 주의

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

LockToken과 BurnWrappedToken의 이벤트는 일반 사용자가 호출 할 수 있는 함수를 통해 발생되는 이벤트 입니다. 인자로 받는 tokenData의 경우 abi.decode를 통해 token ID로 변환하여 사용되지만 emit 되는 이벤트에는 변환된 token ID가 아닌 tokenData가 저장되기 때문에 오프체인에서 이벤트를 통하여 소유자 확인 등의 작업을 수행할때 tokenData 해석 방식에 따라 차이가 발생 될 수 있으니 주의가 필요합니다.

```
example 1)

tokenData = 0x0000000000000000000000000000000000000000000000000000000000000001
tokenId = 1

example 2)
tokenData =
0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
tokenId = 0
```

Suggestion

contract에서 작업이 이루어진 내역과 이벤트 로그가 일치하도록 합니다. 본 contract에서는 tokenData를 통해 tokenID만을 획득하기 때문에 tokenID를 같이 남기도록 합니다.

```
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

#0790f1f5 에서 tokenId가 저장되도록 변경되었습니다.

2.4 결론

람다256에서 개발한 Luniverse Bridge 컨트랙트는 이해하기 쉽게 명명되고 용도와 쓰임에 따라 잘 설계되어 있습니다. 대부분 모범 사례를 따르고 있습니다. 컨트랙트들은 매우 잘 설계되었고 그 구현 또한 훌륭하였음을 강조하고 싶습니다.

코드 검사 결과 발견된 이슈는 **총 5개 (Low 3개, Note 2개) 이며 5개의 이슈가 해결되었음을 확인하였습니다.** 브릿지 프로젝트 특성상 이벤트 로그 기반의 오프체인 작업에 대한 주의가 필요합니다. 꾸준한 코드 감사를 통해 컨트랙트의 안정을 도모하고 잠재적인 취약점에 대한 분석을 하는 것을 추천드립니다.

About SOOHO

SOOHO는 블록체인 생태계의 안전성과 신뢰도를 높이기 위한 기술을 연구하고 제공하고자 시작하였습니다. SOOHO는 자체적으로 개발한 취약점 분석기와 오픈 소스 분석기로 스마트 컨트랙트 취약점을 검증합니다. 더하여, SOOHO의 보안팀은 Defcon, Nuit du Hack, 화이트햇, SamsungCTF 등 국내외의 해킹 대회에서 수상하고, 보안분야 박사 학위와 같은 학문적 배경을 가지는 등 우수한 해킹 실력과 경험을 가진 보안 전문 인력들로 구성되어 있습니다. SOOHO의 전문 전문가들은 알려진 1-DAY 취약점부터 0-DAY 취약점으로부터 고객사의 스마트 컨트랙트를 보호하고자 합니다.

Contact us :)

Twitter: SOOHO_AUDIT

E-mail: audit@sooho.io

Website: https://audit.sooho.io/