PARALINK.NETWORK

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 12, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/12/2021	Nishit Majithia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Nishit Majithia	Halborn	nishit.majithia@halborn.com

EXECUTIVE SUMMARY

1.1 INTRODUCTION

Paralink Network is a scalable solution to the oracle problem on Polkadot. Polkadot is the optimal platform for Paralink protocol due to its fundamental design principles: cross chain interoperability and scalability are the two key components for making Paralink secure, scalable and economically viable. The security assessment was scoped to the smart contract ParaToken.sol. An audit of the security risk and implications regarding the changes introduced by the development team at Paralink Protocol prior to its production release shortly following the assessments deadline.

Overall, the smart contract code is well documented, follows a high-quality software development standard, contains many utilities and automation scripts to support continuous deployment / testing / integration, and does NOT contain any obvious exploitation vectors that Halborn was able to leverage within the timeframe of testing allotted.

Though the outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verication of essential properties were performed to achieve objectives and deliverables set in the scope. It is important to remark the use of the best practices for secure smart contract development. Halborn recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit.
While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage

of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:
- Research into architecture and purpose
- Smart Contract manual code read and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual Assessment of use and safety for the critical solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Scanning of solidity files for vulnerabilities, security hotspots, or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Truffle, Ganache, Infura)
- Smart Contract Fuzzing and dynamic state exploitation (Echidna) Symbolic Execution / EVM bytecode security assessment (limited time)

1.3 SCOPE

IN-SCOPE:
- ParaToken.sol
Specific commit of contract: commit a07e2a51c4d610e525d278937e709b161cbc4d72

OUT-OF-SCOPE:
- OracleUserExample.sol
- ParaFarming.sol
- ParaStaking.sol
- ParalinkOracle.sol
Other smart contracts in the repository, external libraries and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
USE OF BLOCK.TIMESTAMP	Informational	-
STATIC ANALYSIS	Informational	-

FINDINGS & TECH DETAILS

3.1 USE OF BLOCK.TIMESTAMP - INFORMATIONAL

Description:

Block timestamps have historically been used for a variety of applications, such as entropy for random numbers, locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

block.timestamp or it's alias now can be manipulated by miners if they have some incentive to do so

Code Location:

ParaToken.sol Line #121

```
117
118         address signatory = ecrecover(digest, v, r, s);
119         require(signatory != address(0), "PARA::delegateBySig: invalid signature");
120         require(nonce == nonces[signatory]++, "PARA::delegateBySig: invalid nonce");
121         require(now <= expiry, "PARA::delegateBySig: signature expired");
122         return _delegate(signatory, delegatee);
123     }
124
125     /**
126      * @notice Gets the current votes balance for `account`
```

Recommendation:

Avoid relying on block.timestamp

3.2 STATIC ANALYSIS - INFORMATIONAL

Description:

Slither and MythX has been run on all the scoped contracts(ParaToken.sol)

```
INFO:Detectors:
ParaToken._writeCheckpoint(address,uint32,uint256,uint256) (contracts/ParaToken.sol#216-234) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (contracts/ParaToken.sol#226)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
ParaToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (contracts/ParaToken.sol#82-123) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,PARA::delegateBySig: signature expired) (contracts/ParaToken.sol#121)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
ParaToken.getChainId() (contracts/ParaToken.sol#241-245) uses assembly
        - INLINE ASM (contracts/ParaToken.sol#243)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
INFO:Detectors:
Different versions of Solidity is used in :
        - Version used: ['0.6.12', '>=0.6.0<0.8.0']
        - 0.6.12 (contracts/ParaToken.sol#2)
        - >=0.6.0<0.8.0 (contracts/openzeppelin/contracts/GSN/Context.sol#3)
        - >=0.6.0<0.8.0 (contracts/openzeppelin/contracts/access/Ownable.sol#3)
        - >=0.6.0<0.8.0 (contracts/openzeppelin/contracts/math/SafeMath.sol#3)
        - >=0.6.0<0.8.0 (contracts/openzeppelin/contracts/token/ERC20/ERC20.sol#3)
        - >=0.6.0<0.8.0 (contracts/openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#3)
        - >=0.6.0<0.8.0 (contracts/openzeppelin/contracts/token/ERC20/IERC20.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version0.6.12 (contracts/ParaToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version>=0.6.0<0.8.0 (contracts/openzeppelin/contracts/GSN/Context.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/openzeppelin/contracts/access/Ownable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/openzeppelin/contracts/math/SafeMath.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/openzeppelin/contracts/token/ERC20/ERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/openzeppelin/contracts/token/ERC20/IERC20.sol#3) is too complex
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable ParaToken._delegates (contracts/ParaToken.sol#24) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (contracts/openzeppelin/contracts/GSN/Context.sol#21)" inContext (contracts/openzeppelin/contracts/GSN/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
mint(address,uint256) should be declared external:
        - ParaToken.mint(address,uint256) (contracts/ParaToken.sol#12-15)
owner() should be declared external:
        - Ownable.owner() (contracts/openzeppelin/contracts/access/Ownable.sol#35-37)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (contracts/openzeppelin/contracts/access/Ownable.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (contracts/openzeppelin/contracts/access/Ownable.sol#63-67)
symbol() should be declared external:
        - ERC20.symbol() (contracts/openzeppelin/contracts/token/ERC20/ERC20.sol#72-74)
decimals() should be declared external:
        - ERC20.decimals() (contracts/openzeppelin/contracts/token/ERC20/ERC20.sol#89-91)
totalSupply() should be declared external:
        - ERC20.totalSupply() (contracts/openzeppelin/contracts/token/ERC20/ERC20.sol#96-98)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (contracts/openzeppelin/contracts/token/ERC20/ERC20.sol#115-118)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (contracts/openzeppelin/contracts/token/ERC20/ERC20.sol#134-137)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (contracts/openzeppelin/contracts/token/ERC20/ERC20.sol#152-156)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (contracts/openzeppelin/contracts/token/ERC20/ERC20.sol#170-173)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (contracts/openzeppelin/contracts/token/ERC20/ERC20.sol#189-192)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (contracts/openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#21-23)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (contracts/openzeppelin/contracts/token/ERC20/ERC20Burnable.sol#36-41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:contracts/ParaToken.sol analyzed (7 contracts with 72 detectors), 28 result(s) found
```

MythX:

```
Report for ParaToken.sol
https://dashboard.mythx.io/#/console/analyses/51a13a02-76c9-401c-bd4a-0d40ab205a99
```

Line	SWC Title	Severity	Short Description
12	(SWC-000) Unknown	Medium	Function could be marked as external.
95	(SWC-128) DoS With Block Gas Limit	Low	Potentially unbounded data structure passed to builtin.
121	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.
151	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
151	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
170	(SWC-128) DoS With Block Gas Limit	Low	Loop over unbounded data structure.
224	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

THANK YOU FOR CHOOSING

// HALBORN