TECHFUND Audit Score
Smart contract Audit Report

OASYS Audit Report

Completed on 2022-09-31

Score **POSITIVE**

Risk level		
	Critical	0
	High	0
	Medium	1
	Low	6
	Note	0

Risk level detail

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

The tester arrives at the likelihood and impact estimates, they can now combine them to get a final severity rating for this risk. Note that if they have good business impact information, they should use that instead of the technical impact information.

https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

Vulnerability Review

Number of warnings

Compiler Version	1
Weak Random	2
Improper handling of errors	1
Variable Packing	0
Integer Overflow / Underflow	1
State access after external call	2
Callstack Depth Attack	0
Production Node modules security	0
Development Node modules security	0
Re-Entrancy	0
Double Withdrawal	0

TECHFUND Audit Score

Smart contract Audit Report

Title : Oasys-Validator
Commit Id : https://github.com/oasysgames/oasys-validator/tree/4f54fbf0e87021848112df99bc3baa579cf5de5e

Incorrect Handling of errors (oasys-validator) 1

oasys-validator/consensus/oasys/oasys.go:872

SealHash function does not properly handle exceptional conditions that may occur during normal operation. Proper error with correct return must be handled for each step.

```
// SealHash returns the hash of a block prior to it being sealed.
func SealHash(header *types.Header) (hash common.Hash) {
        hasher := sha3.NewLegacyKeccak256()
        encodeSigHeader(hasher, header)
        hasher.(crypto.KeccakState).Read(hash[:])
        return hash
}
```

Use of weak random number generator (oasys-validator) 1

oasys-validator/consensus/oasys/oasys.go:1020

Often PRNG is not designed for cryptography. Sometimes a mediocre source of randomness is sufficient or preferable for algorithms that use random numbers. Also weak generators generally take less processing power and/or do not use the precious, finite, entropy sources on a system.
But we suggest to use crypto/rand instead of math/rand.
For example `choice()` function becomes more deterministic because of use of Math/rand as compared to crypto/rand function.

```
        if (c.max) == 0 {
                i := rand.Intn(len(c.validators))
                return c.validators[i]
        }
```

Use of weak random number generator (oasys-validator) 1

oasys-validator/consensus/oasys/oasys.go:1068

Often PRNG is not designed for cryptography. Sometimes a mediocre source of randomness is sufficient or preferable for algorithms that use random numbers. Also weak generators generally take less processing power and/or do not use the precious, finite, entropy sources on a system. But we suggest to use crypto/rand instead of math/rand.
For example `newWeightedRandomChooser` function becomes more deterministic because of use of Math/rand as compared to crypto/rand function.

```
        validators, stakes = sortValidatorsAndValues(validators, stakes)
        chooser := &weightedRandomChooser{
                random:     rand.New(rand.NewSource(seed)),
                validators: make([]common.Address, len(validators)),
                totals:     make([]int, len(stakes)),
                max:        0,
        }
```

TECHFUND Audit Score

Smart contract Audit Report

Title : Oasys-Optimism
Commit Id : https://github.com/oasysgames/oasys-optimism/tree/d309f17fc0f6ddae83717735cadbfa8222e6fab3

TECHFUND Audit Score

Smart contract Audit Report

Compiler Version 1

packages/contracts/contracts/oasys/L1/messaging/*.sol
packages/contracts/contracts/oasys/L1/rollup/*.sol
packages/contracts/contracts/oasys/L1/token/*.sol

Solidity version used is inconsistent & not fixed. It is highly recommended to update the Solidity version to the latest. Solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statements and using a fixed version of Solidity upon understanding the version specifications.
https://swcregistry.io/docs/SWC-103

```
pragma solidity ^0.8.9;
```

Missing Input address checks / The arithmetic operator can underflow - overflow 1

packages/contracts/contracts/oasys/libraries/Allowlist.sol:86

addAddress function doesn't properly check for the input values of _address, which can later cause issues which comparing _address array in _contains. Cases for zero address and so on should be checked properly.

```
function addAddress(address _address) external onlyOwner {
    require(!_contains(allowlist, _address), "already added");
    allowlist.push(_address);

    emit AllowlistAdded(_address);
}
```

```
function _contains(address[] memory _addresses, address _address) inte
    uint256 length = _addresses.length;
    for (uint256 i = 0; i < length; i++) {
        if (_addresses[i] == _address) {
            return true;
        }
```

Missing Input address checks / The arithmetic operator can underflow - overflow 1

packages/contracts/contracts/oasys/libraries/Allowlist.sol:66

containsAddress function doesn't properly check for the input values of _address, which can later cause issues which comparing _address array in _contains. Cases for zero address and so on should be checked properly.

```
function containsAddress(address _address) external view returns (bool) {
    if (owner() == address(0)) {
        return true;
    }
    return _contains(allowlist, _address);
}
```

```
function _contains(address[] memory _addresses, address _address) inte
    uint256 length = _addresses.length;
    for (uint256 i = 0; i < length; i++) {
        if (_addresses[i] == _address) {
            return true;
        }
```

TECHFUND Audit Score

Smart contract Audit Report

Title : Oasys-Genesis-Contracts
Commit Id : https://github.com/oasysgames/oasys-genesis-contract/tree/f158f70451827a1bbc06c22ea14be3b13a68774a

State access after external call 1

contracts/nft-bridge/NFTBridgeMainchain.sol:119
contracts/nft-bridge/NFTBridgeMainchain.sol:125
contracts/nft-bridge/NFTBridgeMainchain.sol:134

As a part of good practice, it is generally suggested to prevent accessing state suddenly after external call. But the contract account state is accessed after an external call to a user defined address.

```
        try
            IERC721(mainInfo.mainchainERC721).safeTransferFrom(
                address(this),
                mainTo,
                mainInfo.tokenId
            )
        {
            mainInfo.mainTo = mainTo;

            emit WithdrawalFinalized(
                depositIndex,
                sidechainId,
                withdrawalIndex,
                mainInfo.mainchainERC721,
                sideFrom,
                mainTo
            );
        } catch {
            emit WithdrawalFailed(
                depositIndex,
                sidechainId,
                withdrawalIndex,
                mainInfo.mainchainERC721,
                sideFrom,
                mainTo
            );
        }
    }
```

State access after external call

1

contracts/nft-bridge/NFTBridgeMainchain.sol:51
contracts/nft-bridge/NFTBridgeMainchain.sol:56

As a part of good practice, it is generally suggested to prevent accessing state suddenly after external call. But the contract account state is accessed after an external call to a user defined address.

```
function deposit(
    address mainchainERC721,
    uint256 tokenId,
    uint256 sidechainId,
    address sideTo
) external {
    require(sideTo != address(0), "sideTo is zero address.");

    IERC721(mainchainERC721).transferFrom(
        msg.sender,
        address(this),
        tokenId
    );
    _depositInfos.push(
        DepositInfo(mainchainERC721, tokenId, msg.sender, address(0))
    );

    emit DepositInitiated(
        _depositInfos.length - 1,
        mainchainERC721,
        tokenId,
        sidechainId,
        msg.sender,
        sideTo
    );
}
```