TECHFUND Audit Score

Smart contract Audit Report

Oasys Audit

Completed on 2022-04-20

Score **POSITIVE**

Risk level		
	Critical	0
	High	0
	Medium	2
	Low	2
	Note	2

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	0
Array access optimisation for large arrays	1
Coding convention for events emitted during adding address	1
Checks for L2Token during ERC721 deposit	1
Checks for L1Token during ERC721 finalisation	1
Using platform independent path construction to read artifacts	1
Prevent accessing same variable multiple times while getting extra headers	1
Error in setting up Dynamic Arrays	0
Gas wastage because of un-necessary bound checks	0
Re-Entrancy (out of order events)	0
Double Withdrawal	0

Array access optimisation for large arrays 1

https://github.com/oasysgames/oasys-genesis-contract/blob/b6dc411402eff2006fe4c489cbee86898edd0b49/contracts/lib/AddressList.sol

Gas Optimisation: In the list function, the loop retrieves elements from the _addresses array by index. Instead of using a loop, you can use the array.slice function to extract the desired portion of the array.
This can reduce gas costs when the array size is large and lead to a cleaner code.

```
function list(
    uint256 cursor,
    uint256 howMany
) external view returns (address[] memory addresses, uint256 newCursor) {
    uint256 _length = _addresses.length;
    if (cursor + howMany >= _length) {
        howMany = _length - cursor;
    }
    newCursor = cursor + howMany;

    addresses = new address[](howMany);
    for (uint256 i = 0; i < howMany; i++) {
        addresses[i] = _addresses[cursor + i];
    }

    return (addresses, newCursor);
}
```

TECHFUND Audit Score

Smart contract Audit Report

Coding convention for events emitted during adding address

1

https://github.com/oasysgames/oasys-genesis-contract/blob/b6dc411402eff2006fe4c489cbee86898edd0b49/contracts/lib/AddressList.sol

Events should follow a naming convention to distinguish them from functions and variables. Event names are typically in the past tense and indicate what has occurred. For example, instead of Added, you can use AddressAdded. This is part of coding convention suggestion and is not from a point of view of security.

```
function _add(address _address) internal returns (bool result) {
    if (_address == address(0)) revert NullAddress();

    uint256 _id = _ids[_address];
    if (_id != 0) return false;

    _addresses.push(_address);
    _ids[_address] = _addresses.length;

    emit Added(_address);

    return true;
}
```

Checks for L2Token during ERC721 deposit

1

https://github.com/oasysgames/oasys-optimism/compare/develop...update-contracts

The function _initiateERC721Deposit is a private or internal function that is used internally within the contract. It takes several input parameters, including _l1Token, _l2Token, _from (the address from which the token is being transferred), _to (the address to which the token is being transferred), _tokenId (the ID of the ERC721 token), _l2Gas (the amount of gas to be sent along with the transaction), and _data (additional data to be included with the transaction).

The issue in the code is the lack of a check for the _l2Token address. The require statement must be used to enforce a condition, and if the condition is not met, it will revert the transaction with an error message.

```
require(_l2Token != address(0), "L1ERC721Bridge: _l2Token cannot be address(0)");
```

Checks for L2Token during ERC721 deposit

1

The purpose of this check is to prevent any accidental or intentional misuse of the function by passing an invalid _l2Token address. If an invalid address is provided, the require statement will throw an exception and revert the transaction, providing the error message "L1ERC721Bridge: _l2Token cannot be address(0)".

```
function _initiateERC721Deposit(
    address _l1Token,
    address _l2Token,
    address _from,
    address _to,
    uint256 _tokenId,
    uint32 _l2Gas,
    bytes calldata _data
) internal override {
    require(!deposits[_l1Token][_l2Token][_tokenId], "Already deposited");
```

Checks for L1Token during ERC721 finalisation

1

https://github.com/oasysgames/oasys-optimism/compare/develop...update-contracts

The finalizeERC721Withdrawal function appears to handle the finalization of an ERC721 token withdrawal process. It accepts several input parameters including _l1Token, _l2Token, _from (the address from which the token is being transferred), _to (the address to which the token is being transferred), _tokenId (the ID of the ERC721 token), and _data (additional data associated with the withdrawal).

```
require(_l1Token != address(this), "L1ERC721Bridge: _l1Token cannot be self");
```

This require statement enforces the condition that the _l1Token address provided should not be equal to the contract's own address. The address(this) expression represents the address of the contract itself.

Including this check helps prevent potential issues or vulnerabilities that could arise if the contract's own address is mistakenly used as the _l1Token address during the withdrawal process. If the check fails and the _l1Token address matches the contract's address, the require statement will revert the transaction and provide the error message "L1ERC721Bridge: _l1Token cannot be self".

Using platform independent path construction to read artifacts

1

By using filepath.Join instead of manually concatenating the path components, you ensure that the resulting path is correctly constructed and compatible with the underlying platform. The suggested alternative replaces the hardcoded string path with a more flexible and platform-independent approach using the **filepath.Join** function.

```
// Oasys genesis contracts
environment = &genesisContract{
        address: common.HexToAddress(environmentAddress),
        artifact: &artifact{
                path: "oasys-genesis-contract-cfb3cd0/artifacts/contracts/Environment.sol/Environment.json",
        },
}
stakeManager = &genesisContract{
        address: common.HexToAddress(stakeManagerAddress),
        artifact: &artifact{
                path: "oasys-genesis-contract-cfb3cd0/artifacts/contracts/StakeManager.sol/StakeManager.json"
        },
}
systemMethods = map[*genesisContract]map[string]int{
        // Methods with the `onlyCoinbase` modifier are system methods.
        // See: https://github.com/oasysgames/oasys-genesis-contract/search?q=onlyCoinbase
        environment:  {"initialize": 0, "updateValue": 0},
        stakeManager: {"initialize": 0, "slash": 0},
}

candidateManager = &builtinContract{
        address: common.HexToAddress(candidateManagerAddress),
        artifact: &artifact{
                path: "oasys-genesis-contract-6037082/artifacts/contracts/CandidateValidatorManager.sol/Candi
        },
}
```

The original code assigns the path directly as a string, which is "oasys-genesis-contract-cfb3cd0/artifacts/contracts/Environment.sol/Environment.json". It represents a hardcoded path with directory and file names concatenated using forward slashes.
The suggested alternative uses the filepath.Join function to construct the path. Each path component is provided as a separate argument to the function.
The filepath.Join function takes care of properly joining the components using the appropriate path separator based on the underlying operating system. For example, on Unix-like systems, it would use forward slashes, while on Windows, it would use backslashes.

Prevent accessing same variable multiple times while getting extra headers

1

In the provided code, the common.AddressLength is accessed multiple times to determine the length of the byte representation of an address. Here's the relevant section of the code:

```
extra := make([]byte, len(cpy)*common.AddressLength)
for i, v := range cpy {
        copy(extra[i*common.AddressLength:], v.Bytes())
}
```

When this line is executed, the common.AddressLength value is retrieved and multiplied by len(cpy) to determine the required length of the extra byte slice. We suggest to use the common address length as follow :

```
const addressLength = common.AddressLength
```

Use constant for common.AddressLength:
If common.AddressLength is a constant value, it's better to define it as a constant variable rather than accessing it repeatedly. This will help in better readability and slightly better performance.