BASE
Findings & Analysis Report

2023-08-10

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit outlined in this document, C4 conducted an analysis of the BASE smart contract system written in Solidity. The audit took place between May 26 - June 9 2023.

Wardens

32 Wardens contributed reports to the BASE:

1. 0x73696d616f
2. 0xTheC0der
3. 0xdeadbeef0x
4. 0xhacksmithh
5. Bauchibred
6. GalloDaSballo
7. KKat7531
8. Madalad
9. MohammedRizwan
10. Rolezn
11. SAAJ
12. SanketKogekar

13. Sathish9098
14. VictoryGod
15. brgltd
16. btk
17. codeslide
18. descharre
19. fatherOfBlocks
20. hunter_w3b
21. jauvany
22. judeabara
23. kaveyjoe
24. koxuan
25. ladboy233
26. nadin
27. niser93
28. rbserver
29. shealtielanz
30. souilos
31. trysam2003
32. yongskiws

This audit was judged by 0xleastwood.

Final report assembled by thebrittfactor.

Summary

The C4 analysis yielded 0 unique HIGH or MEDIUM vulnerabilities.

Additionally, C4 analysis included 29 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 4 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 BASE repository, and is composed of 14 smart contracts written in the Solidity programming language and includes 570 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

Low Risk and Non-Critical Issues

For this audit, 29 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **rbserver** received the top score from the judge.

The following wardens also submitted reports: brgltd, shealtielanz, KKat7531, yongskiws, btk, descharre, niser93, 0xTheC0der, GalloDaSballo, codeslide, kaveyjoe, VictoryGod, SAAJ, Madalad, jauvany, nadin, MohammedRizwan, hunter_w3b, 0x73696d616f, Rolezn, SanketKogekar, trysam2003, Bauchibred, Sathish9098, souilos, 0xhacksmithh, 0xdeadbeef, *and* ladboy.

QA Report Summary

	Issue
[01]	`OptimismPortal.receive` FUNCTION SHOULD ONLY BE CALLABLE BY EOAS
[02]	CALLING `OptimismPortal.depositTransaction` FUNCTION DOES NOT REVERT WHEN `_isCreation` IS FALSE AND `_to == address(0)` IS TRUE
[03]	`L2CrossDomainMessenger._sendMessage`, `L2StandardBridge._initiateWithdrawal`, AND `L2ToL1MessagePasser.initiateWithdrawal` FUNCTIONS ALLOW RESPECTIVE `_to` OR `_target` TO BE `address(0)`
[04]	`L2ToL1MessagePasser.burn` FUNCTION CAN STOP WORKING WHEN `SELFDESTRUCT` BECOMES DEACTIVATED
[05]	`L2ToL1MessagePasser.burn` IS AN UNPERMISSIONED FUNCTION
[06]	CALLING `OptimismPortal.finalizeWithdrawalTransaction` FUNCTION DOES NOT REVERT WHEN LOW LEVEL CALL TO `_tx.target` FAILS FOR SOME REASONS OTHER THAN HAVING INSUFFICIENT GAS IN CURRENT CONTEXT
[07]	`OptimismMintableERC20Factory.createOptimismMintableERC20` FUNCTION CAN BE CALLED FOR MULTIPLE TIMES FOR SAME `_remoteToken` - `_name` - `_symbol` COMBINATION
[08]	`SystemConfig._setResourceConfig` FUNCTION SHOULD REQUIRE `_config.minimumBaseFee < _config.maximumBaseFee`
[09]	CALLING `L2OutputOracle.proposeL2Output` FUNCTION AT PRESENT REVERTS
[10]	MISSING `address(0)` CHECKS FOR `address` CONSTRUCTOR INPUTS
[11]	VULNERABILITIES IN VERSION 4.7.3 OF `@openzeppelin/contracts` AND `@openzeppelin/contracts-upgradeable`
[12]	MORE UPDATED VERSION OF SOLIDITY CAN BE USED
[13]	`require(_gasLimit >= minimumGasLimit(), "SystemConfig: gas limit too low")` CAN BE REFACTORED INTO A MODIFIER TO BE USED IN CORRESPONDING FUNCTIONS
[14]	CONSTANTS CAN BE USED INSTEAD OF MAGIC NUMBERS
[15]	HARDCODED STRING THAT IS REPEATEDLY USED CAN BE REPLACED WITH A CONSTANT

	Issue
[16]	`revert` CAN BE CALLED INSTEAD OF `assert` TO PROVIDE CLEARER REASON FOR REVERSION
[17]	PUBLIC FUNCTIONS THAT ARE NOT CALLED BY OTHER FUNCTIONS IN SAME CONTRACT CAN BE EXTERNAL
[18]	UNDERSCORE CAN BE ADDED FOR `21000` IN `OptimismPortal.minimumGasLimit` FUNCTION
[19]	WORD TYPING TYPO
[20]	BOOLEAN VARIABLE COMPARISONS ARE NOT HANDLED CONSISTENTLY
[21]	`revert` WITH CUSTOM ERRORS CAN BE EXECUTED INSTEAD OF EXECUTING `require` OR `revert` WITH REASON STRINGS
[22]	ORDERS OF FUNCTIONS DO NOT FOLLOW OFFICIAL STYLE GUIDE
[23]	INCOMPLETE NATSPEC COMMENTS

[01] `OptimismPortal.receive` FUNCTION SHOULD ONLY BE CALLABLE BY EOAS

The following `OptimismPortal.receive` function should only be callable by EOAs. If it is called by a contract, the deposited funds would be lost. To prevent this from happening, the `OptimismPortal.receive` function can use a modifier that is similar to the `L1StandardBridge.onlyEOA` modifier below, which is used in the `L1StandardBridge.receive` function.

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L200-L209

```
    /**
     * @notice Accepts value so that users can send ETH directly
     *         funds be deposited to their address on L2. This i
     *         function for EOAs. Contracts should call the depo
     *         otherwise any deposited funds will be lost due to
     */
```

```
        // solhint-disable-next-line ordering
        receive() external payable {
            depositTransaction(msg.sender, msg.value, RECEIVE_DEFAUI
        }
```

https://github.com/ethereum-optimism/optimism/blob/95c16b23ae0528e0b4efce04ac5d9d0f2a289adf/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L64-L68

```
        modifier onlyEOA() {
            // Used to stop deposits from contracts (avoid accidenta
            require(!Address.isContract(msg.sender), "Account not EC
            _;
        }
```

https://github.com/ethereum-optimism/optimism/blob/95c16b23ae0528e0b4efce04ac5d9d0f2a289adf/packages/contracts/contracts/L1/messaging/L1StandardBridge.sol#L76-L78

```
        receive() external payable onlyEOA {
            _initiateETHDeposit(msg.sender, msg.sender, 200_000, byt
        }
```

[02] CALLING `OptimismPortal.depositTransaction` FUNCTION DOES NOT REVERT WHEN `_isCreation` IS FALSE AND `_to == address(0)` IS TRUE

When `_isCreation` is false, `_to` can be mistakenly input as `address(0)` when calling the `OptimismPortal.depositTransaction` function. When this occurs, the deposited funds would be lost. To avoid this, please consider updating the `OptimismPortal.depositTransaction` function to revert when `_isCreation` is false and when `_to == address(0)` is true.

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L434-L483

```
function depositTransaction(
    address _to,
    uint256 _value,
    uint64 _gasLimit,
    bool _isCreation,
    bytes memory _data
) public payable metered(_gasLimit) {
    // Just to be safe, make sure that people specify addres
    // contract creations.
    if (_isCreation) {
        require(
            _to == address(0),
            "OptimismPortal: must send to address(0) when cr
        );
    }

    ...
}
```

[03] `L2CrossDomainMessenger._sendMessage`, `L2StandardBridge._initiateWithdrawal`, AND `L2ToL1MessagePasser.initiateWithdrawal` FUNCTIONS ALLOW RESPECTIVE `_to` OR `_target` TO BE `address(0)`

Calling `L2CrossDomainMessenger._sendMessage`, `L2StandardBridge._initiateWithdrawal`, and `L2ToL1MessagePasser.initiateWithdrawal` functions with the respective `_to` or `_target` being `address(0)` can cause funds to be sent to `address(0)` on L1. To prevent users from losing their funds being withdrawn on L1, please consider updating these functions to revert when the respective `_to` or `_target` is `address(0)`.

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L2/L2CrossDomainMessenger.sol#L51-L60

```
function _sendMessage(
    address _to,
```

```
        uint64 _gasLimit,
        uint256 _value,
        bytes memory _data
    ) internal override {
        L2ToL1MessagePasser(payable(Predeploys.L2_TO_L1_MESSAGE_
            value: _value
        }(_to, _gasLimit, _data);
    }
```

https://github.com/ethereum-optimism/optimism/blob/bf51c4935261634120f31827c3910aa631f6bf9c/packages/contracts-bedrock/contracts/L2/L2StandardBridge.sol#L179-L193

```
    function _initiateWithdrawal(
        address _l2Token,
        address _from,
        address _to,
        uint256 _amount,
        uint32 _minGasLimit,
        bytes memory _extraData
    ) internal {
        if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
            _initiateBridgeETH(_from, _to, _amount, _minGasLimit
        } else {
            address l1Token = OptimismMintableERC20(_l2Token).l1
            _initiateBridgeERC20(_l2Token, l1Token, _from, _to,
        }
    }
```

https://github.com/ethereum-optimism/optimism/blob/5e33fd7695acfe86adbf6e6d397390d09b3b6d8c/packages/contracts-bedrock/contracts/L2/L2ToL1MessagePasser.sol#L98-L129

```
    function initiateWithdrawal(
        address _target,
        uint256 _gasLimit,
        bytes memory _data
    ) public payable {
        bytes32 withdrawalHash = Hashing.hashWithdrawal(
            Types.WithdrawalTransaction({
                nonce: messageNonce(),
```

```
                sender: msg.sender,
                target: _target,
                value: msg.value,
                gasLimit: _gasLimit,
                data: _data
            })
        );

        sentMessages[withdrawalHash] = true;

        emit MessagePassed(
            messageNonce(),
            msg.sender,
            _target,
            msg.value,
            _gasLimit,
            _data,
            withdrawalHash
        );

        unchecked {
            ++msgNonce;
        }
    }
```

[04] `L2ToL1MessagePasser.burn` FUNCTION CAN STOP WORKING WHEN `SELFDESTRUCT` BECOMES DEACTIVATED

The `L2ToL1MessagePasser.burn` function executes `Burn.eth(balance)`, which eventually executes `selfdestruct(payable(address(this)))`, for burning all ETH held by the `L2ToL1MessagePasser` contract. However, when EIP-4758 becomes effective to deactivate `SELFDESTRUCT` in the future, this `L2ToL1MessagePasser.burn` function will no longer work, and such functionality for burning all ETH held by the `L2ToL1MessagePasser` contract will be unavailable; in which the inflation of the amount of ETH on L2 when ETH is withdrawn cannot be prevented. To be more future-proofed, please consider updating the `L2ToL1MessagePasser.burn` function to send the ETH amount of `address(this).balance` to a designated address, such as `address(0)`, for burning all ETH held by the `L2ToL1MessagePasser` contract.

https://github.com/ethereum-optimism/optimism/blob/5e33fd7695acfe86adbf6e6d397390d09b3b6d8c/packages/contracts-bedrock/contracts/L2/L2ToL1MessagePasser.sol#L85-L89

```
    function burn() external {
        uint256 balance = address(this).balance;
        Burn.eth(balance);
        emit WithdrawerBalanceBurnt(balance);
    }
```

https://github.com/ethereum-optimism/optimism/blob/da7ee2228c48f9280b336bb9da316057e082d364/packages/contracts-bedrock/contracts/libraries/Burn.sol#L14-L16

```
    function eth(uint256 _amount) internal {
        new Burner{ value: _amount }();
    }
```

https://github.com/ethereum-optimism/optimism/blob/da7ee2228c48f9280b336bb9da316057e082d364/packages/contracts-bedrock/contracts/libraries/Burn.sol#L38-L42

```
contract Burner {
    constructor() payable {
        selfdestruct(payable(address(this)));
    }
}
```

[05] `L2ToL1MessagePasser.burn` IS AN UNPERMISSIONED FUNCTION

Anyone can call the `L2ToL1MessagePasser.burn` function to burn all ETH held by the `L2ToL1MessagePasser` contract. Yet, when ETH is not withdrawn, this function can still be called, which can deflate the amount of ETH on L2. To avoid this from occurring, please consider updating the `L2ToL1MessagePasser.burn` function to be only callable by the trusted admin.

https://github.com/ethereum-optimism/optimism/blob/5e33fd7695acfe86adbf6e6d397390d09b3b6d8c/packages/contracts-bedrock/contracts/L2/L2ToL1MessagePasser.sol#L85-L89

```
function burn() external {
    uint256 balance = address(this).balance;
    Burn.eth(balance);
    emit WithdrawerBalanceBurnt(balance);
}
```

After the `L2ToL1MessagePasser.burn` function is called to remove all ETH held by the `L2ToL1MessagePasser` contract from the state, this function can still be called. Because `address(this).balance` has become 0 already, calling this function for many times; in this case, it would emit meaningless `WithdrawerBalanceBurnt` events, which spam the monitor system that consumes such event. To prevent such event log poisoning, please consider updating the `L2ToL1MessagePasser.burn` function to revert when `address(this).balance` is 0.

```
function burn() external {
    uint256 balance = address(this).balance;
    Burn.eth(balance);
    emit WithdrawerBalanceBurnt(balance);
}
```

[06] CALLING `OptimismPortal.finalizeWithdrawalTransaction` FUNCTION DOES NOT REVERT WHEN LOW LEVEL CALL TO `_tx.target` FAILS FOR SOME REASONS OTHER THAN HAVING INSUFFICIENT GAS IN CURRENT CONTEXT

Calling the `OptimismPortal.finalizeWithdrawalTransaction` function can result in a false `success` when the low level call to `_tx.target` fails for some reasons other than having insufficient gas in the current context. In this case, since `tx.origin == Constants.ESTIMATION_ADDRESS` is false, calling the `OptimismPortal.finalizeWithdrawalTransaction` function would not revert, the withdrawal transaction is considered as finalized in which

`finalizedWithdrawals[withdrawalHash]` would be set to true even though the low level call to `_tx.target` fails, and the associated funds to be withdrawn remain in the `OptimismPortal` contract. Because `finalizedWithdrawals[withdrawalHash]` is already true, calling the `OptimismPortal.finalizeWithdrawalTransaction` function for the same `withdrawalHash` again would revert due to `require(finalizedWithdrawals[withdrawalHash] == false, "OptimismPortal: withdrawal has already been finalized")`. To allow the reattempt for withdrawing the funds for the same `withdrawalHash` in this situation, please consider updating the `OptimismPortal.finalizeWithdrawalTransaction` function to revert when `success == false` is true without requiring `tx.origin == Constants.ESTIMATION_ADDRESS` to also be true.

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L325-L420

```
    function finalizeWithdrawalTransaction(Types.WithdrawalTrans
        external
        whenNotPaused
    {
        ...

        // Check that this withdrawal has not already been final
        require(
            finalizedWithdrawals[withdrawalHash] == false,
            "OptimismPortal: withdrawal has already been finaliz
        );

        // Mark the withdrawal as finalized so it can't be repla
        finalizedWithdrawals[withdrawalHash] = true;

        // Set the l2Sender so contracts know who triggered this
        l2Sender = _tx.sender;

        // Trigger the call to the target contract. We use a cus
        // SafeCall.callWithMinGas to ensure two key properties
        //   1. Target contracts cannot force this call to run c
        //      amount of data (and this is OK because we don't
        //   2. The amount of gas provided to the execution cont
```

```
            //      gas limit specified by the user. If there is not
            //      to accomplish this, `callWithMinGas` will revert
            bool success = SafeCall.callWithMinGas(_tx.target, _tx.g

            // Reset the l2Sender back to the default value.
            l2Sender = Constants.DEFAULT_L2_SENDER;

            // All withdrawals are immediately finalized. Replayabil
            // be achieved through contracts built on top of this cc
            emit WithdrawalFinalized(withdrawalHash, success);

            // Reverting here is useful for determining the exact ga
            // sub call to the target contract if the minimum gas li
            // be sufficient to execute the sub call.
            if (success == false && tx.origin == Constants.ESTIMATIC
                revert("OptimismPortal: withdrawal failed");
            }
        }
```

[07] `OptimismMintableERC20Factory.createOptimismMintableERC20` FUNCTION CAN BE CALLED FOR MULTIPLE TIMES FOR SAME `_remoteToken` - `_name` - `_symbol` COMBINATION

The `OptimismMintableERC20Factory.createOptimismMintableERC20` function can be called multiple times for the same `_remoteToken` - `_name` - `_symbol` combination. This means that multiple instances of the `OptimismMintableERC20` contract can be created for the same `_remoteToken` - `_name` - `_symbol` combination, which can then be used to trick and phishing attack users. To be more secured, please consider updating the `OptimismMintableERC20Factory.createOptimismMintableERC20` function to revert when an instance of the `OptimismMintableERC20` contract has already been created for the same `_remoteToken` - `_name` - `_symbol` combination.

https://github.com/ethereum-optimism/optimism/blob/365367e6716d65ba45b583d72d9b030fbb5c0e47/packages/contracts-bedrock/contracts/universal/OptimismMintableERC20Factory.sol#L87-L109

```
function createOptimismMintableERC20(
    address _remoteToken,
    string memory _name,
    string memory _symbol
) public returns (address) {
    require(
        _remoteToken != address(0),
        "OptimismMintableERC20Factory: must provide remote t
    );

    address localToken = address(
        new OptimismMintableERC20(BRIDGE, _remoteToken, _nam
    );

    // Emit the old event too for legacy support.
    emit StandardL2TokenCreated(_remoteToken, localToken);

    // Emit the updated event. The arguments here differ frc
    // are consistent with the ordering used in StandardBric
    emit OptimismMintableERC20Created(localToken, _remoteTok

    return localToken;
}
```

[08] `SystemConfig._setResourceConfig` FUNCTION SHOULD REQUIRE `_config.minimumBaseFee < _config.maximumBaseFee`

The reason string of the `require` statement below in the following `SystemConfig._setResourceConfig` function specifies `SystemConfig: min base fee must be less than max base`. However, the condition of such `require` statement is `_config.minimumBaseFee <= _config.maximumBaseFee`, which would be true if `_config.minimumBaseFee` and `_config.maximumBaseFee` are equal. This means that the `SystemConfig._setResourceConfig` function allows the minimum base fee to equal the maximum base fee, even though the specification requires the minimum base fee to be less than the maximum base fee. To match the intended specification, please consider updating `_config.minimumBaseFee <= _config.maximumBaseFee` to `_config.minimumBaseFee <`

`_config.maximumBaseFee` in the `require` statement in the `SystemConfig._setResourceConfig` function.

https://github.com/ethereum-optimism/optimism/blob/4a01d2750ea10ad1109ff643faea2d8cfb28013f/packages/contracts-bedrock/contracts/L1/SystemConfig.sol#L266-L296

```
        function _setResourceConfig(ResourceMetering.ResourceConfig
            // Min base fee must be less than or equal to max base f
            require(
                _config.minimumBaseFee <= _config.maximumBaseFee,
                "SystemConfig: min base fee must be less than max ba
            );
            ...
        }
```

[09] CALLING `L2OutputOracle.proposeL2Output` FUNCTION AT PRESENT REVERTS

The `L2OutputOracle.proposeL2Output` function executes `require(computeL2Timestamp(_l2BlockNumber) < block.timestamp, "L2OutputOracle: cannot propose L2 output in the future")`, which does not allow proposing L2 output in the future. Yet, when `computeL2Timestamp(_l2BlockNumber) == block.timestamp` is true, the time is the present, not the future. But executing such `require` statement reverts, even though the `require` statement's reason string is `L2OutputOracle: cannot propose L2 output in the future`. To allow proposing L2 output at present, please consider updating such `require` statement to `require(computeL2Timestamp(_l2BlockNumber) <= block.timestamp, "L2OutputOracle: cannot propose L2 output in the future")`.

https://github.com/ethereum-optimism/optimism/blob/d322c6d651022ceb0798168726fe47416c6ddf00/packages/contracts-bedrock/contracts/L1/L2OutputOracle.sol#L179-L229

```
        function proposeL2Output(
            bytes32 _outputRoot,
```

```
        uint256 _l2BlockNumber,
        bytes32 _l1BlockHash,
        uint256 _l1BlockNumber
    ) external payable {
        ...

        require(
            computeL2Timestamp(_l2BlockNumber) < block.timestamp
            "L2OutputOracle: cannot propose L2 output in the fut
        );

        ...
    }
```

[10] MISSING `address(0)` CHECKS FOR `address` CONSTRUCTOR INPUTS

To prevent unintended behaviors, critical constructor inputs that are `address` should be checked against `address(0)`. `address(0)` checks are missing for the `address` inputs of the following constructors. Please consider checking them.

https://github.com/ethereum-optimism/optimism/blob/d322c6d651022ceb0798168726fe47416c6ddf00/packages/contracts-bedrock/contracts/L1/L2OutputOracle.sol#L90-L112

```
    constructor(
        ...
        address _proposer,
        address _challenger,
        ...
    ) Semver(1, 3, 0) {
        ...
        PROPOSER = _proposer;
        CHALLENGER = _challenger;
        ...
    }
```

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L150-L160

```
    constructor(
        ...
        address _guardian,
        ...
    ) Semver(1, 6, 0) {
        ...
        GUARDIAN = _guardian;
        ...
    }
```

https://github.com/ethereum-optimism/optimism/blob/4a01d2750ea10ad1109ff643faea2d8cfb28013f/packages/contracts-bedrock/contracts/L1/SystemConfig.sol#L93-L111

```
    constructor(
        address _owner,
        ...
        address _unsafeBlockSigner,
        ...
    ) Semver(1, 3, 0) {
        initialize({
            _owner: _owner,
            ...
            _unsafeBlockSigner: _unsafeBlockSigner,
            ...
        });
    }
```

https://github.com/ethereum-optimism/optimism/blob/365367e6716d65ba45b583d72d9b030fbb5c0e47/packages/contracts-bedrock/contracts/universal/OptimismMintableERC20Factory.sol#L55-L57

```
    constructor(address _bridge) Semver(1, 1, 0) {
        BRIDGE = _bridge;
    }
```

https://github.com/ethereum-optimism/optimism/blob/3b8fcafab4438b17e019c72391f829450a94a434/packa

ges/contracts-bedrock/contracts/universal/ProxyAdmin.sol#L75-L77

```
    constructor(address _owner) Ownable() {
        _transferOwnership(_owner);
    }
```

[11] VULNERABILITIES IN VERSION 4.7.3 OF `@openzeppelin/contracts` AND `@openzeppelin/contracts-upgradeable`

As shown in the following code in `package.json`, version 4.7.3 of `@openzeppelin/contracts` and `@openzeppelin/contracts-upgradeable` are used. As described in https://security.snyk.io/package/npm/@openzeppelin%2Fcontracts/4.7.3 and https://security.snyk.io/package/npm/@openzeppelin%2Fcontracts-upgradeable/4.7.3, this version of `@openzeppelin/contracts` and `@openzeppelin/contracts-upgradeable` has the Missing Authorization, DOS, and Improper Input Validation vulnerabilities. To reduce the potential attack surface and be more future-proofed, please consider upgrading these packages to at least version 4.9.1.

https://github.com/ethereum-optimism/optimism/blob/086d7fb67d2b7e0d1a31042e3772eac0649bfcb9/packages/contracts-bedrock/package.json#L55-L61

```
  "dependencies": {
    ...
    "@openzeppelin/contracts": "4.7.3",
    "@openzeppelin/contracts-upgradeable": "4.7.3",
    ...
  },
```

[12] MORE UPDATED VERSION OF SOLIDITY CAN BE USED

Using the more updated version of Solidity can add new features and enhance security. As described in https://github.com/ethereum/solidity/releases, Version

`0.8.20` is the latest version of Solidity, which includes support for Shanghai. If Optimism does not support `PUSH0` at this moment, Version `0.8.19`, which "contains a fix for a long-standing bug that can result in code that is only used in creation code to also be included in runtime bytecode", can also be used. To be more secured and future-proofed, please consider using the more updated version of Solidity for the following contracts:

```
optimism\packages\contracts-bedrock\contracts\deployment\SystemI
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L1\L1CrossDomainMe
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L1\L1StandardBridg
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L1\L2OutputOracle.
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L1\OptimismPortal.
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L1\SystemConfig.so
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L2\GasPriceOracle.
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L2\L1Block.sol
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L2\L2CrossDomainMe
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L2\L2StandardBridg
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L2\L2ToL1MessagePa
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\L2\SequencerFeeVau
  2: pragma solidity 0.8.15;

optimism\packages\contracts-bedrock\contracts\universal\Optimism
```

```
2: pragma solidity 0.8.15;
```

```
optimism\packages\contracts-bedrock\contracts\universal\ProxyAdm
2: pragma solidity 0.8.15;
```

[13] `require(_gasLimit >= minimumGasLimit(), "SystemConfig: gas limit too low")` CAN BE REFACTORED INTO A MODIFIER TO BE USED IN CORRESPONDING FUNCTIONS

`require(_gasLimit >= minimumGasLimit(), "SystemConfig: gas limit too low")` is executed in both of the `SystemConfig.initialize` and `SystemConfig.setGasLimit` functions. For better code organization and maintainability, please consider refactoring, such as a `require` statement, into a modifier to be used in these functions.

https://github.com/ethereum-optimism/optimism/blob/4a01d2750ea10ad1109ff643faea2d8cfb28013f/packages/contracts-bedrock/contracts/L1/SystemConfig.sol#L125-L143

```
function initialize(
    address _owner,
    uint256 _overhead,
    uint256 _scalar,
    bytes32 _batcherHash,
    uint64 _gasLimit,
    address _unsafeBlockSigner,
    ResourceMetering.ResourceConfig memory _config
) public initializer {
    __Ownable_init();
    transferOwnership(_owner);
    overhead = _overhead;
    scalar = _scalar;
    batcherHash = _batcherHash;
    gasLimit = _gasLimit;
    _setUnsafeBlockSigner(_unsafeBlockSigner);
    _setResourceConfig(_config);
    require(_gasLimit >= minimumGasLimit(), "SystemConfig: g
}
```

https://github.com/ethereum-optimism/optimism/blob/4a01d2750ea10ad1109ff643faea2d8cfb28013f/packages/contracts-bedrock/contracts/L1/SystemConfig.sol#L218-L224

```
function setGasLimit(uint64 _gasLimit) external onlyOwner {
    require(_gasLimit >= minimumGasLimit(), "SystemConfig: g
    gasLimit = _gasLimit;

    bytes memory data = abi.encode(_gasLimit);
    emit ConfigUpdate(VERSION, UpdateType.GAS_LIMIT, data);
}
```

[14] CONSTANTS CAN BE USED INSTEAD OF MAGIC NUMBERS

To improve readability and maintainability, a constant can be used instead of the magic number. Please consider replacing the magic numbers, such as `16`, used in the following code with constants:

```
optimism\packages\contracts-bedrock\contracts\L1\OptimismPortal.
  197: return _byteCount * 16 + 21000;
  461: require(_data.length <= 120_000, "OptimismPortal: data tc

optimism\packages\contracts-bedrock\contracts\L2\GasPriceOracle.
  122: total += 4;
  124: total += 16;
  128: return unsigned + (68 * 16);
```

[15] HARDCODED STRING THAT IS REPEATEDLY USED CAN BE REPLACED WITH A CONSTANT

`ProxyAdmin: unknown proxy type` is repeatedly used in the `ProxyAdmin.getProxyImplementation`, `ProxyAdmin.getProxyAdmin`, and `ProxyAdmin.changeProxyAdmin` functions. For better maintainability, please consider creating and using a constant for `ProxyAdmin: unknown proxy type` instead of hardcoding `ProxyAdmin: unknown proxy type` in these functions:

https://github.com/ethereum-optimism/optimism/blob/3b8fcafab4438b17e019c72391f829450a94a434/packages/contracts-bedrock/contracts/universal/ProxyAdmin.sol#L153-L164

```
    function getProxyImplementation(address _proxy) external vie
        ProxyType ptype = proxyType[_proxy];
        ...
        } else {
            revert("ProxyAdmin: unknown proxy type");
        }
    }
```

https://github.com/ethereum-optimism/optimism/blob/3b8fcafab4438b17e019c72391f829450a94a434/packages/contracts-bedrock/contracts/universal/ProxyAdmin.sol#L173-L184

```
    function getProxyAdmin(address payable _proxy) external viev
        ProxyType ptype = proxyType[_proxy];
        ...
        } else {
            revert("ProxyAdmin: unknown proxy type");
        }
    }
```

https://github.com/ethereum-optimism/optimism/blob/3b8fcafab4438b17e019c72391f829450a94a434/packages/contracts-bedrock/contracts/universal/ProxyAdmin.sol#L192-L203

```
    function changeProxyAdmin(address payable _proxy, address _r
        ProxyType ptype = proxyType[_proxy];
        ...
        } else {
            revert("ProxyAdmin: unknown proxy type");
        }
    }
```

[16] `revert` CAN BE CALLED INSTEAD OF `assert` TO PROVIDE CLEARER REASON FOR REVERSION

When the `ProxyAdmin.upgrade` function reverts due to `assert(false)`, it can be less clear about why such reversion happens since no reason is returned. To provide clearer reason for such reversion, please consider updating the `ProxyAdmin.upgrade` function to call `revert` with an appropriate reason string instead of executing `assert(false)` in the corresponding `else` block.

https://github.com/ethereum-optimism/optimism/blob/3b8fcafab4438b17e019c72391f829450a94a434/packages/contracts-bedrock/contracts/universal/ProxyAdmin.sol#L211-L229

```
    function upgrade(address payable _proxy, address _implementa
        ProxyType ptype = proxyType[_proxy];
        if (ptype == ProxyType.ERC1967) {
            Proxy(_proxy).upgradeTo(_implementation);
        } else if (ptype == ProxyType.CHUGSPLASH) {
            L1ChugSplashProxy(_proxy).setStorage(
                // bytes32(uint256(keccak256('eip1967.proxy.impl
                0x360894a13ba1a3210667c828492db98dca3e2076cc373
                bytes32(uint256(uint160(_implementation)))
            );
        } else if (ptype == ProxyType.RESOLVED) {
            string memory name = implementationName[_proxy];
            addressManager.setAddress(name, _implementation);
        } else {
            // It should not be possible to retrieve a ProxyType
            // one of the previous conditions.
            assert(false);
        }
    }
```

[17] PUBLIC FUNCTIONS THAT ARE NOT CALLED BY OTHER FUNCTIONS IN SAME CONTRACT CAN BE EXTERNAL

The `GasPriceOracle.gasPrice`, `GasPriceOracle.baseFee`, and `GasPriceOracle.decimals` functions are not called by other functions in the `GasPriceOracle` contract. Thus, the visibilities of these functions can be external instead of public.

https://github.com/ethereum-optimism/optimism/blob/404267b7d2cc2842d7fbf9bdfb92ac248eed15ef/packages/contracts-bedrock/contracts/L2/GasPriceOracle.sol#L57-L59

```
function gasPrice() public view returns (uint256) {
    return block.basefee;
}
```

https://github.com/ethereum-optimism/optimism/blob/404267b7d2cc2842d7fbf9bdfb92ac248eed15ef/packages/contracts-bedrock/contracts/L2/GasPriceOracle.sol#L66-L68

```
function baseFee() public view returns (uint256) {
    return block.basefee;
}
```

https://github.com/ethereum-optimism/optimism/blob/404267b7d2cc2842d7fbf9bdfb92ac248eed15ef/packages/contracts-bedrock/contracts/L2/GasPriceOracle.sol#L103-L105

```
function decimals() public pure returns (uint256) {
    return DECIMALS;
}
```

[18] UNDERSCORE CAN BE ADDED FOR `21000` IN `OptimismPortal.minimumGasLimit` FUNCTION

It is a common practice to separate each 3 digits in a number by an underscore to improve code readability. `21000` can be updated to `21_000` in the following `OptimismPortal.minimumGasLimit` function.

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L196-L198

```
function minimumGasLimit(uint64 _byteCount) public pure retu
    return _byteCount * 16 + 21000;
}
```

[19] WORD TYPING TYPO

The following code comment uses two “to’s” in “having to to use”. Please consider changing this phrase to “having to use”.

https://github.com/ethereum-optimism/optimism/blob/536178b0e28f7015e036ad050945ea5633dacf02/packages/contracts-bedrock/contracts/deployment/SystemDictator.sol#L168-L169

```
        // Using this shorter variable as an alias for address(t
        // to use a new line for every single parameter.
```

[20] BOOLEAN VARIABLE COMPARISONS ARE NOT HANDLED CONSISTENTLY

When checking whether a boolean variable is true or false, some code explicitly compares it to `true` or `false`, such as the following code:

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L138

```
require(paused == false, ...);
```

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L387-L390

```
require(
    finalizedWithdrawals[withdrawalHash] == false,
    ...
```

```
        );
```

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L417-L419

```
        if (success == false && tx.origin == Constants.ESTIMATION_AI
            ...
        }
```

Yet, some other codes do not explicitly compare it to `true` or `false`, such as the following code:

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L443-L448

```
        if (_isCreation) {
            ...
        }
```

To improve code consistency, please consider updating the relevant code to handle the boolean variable comparisons consistently. If in favor of code readability, the relevant boolean variables can all be explicitly compared to `true` or `false`. Otherwise, if in favor of code efficiency, the relevant boolean variables can all utilize `!` when needed and not be explicitly compared to `true` or `false`.

[21] `revert` WITH CUSTOM ERRORS CAN BE EXECUTED INSTEAD OF EXECUTING `require` OR `revert` WITH REASON STRINGS

As mentioned here, executing `revert` with a custom error can be more efficient than executing `require` or `revert` with a reason string. The following examples are some where `require` or `revert` is executed. To make the code more efficient,

please consider using `revert` statements with custom errors to replace the relevant `require` and `revert` statements.

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L434-L483

```
function depositTransaction(
    address _to,
    uint256 _value,
    uint64 _gasLimit,
    bool _isCreation,
    bytes memory _data
) public payable metered(_gasLimit) {
    ...
    if (_isCreation) {
        require(
            _to == address(0),
            "OptimismPortal: must send to address(0) when cr
        );
    }

    ...
    require(
        _gasLimit >= minimumGasLimit(uint64(_data.length)),
        "OptimismPortal: gas limit too small"
    );

    ...
    require(_data.length <= 120_000, "OptimismPortal: data t

    ...
}
```

https://github.com/ethereum-optimism/optimism/blob/536178b0e28f7015e036ad050945ea5633dacf02/packages/contracts-bedrock/contracts/deployment/SystemDictator.sol#L338-L411

```
function step5() public onlyOwner step(5) {
    ...
    require(dynamicConfigSet, "SystemDictator: dynamic oracl
```

```
        ...
        try
            L1CrossDomainMessenger(config.proxyAddressConfig.l1C
                .initialize()
        {
            ...
        } catch Error(string memory reason) {
            require(
                keccak256(abi.encodePacked(reason)) ==
                    keccak256("Initializable: contract is alreac
                string.concat("SystemDictator: unexpected error
            );
        } catch {
            revert("SystemDictator: unexpected error initializir
        }
        ...
    }
```

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L325-L420

```
    function finalizeWithdrawalTransaction(Types.WithdrawalTrans
        external
        whenNotPaused
    {
        ...
        require(
            l2Sender == Constants.DEFAULT_L2_SENDER,
            "OptimismPortal: can only trigger one withdrawal per
        );
        ...
        require(
            provenWithdrawal.timestamp != 0,
            "OptimismPortal: withdrawal has not been proven yet"
        );
        ...
        require(
            provenWithdrawal.timestamp >= L2_ORACLE.startingTime
            "OptimismPortal: withdrawal timestamp less than L2 C
        );
        ...
        require(
```

```
            _isFinalizationPeriodElapsed(provenWithdrawal.timest
            "OptimismPortal: proven withdrawal finalization peri
        );
        ...
        require(
            proposal.outputRoot == provenWithdrawal.outputRoot,
            "OptimismPortal: output root proven is not the same
        );
        ...
        require(
            _isFinalizationPeriodElapsed(proposal.timestamp),
            "OptimismPortal: output proposal finalization perioc
        );
        ...
        require(
            finalizedWithdrawals[withdrawalHash] == false,
            "OptimismPortal: withdrawal has already been finaliz
        );
        ...
        if (success == false && tx.origin == Constants.ESTIMATIC
            revert("OptimismPortal: withdrawal failed");
        }
    }
```

[22] ORDERS OF FUNCTIONS DO NOT FOLLOW OFFICIAL STYLE GUIDE

Order of functions suggests that functions in a contract should be grouped and ordered as follows with the `view` and `pure` functions being placed last within each group:

1. constructor
2. receive function (if exists)
3. fallback function (if exists)
4. external
5. public
6. internal
7. private

The following order of functions are some examples that do not follow the official style guide. Please consider updating the relevant order of functions accordingly.

The `OptimismPortal.receive` function is placed after the `OptimismPortal.minimumGasLimit` public function:
https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L196-L209

```
function minimumGasLimit(uint64 _byteCount) public pure retu
    return _byteCount * 16 + 21000;
}

...
receive() external payable {
    depositTransaction(msg.sender, msg.value, RECEIVE_DEFAUI
}
```

The `OptimismPortal.donateETH` external function is placed after the `OptimismPortal.minimumGasLimit` public function:
https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L196-L217

```
function minimumGasLimit(uint64 _byteCount) public pure retu
    return _byteCount * 16 + 21000;
}

...
function donateETH() external payable {
    // Intentionally empty.
}
```

The `SystemConfig.resourceConfig` external view function is placed before the `SystemConfig.setResourceConfig` external non-view function:
https://github.com/ethereum-optimism/optimism/blob/4a01d2750ea10ad1109ff643faea2d8cfb28013f/packages/contracts-bedrock/contracts/L1/SystemConfig.sol#L245-L258

```
function resourceConfig() external view returns (ResourceMet
    return _resourceConfig;
}

...
function setResourceConfig(ResourceMetering.ResourceConfig m
    _setResourceConfig(_config);
}
```

[23] INCOMPLETE NATSPEC COMMENTS

NatSpec comments provide rich code documentation. The following functions miss the `@param` and/or `@return` comments. Please consider completing the NatSpec comments for these functions.

The `@param` comment for `_finalizationPeriodSeconds` is missing for the following constructor of the `L2OutputOracle` contract:

https://github.com/ethereum-optimism/optimism/blob/d322c6d651022ceb0798168726fe47416c6ddf00/packages/contracts-bedrock/contracts/L1/L2OutputOracle.sol#L80-L112

```
/**
 * @custom:semver 1.3.0
 *
 * @param _submissionInterval  Interval in blocks at which c
 * @param _l2BlockTime         The time per L2 block, in sec
 * @param _startingBlockNumber The number of the first L2 bl
 * @param _startingTimestamp   The timestamp of the first L2
 * @param _proposer            The address of the proposer.
 * @param _challenger          The address of the challenger
 */
constructor(
    uint256 _submissionInterval,
    uint256 _l2BlockTime,
    uint256 _startingBlockNumber,
    uint256 _startingTimestamp,
    address _proposer,
    address _challenger,
    uint256 _finalizationPeriodSeconds
) Semver(1, 3, 0) {
```

```
    ...
}
```

The `@param` comment for `_paused` is missing for the following `OptimismPortal.initialize` function:

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L162-L169

```
/**
 * @notice Initializer.
 */
function initialize(bool _paused) public initializer {
    l2Sender = Constants.DEFAULT_L2_SENDER;
    paused = _paused;
    __ResourceMetering_init();
}
```

The `@param` comment for `_byteCount` and `@return` comment are missing for the following `OptimismPortal.minimumGasLimit` function:

https://github.com/ethereum-optimism/optimism/blob/6eb05430d1ec1ae18ee96c2a206c60cc80fcbcf6/packages/contracts-bedrock/contracts/L1/OptimismPortal.sol#L189-L198

```
/**
 * @notice Computes the minimum gas limit for a deposit. The
 *         linearly increases based on the size of the callc
 *         users from creating L2 resource usage without pay
 *         can be used when interacting with the portal to e
 *
 */
function minimumGasLimit(uint64 _byteCount) public pure retu
    return _byteCount * 16 + 21000;
}
```

Oxleastwood (judge) commented:

> I agree with the findings raised here.

Gas Optimizations

For this audit, 4 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **souilos** received the top score from the judge.

The following wardens also submitted reports: koxuan, judeabara *and* fatherOfBlocks.

[01] Use `!= 0` instead of `> 0` for unsigned integer comparison

Proof of concept

Found in line 99 at optimismContest/L1/L2OutputOracle.sol:

```
        require(_l2BlockTime > 0, "L2OutputOracle: L2 block time mus
```

Found in line 101 at optimismContest/L1/L2OutputOracle.sol:

```
            _submissionInterval > 0,
```

Found in line 285 at optimismContest/L1/SystemConfig.sol:

```
            _config.elasticityMultiplier > 0,
```

Found in line 100 at optimismContest/L1/ResourceMetering.sol:

```
        if (blockDiff > 0) {
```

[02] Use shift Right/Left instead of division/multiplication if possible

Proof of concept

Found in line 272 at optimismContest/L1/L2OutputOracle.sol:

```
            uint256 mid = (lo + hi) / 2;
```

[03] `++i/i++` should be unchecked `{++i}/unchecked{i++}` and `++i` costs less gas than `i++`

Proof of concept

Found in line 120 at optimismContest/L2/GasPriceOracle.sol:

```
        for (uint256 i = 0; i < length; i++) {
```

Mitigation

`++i/i++` should be unchecked `{++i}/unchecked{i++}` when it is not possible for them to overflow, as in the case when used in “for” and “while” loops. Moreover, `++i` costs less gas than `i++`, especially when its used in “for” loops `(--i/i--too)`.

[04] Don’t initialize variables with default value

Proof of concept

Found in line 118 at optimismContest/L2/GasPriceOracle.sol:

```
        uint256 total = 0;
```

Found in line 120 at optimismContest/L2/GasPriceOracle.sol:

```
        for (uint256 i = 0; i < length; i++) {
```

Found in line 269 at optimismContest/L1/L2OutputOracle.sol:

```
        uint256 lo = 0;
```

Mitigation

In such cases, initializing the variables with default values would be unnecessary and can be considered a waste of gas. Additionally, initializing variables with default values can sometimes lead to unnecessary storage operations, which can increase gas costs. For example, if you have a large array of variables, initializing them all with default values can result in a lot of unnecessary storage writes, which can increase the gas costs of your contract.

[05] Use Custom Errors

Proof of concept

There are 14 instances of this issue. (For in-depth details on this and all further gas optimizations with multiple instances, please see the warden's full report).

Mitigation

Instead of using error strings to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

[06] Use `calldata` instead of `memory` for function arguments that do not get mutated

Proof of concept

Found in line 43 at optimismContest/L2/GasPriceOracle.sol:

```
    function getL1Fee(bytes memory _data) external view returns (uir
```

Found in line 117 at optimismContest/L2/GasPriceOracle.sol:

```
    function getL1GasUsed(bytes memory _data) public view returns (u
```

Found in line 325 at optimismContest/L1/OptimismPortal.sol:

```
    function finalizeWithdrawalTransaction(Types.WithdrawalTransacti
```

Found in line 256 at optimismContest/L1/SystemConfig.sol:

```
    function setResourceConfig(ResourceMetering.ResourceConfig memor
```

Found in line 266 at optimismContest/L1/SystemConfig.sol:

```
    function _setResourceConfig(ResourceMetering.ResourceConfig memc
```

Mitigation

Mark data types as `calldata` instead of `memory` where possible. This makes it so that the data is not automatically loaded into `memory`. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`. The one exception is if the argument must later be passed into another function, which takes an argument that specifies `memory` storage.

[07] Use `assembly` to check for `address(0)`

Proof of concept

Found in line 151 at optimismContest/L2/L2StandardBridge.sol:

```
        if (_l1Token == address(0) && _l2Token == Predeploys.LEGACY_
```

Found in line 445 at optimismContest/L1/OptimismPortal.sol:

```
                _to == address(0),
```

Mitigation

Using `assembly` to check for the zero address can result in significant gas savings compared to using a Solidity expression; especially if the check is performed frequently or in a loop. However, it's important to note that using `assembly` can make the code less readable and harder to maintain, so it should be used judiciously and with caution.

[08] Usage of `uints/ints` smaller than 32 bytes (256 bits) incurs overhead

Proof of concept

There are 17 instances of this issue.

Mitigation

When using elements that are smaller than 32 bytes, your contract's gas usage may be higher. This is because the EVM operates on 32 bytes at a time. Therefore, if the element is smaller than that, the EVM must use more operations in order to reduce the size of the element from 32 bytes to the desired size. See **here**.

Each operation involving a `uint8` costs an extra 22-28 gas (depending on whether the other operation is also a variable of type `uint8`), compared to ones involving `uint256`. This is due to the compiler having to clear the higher bits of the memory word before operating on the `uint8`, as well as the associated stack operations of doing so. Use a larger size, then downcast where needed.

[09] `+=` Costs More Gas Than `=+` For State Variables

Proof of concept

Found in line 122 at optimismContest/L2/GasPriceOracle.sol:

```
                total += 4;
```

Found in line 124 at optimismContest/L2/GasPriceOracle.sol:

```
                total += 16;
```

Found in line 141 at optimismContest/L1/ResourceMetering.sol:

```
        params.prevBoughtGas += _amount;
```

Mitigation

When you use the `+=` operator on a state variable, the EVM has to perform three operations:

- Load the current value of the state variable.
- Add the new value to it.
- Then store the result back in the state variable.

On the other hand, when you use the `=` operator and then add the values separately, the EVM only needs to perform two operations:

- Load the current value of the state variable.
- Add the new value to it.

Better use `=+` For State Variables.

[10] Use hardcode `address` instead `address(this)`

Proof of concept

There are 8 instances of this issue.

Mitigation

Instead of using `address(this)`, it is more gas-efficient to pre-calculate and use the hardcoded address. Foundry's `script.sol` and solmate's `LibRlp.sol` contracts can help achieve this.

[11] Functions guaranteed to revert when called by normal users can be marked payable

Proof of concept

There are 6 instances of this issue.

Mitigation

If a function `modifier` or `require` such as `onlyOwner/onlyX` is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are `CALLVALUE` (2), `DUP1` (3), `ISZERO` (3), `PUSH2` (3), `JUMPI` (10), `PUSH1` (3), `DUP1` (3), `REVERT` (0), `JUMPDEST` (1) and `POP` (2), which costs an average of about 21 gas per call to the function, in addition to the extra deployment cost.

[12] Don't compare boolean expressions to boolean literals

Proof of concept

Found in line 417 at optimismContest/L1/OptimismPortal.sol:

```
        if (success == false && tx.origin == Constants.ESTIMATION_AI
```

Mitigation

In Solidity, when a boolean expression is compared to a boolean literal, it can result in additional gas costs due to the additional comparison operation that is performed.

[13] Using private rather than public for constants, saves gas

Proof of concept

Found in line 20 at optimismContest/L1/L2OutputOracle.sol:

```
    uint256 public immutable SUBMISSION_INTERVAL;
```

Found in line 25 at optimismContest/L1/L2OutputOracle.sol:

```
    uint256 public immutable L2_BLOCK_TIME;
```

Found in line 40 at optimismContest/L1/L2OutputOracle.sol:

```
    uint256 public immutable FINALIZATION_PERIOD_SECONDS;
```

Mitigation

If needed, the values can be read from the verified contract source code. If there are multiple values, there can be a single getter function that returns a tuple of the values of all currently-public constants. Saves 3406-3606 in deployment gas due to the compiler not having to create non-payable getter functions for deployment `calldata`; this is by not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table.

[14] `>=` costs less gas than `>`

Proof of concept

Found in line 505 at optimismContest/L1/OptimismPortal.sol:

```
        return block.timestamp > _timestamp + L2_ORACLE.FINALIZATIO
```

Mitigation

The compiler uses opcodes `GT` and `ISZERO` for solidity code that uses `>`, but only requires `LT` for `>=`, which saves 3 gas.

[15] Empty blocks should be removed or emit something

Proof of concept

There are 10 instances of this issue.

Mitigation

The code should be refactored so they no longer exist; or the block should do something useful, such as emitting an event or reverting. If the contract is meant to be extended, the contract should be abstract and the function signatures be added without any default implementation. If the block is an empty if-statement block to avoid doing subsequent checks in the else-if/else conditions, the else-if/else conditions should be nested under the negation of the if-statement, because they involve different classes of checks, which may lead to the introduction of errors when the code is later modified `(if(x){}else if(y){...}else{...} => if(!x){if(y){...}else{...}})`.

[16] Use `assembly` to write address storage values

Proof of concept

Found in lines 150 to 155 at optimismContest/L1/OptimismPortal.sol:

```
    constructor(
        L2OutputOracle _l2Oracle,
        address _guardian,
        bool _paused,
        SystemConfig _config
    ) Semver(1, 6, 0) {
```

Found in lines 90 to 98 at optimismContest/L1/L2OutputOracle.sol:

```
    constructor(
        uint256 _submissionInterval,
        uint256 _l2BlockTime,
        uint256 _startingBlockNumber,
        uint256 _startingTimestamp,
```

```
        address _proposer,
        address _challenger,
        uint256 _finalizationPeriodSeconds
    ) Semver(1, 3, 0) {
```

Found in lines 93 to 101 at optimismContest/L1/SystemConfig.sol:

```
    constructor(
        address _owner,
        uint256 _overhead,
        uint256 _scalar,
        bytes32 _batcherHash,
        uint64 _gasLimit,
        address _unsafeBlockSigner,
        ResourceMetering.ResourceConfig memory _config
    ) Semver(1, 3, 0) {
```

Mitigation

Use `assembly` to write address storage values. Here are a few reasons:

- Reduced opcode usage: When using `assembly`, you can directly manipulate storage values using lower-level instructions like `sstore` (storage store) instead of relying on higher-level Solidity storage assignments. These direct operations typically result in fewer opcode executions, reducing gas costs.
- Avoiding unnecessary checks: Solidity storage assignments often involve additional checks and operations, such as enforcing security modifiers or triggering events. By using `assembly`, you can bypass these additional checks and perform the necessary storage operations directly, resulting in gas savings.
- Optimized packing: `Assembly` provides greater flexibility in packing and unpacking data structures. By carefully arranging and manipulating the storage layout in `assembly`, you can achieve more efficient storage utilization and minimize wasted storage space.
- Fine-grained control: `Assembly` allows for precise control over gas-consuming operations. You can optimize gas usage by selecting specific instructions and minimizing unnecessary operations or data copying.

[17] Use `ERC721A` instead `ERC721`

Proof of concept

There are 31 instances of this issue.

Mitigation

`ERC721A` is an improvement standard for `ERC721` tokens. It was proposed by the Azuki team and used for developing their NFT collection. Compared with `ERC721`, `ERC721A` is a more gas-efficient standard to mint a lot of of NFTs simultaneously. It allows developers to mint multiple NFTs at the same gas price. This has been a great improvement due to Ethereum's sky-rocketing gas fee. Reference: https://nextrope.com/erc721-vs-erc721a-2/.

[18] `require()` or `revert()` statements that check input arguments should be at the top of the function (Also restructured some "if's")

Proof of concept

There are 45 instances of this issue.

Mitigation

Checks that involve constants should come before checks that involve state variables, function calls, and calculations. By doing these checks first, the function is able to revert before wasting alot of gas in a function that may ultimately revert in the unhappy case.

Disclosures

C4 is an open organization governed by participants in the community.

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct

formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top