Optimism PBC U18

Security Review

Cantina Managed review by:
Alireza Arjmand, Lead Security Researcher
LonelySloth, Lead Security Researcher
Mustafa Hasan, Lead Security Researcher

January 12, 2026

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Optimism is a fast, stable, and scalable L2 blockchain built by Ethereum developers, for Ethereum developers. Built as a minimal extension to existing Ethereum software, Optimism's EVM-equivalent architecture scales your Ethereum apps without surprises. If it works on Ethereum, it works on Optimism at a fraction of the cost.

From Nov 26th to Dec 3rd the Cantina team conducted a review of optimism on commit hash 87d406db. The team identified a total of **7** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	4	0	4
Gas Optimizations	1	0	1
Informational	2	0	2
Total	**7**	**0**	**7**

2.1 Scope

The security review had the following components in scope for optimism on commit hash 87d406db:

```
packages/contracts-bedrock/src
├── dispute
│   ├── AnchorStateRegistry.sol
│   ├── DisputeGameFactory.sol
│   ├── lib
│   │   ├── Errors.sol
│   │   ├── LibGameArgs.sol
│   │   └── Types.sol
│   ├── SuperFaultDisputeGame.sol
│   ├── SuperPermissionedDisputeGame.sol
│   ├── v2
│   │   ├── FaultDisputeGameV2.sol
│   │   └── PermissionedDisputeGameV2.sol
│   └── zk
│       ├── AccessManager.sol
│       ├── ISP1Verifier.sol
│       └── OPSuccinctFaultDisputeGame.sol
├── L1
│   ├── FeesDepositor.sol
│   ├── OPContractsManager.sol
│   ├── OPContractsManagerStandardValidator.sol
│   ├── OptimismPortal2.sol
│   └── SystemConfig.sol
├── L2
│   ├── BaseFeeVault.sol
│   ├── FeeSplitter.sol
│   ├── FeeVault.sol
│   ├── L1Block.sol
│   ├── L1BlockCGT.sol
│   ├── L1FeeVault.sol
│   ├── L1Withdrawer.sol
│   ├── L2ToL1MessagePasser.sol
│   ├── L2ToL1MessagePasserCGT.sol
│   ├── LiquidityController.sol
```

```
│   ├── NativeAssetLiquidity.sol
│   ├── OperatorFeeVault.sol
│   ├── SequencerFeeVault.sol
│   └── SuperchainRevSharesCalculator.sol
├── libraries
│   ├── Constants.sol
│   ├── DevFeatures.sol
│   ├── Features.sol
│   └── Predeploys.sol
└── safe
    ├── DeputyPauseModule.sol
    ├── LivenessGuard.sol
    ├── LivenessModule.sol
    ├── LivenessModule2.sol
    ├── SaferSafes.sol
    └── TimelockGuard.sol
```

3 Findings

3.1 Low Risk

3.1.1 `DisputeGameFactory.setImplementation` allows for setting new implementation with outdated parameters

Severity: Low Risk

Context: DisputeGameFactory.sol#L296

Description: The function `setImplementation(GameType,IDisputeGame)` in `DisputeGameFactory` allows for changing the implementation for a game type without either setting or clearing existing arguments.

That means if an implementation is set with arguments, then later the version without arguments is invoked, the final state can be the use of outdated arguments, leading potentially to undefined behaviour.

While the damaging result needs an incorrect action by a privileged role, the likelihood of simply of using the wrong version of the function (same name) seems considerable.

Proof of Concept:

```
function test_setImplementation_withArgs_succeeds() public {
    address fakeGame = address(1);
    Claim absolutePrestate = Claim.wrap(bytes32(hex"dead"));
    AlphabetVM vm_;
    IPreimageOracle preimageOracle_;
    (vm_, preimageOracle_) = _createVM(absolutePrestate);

    bytes memory args = abi.encodePacked(
        absolutePrestate, // 32 bytes
        vm_, // 20 bytes
        anchorStateRegistry, // 20 bytes
        delayedWeth, // 20 bytes
        l2ChainId // 32 bytes (l2ChainId)
    );


    // Set the implementation and args for the `GameTypes.CANNON` enum value.
    disputeGameFactory.setImplementation(GameTypes.CANNON, IDisputeGame(fakeGame), args);

    assertEq(address(disputeGameFactory.gameImpls(GameTypes.CANNON)), address(1));
    assertEq(disputeGameFactory.gameArgs(GameTypes.CANNON), args);


    // We set implementation again, but without args.
    disputeGameFactory.setImplementation(GameTypes.CANNON, IDisputeGame(address(2)));

    // Ensure that the implementation for the `GameTypes.CANNON` enum value is set.
    assertEq(address(disputeGameFactory.gameImpls(GameTypes.CANNON)), address(2));

    // The old args are still there!!!!
    assertEq(disputeGameFactory.gameArgs(GameTypes.CANNON), args);
}
```

Recommendation: The `setImplementation(GameType,IDisputeGame)` function should require that there are no arguments set for the game type, reverting in case an argument has been set.

OP Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.1.2 Potential calldata injection in dispute games

Severity: Low Risk

Context: DisputeGameFactory.sol#L197

Description: Clones with immutable arguments passed in calldata (as implemented in this code base) suffer in general from the potential for calls such as `address(game).call(bytes(""))` being mishandled as calls to specific selectors -- as the additional calldata added by the proxy can start with a valid 4-byte selector. Such potential issues are only relevant if:

1. An external function is permissioned or result in different states depending on `msg.sender`.
2. The permissioned role is a contract that can be caused to perform `address(game).call(bytes(""))` -- e.g. sending ETH (or a user phished into doing so, though it seems to add little risk compared to phishing in general).

The conditions seem exceedingly unlikely for existing dispute game contracts and planned use cases.

However it might be hard to determine how safe from such attacks contracts are in the long run, as any privileged role could be an arbitrary contract.

Recommendation: Ideally, completely preventing that class of attacks would require checking `msg.data` contains at least `4 + len(immutable args) bytes`. This would guarantee that the sender deliberately called the function being executed.

The check would be necessary in every function that has behaviour that depends on `msg.sender`. Ideally this should be added as a modifier in all functions to prevent selective enforcement being implemented wrong.

While it would add a little complexity and gas costs, it would make any such attack absolutely impossible instead of simply very unlikely.

OP Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.1.3 New `DisputeGameFactory` allows creation of games using incompatible old implementations

Severity: Low Risk

Context: DisputeGameFactory.sol#L201

Description: The new version of the `DisputeGameFactory` shouldn't be able to use old-style implementations. From the scope documentation:

> After upgrading OPCM, all game implementations that existed before the upgrade must be fully migrated to use the creator pattern. The system should not contain any old-style dispute game implementations. All future OPCM operations after the upgrade must use the creator pattern only.

However, an attempt to create a clone with an old implementation isn't guaranteed to fail, resulting in potential invalid state in the case of mishandled or partial upgrade.

Recommendation: The factory should validate that the implementation is compatible with its creation strategy. For example, each implementation could implement:

```
function expectedFactoryVersion() returns (uint256) external view;
```

Then `create` could call this either on the implementation directly or in the deployed clone and match it with it's own version. Old-style implementations that expect non-CWIA cloning dont' implement such view and would revert during creation. Any further breaking change to how dispute games are cloned could be validated by increasing the version number.

OP Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.1.4 Withdrawals to the zero address leading to loss of funds

Severity: Low Risk

Context: `packages/contracts-bedrock/src/L2/FeeVault.sol`

Description: The `recipient` address can be zero in general (via the constructor and `setRecipient()`). The following logic exists in the `withdraw()` function:

```
if (withdrawalNetwork == Types.WithdrawalNetwork.L2) {
    bool success = SafeCall.send(recipientAddr, value_);
    require(success, "FeeVault: failed to send ETH to L2 fee recipient");
} else {
    IL2ToL1MessagePasser(payable(Predeploys.L2_TO_L1_MESSAGE_PASSER)).initiateWithdrawal
    ↪ { value: value_ }({
        _target: recipientAddr,
        _gasLimit: _WITHDRAWAL_MIN_GAS,
        _data: hex""
    });
}
```

Since there are no zero address checks, the value will be sent to the zero address if the recipient was not set, effectively burning the ETH.

Impact Explanation: Loss of funds.

Proof of Concept: A minimal proof of concept showing that at least L2 withdrawals will pass and send the ETH to the zero address is provided:

```
pragma solidity 0.8.30;
import { SafeCall } from "./SafeCall.sol";

contract Test {
    constructor() payable {} //Attach ETH while deploying

    function withdraw() external {
        bool success = SafeCall.send(address(0), address(this).balance); //Sending to the
        ↪  zero address does not revert
        require(success, "FeeVault: failed to send ETH to L2 fee recipient");
    }

    function balance() external view returns (uint256) { //Auxiliary function for balance
    ↪  display
        return address(this).balance;
    }
}
```

Recommendation: Validate against the zero address in the constructor and the `setRecipient()` function.

OP Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.2 Gas Optimization

3.2.1 Storage management improvement

Severity: Gas Optimization

Context: TimelockGuard.sol#L500

Description: The `_safeStates[_safe][_safeConfigNonces[_safe]]` is not cleared. Clearing up unused storage is generally good practice and it allows gas refunds.

Recommendation: Clear the unused storage values.

OP Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Game implementations might be unable to distinguish between game arguments and extra data

Severity: Informational

Context: DisputeGameFactory.sol#L192

Description: When game arguments are present the `create` function in `DisputeGameFactory` encodes both the provided extra data and the arguments by concatenation with `abi.encodePacked`.

Since both have variable length, it means implementations can't distinguish between between extra data and arguments unless there's careful encoding on the side of game implementations. That limitation doesn't seem to be documented.

In practice, existing game implementations expect always a statically defined number of bytes, and failure to provide the precise length will revert the initialization, making it an unusable attack vector at present.

However, if there's ever any game that accepts a variable length of immutable args this might become a problem.

Recommendation: After discussion with the development team, they proposed that any variable length game argument should contain the length at the end of the encoded data. That seems to be correct way of preventing the issue. The recommendation is that this design decision be documented such that it becomes clear that any future game implementation must be aware of the necessity to encode variable length arguments in this way.

OP Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Improper function naming

Severity: Informational

Context: LibGameArgs.sol#L88

Description: The name of the functions `isValidPermissionlessArgs()` and `isValidPermissionedArgs()` and the documentation text both imply validation of the args is performed, while only a length check takes place.

Recommendation: Update the functions names so they reflect the logic they perform.

OP Labs: Acknowledged.

Cantina Managed: Acknowledged.