OP Succinct

Security Review

Cantina Managed review by:
0xLeastwood, Lead Security Researcher
Jerome Rousselot, Security Researcher

December 20, 2024

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	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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

OP Succinct transforms any OP Stack rollup into a fully type-1 ZK rollup using SP1.

From Dec 3rd to Dec 8th the Cantina team conducted a review of op-succinct on commit hash b73cc7d0.

The scope of the the review includes the `OPSuccinctL2OutputOracle` contract, which verifies SP1 proofs of Optimism's STF on-chain and the aggregation program, which in turn verifies a set of range proofs and is posted on-chain.

The team identified a total of **6** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 1
- Gas Optimizations: 0
- Informational: 5

3 Findings

3.1 Low Risk

3.1.1 Data read from permissionless proposers can cause undefined behaviour in the range program due to unsafe cast

Severity: Low Risk

Context: main.rs#L107

Description: The range program reads data submitted on-chain by permissionless proposers. The `InMemoryOracle` object uses the `rkyv` library to perform high performance zero-copy object deserialization in the `Arc::new(InMemoryOracle::from_raw_bytes(in_memory_oracle_bytes))` call. This unsafe Rust code casts raw bytes into an `ARC` pointer on a `InMemoryOracle` object. This can cause undefined behaviour if the data is invalid. Most likely, the process will terminate.

Recommendation: Consider using another `rkyv` API using safe Rust, or try using `bytecheck` to validate the data before reading it, or another way to perform this conversion. These approaches can be compared by taking into account their performance cost.

Succinct: From our conversation, it's likely that hijacking this deserialization call with malicious data would not result in undefined behavior that would allow verifying invalid input to the program, so a fix PR isn't necessary. We may decide to address in the future to alleviate concerns with the API and for simplicity.

Cantina Managed: Acknowledged as a won't-fix for now.

3.2 Informational

3.2.1 Non-null input data is not verified

Severity: Informational

Context: main.rs#L28-L38

Description: The aggregation program reads data coming from `sp1_zkvm`. The `AggregationInputs.boot_infos` struct values are not verified to not to be empty. Hence, we should verify that both `boot_info.l2PreRoot` and the `boot_info.rollupConfigHash` are not null.

Recommendation: Consider adding the following assertions:

```
assert_ne!(agg_inputs.boot_infos[0].l2PreRoot.to_string(), "0");
assert_ne!( agg_inputs.boot_infos[0].rollupConfigHash.to_string(), "0");
```

Succinct: As long as the `OPSuccinctL2OutputOracle` contract is initialized correctly, `l2PreRoot` and `rollupConfigHash` will never be null for a valid chain. Given these are committed to as a part of the proof, there's no reason to check these.

Cantina Managed: Acknowledged.

3.2.2 L2 oracle upgrades do not reinitialize contract state as expected

Severity: Informational

Context: OPSuccinctL2OutputOracle.sol#L184

Description: OP Succinct attempts to enable existing OP-Stack chains to upgrade their `L2OutputOracle` to the new `OPSuccinctL2OutputOracle` contract by calling `upgradeToAndCall()` on the proxy. This sets the new implementation and subsequently calls `initialize()` as per the upgrader script.

This will consequently revert because the `initializer` modifier has already set `_initialized = 1`, preventing re-initialization. To make use of OpenZeppelin's reinitialization feature, the `reinitializer` modifier must replace the current modifier on the `initialize()` function. There are several new state variables introduced for enabling proof verification via the `SP1Verifier` which must be configurable from within the `initialize()` function.

Fortunately in-production this is easily caught because the entire upgrade fails and can be remedied by implementing the above suggestion.

Recommendation: Update the `initializer` modifier to `reinitializer` with proper version management.

Note: it might be worthwhile storing the current version as a constant.

```
/// @notice The current version of the contract.
uint8 public constant version = 2;

// ...

/// @notice Initializer.
/// @param _initParams The initialization parameters for the contract.
function initialize(InitParams memory _initParams) public reinitializer(version) {
    require(_initParams.submissionInterval > 0, "L2OutputOracle: submission interval must be greater than 0");
    require(_initParams.l2BlockTime > 0, "L2OutputOracle: L2 block time must be greater than 0");
    require(
        _initParams.startingTimestamp <= block.timestamp,
        "L2OutputOracle: starting L2 timestamp must be less than current time"
    );

    submissionInterval = _initParams.submissionInterval;
    l2BlockTime = _initParams.l2BlockTime;

    // For proof verification to work, there must be an initial output.
    // Disregard the _startingBlockNumber and _startingTimestamp parameters during upgrades, as they're
↪   already set.
    if (l2Outputs.length == 0) {
        l2Outputs.push(
            Types.OutputProposal({
                outputRoot: _initParams.startingOutputRoot,
                timestamp: uint128(_initParams.startingTimestamp),
                l2BlockNumber: uint128(_initParams.startingBlockNumber)
            })
        );

        startingBlockNumber = _initParams.startingBlockNumber;
        startingTimestamp = _initParams.startingTimestamp;
    }

    challenger = _initParams.challenger;
    finalizationPeriodSeconds = _initParams.finalizationPeriodSeconds;

    // Add the initial proposer.
    approvedProposers[_initParams.proposer] = true;

    // OP Succinct initialization parameters.
    aggregationVkey = _initParams.aggregationVkey;
    rangeVkeyCommitment = _initParams.rangeVkeyCommitment;
    verifier = _initParams.verifier;
    rollupConfigHash = _initParams.rollupConfigHash;
    owner = _initParams.owner;
}
```

Succinct: Resolved in PR 265 and documentation was updated in PR 264.

Cantina Managed: Verified fix. The `reinitializer` modiifer is now being used with proper in-contract version management.

3.2.3 Fresh OP-Stack deployments do not verify initial output

Severity: Informational

Context: OPSuccinctL2OutputOracle.sol#L197-L208

Description: When the proxy admin upgrades to the new `OPSuccinctL2OutputOracle` contract, the initial output is stored along with `startingBlockNumber` and `startingTimestamp`. However, this initial output is considered trusted and not verified by the `SP1Verifier` contract as seen in `proposeL2Output()`. While initially, new L2 outputs were treated optimistically, there is little drawback besides gas costs in verifying the initial output stored on contract initialization.

Recommendation: Considering the initializer is trusted for initializing `aggregationVkey` and `rangeVkeyCommitment`, it may only be worth documenting this assumption.

Succinct: It's not possible for the proxy admin to `initialize` the contract with a proof b/c the first proof requires "trusted" initialization state.

Cantina Managed: Acknowledged.

3.2.4 Upgrades may instantly finalize optimistically proven state roots

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The proxy admin of the `L2OutputOracle` contract can execute an upgrade to the new `OPSuccinctL2OutputOracle` contract which enables proof verification via the `SP1Verifier`. All existing `l2Outputs` will be preserved in the upgrade, hence, there will be many transactions in the `OptimismPortal` contract which are in the prove phase, awaiting execution of the finalize withdrawal action.

Consequently, any transaction in the `provenWithdrawals` mapping with a timestamp that has already elapsed the 1-hour finalization afforded by OP succinct will be able to instantly finalize upon upgrade. This seems to circumvent the 7-day expected withdrawal window required for optimistically proven transactions that have now been shorted to 1-hour even though the L2 output is potentially incorrect.

Recommendation: It's important that `finalizationPeriodSeconds` is not reduced on upgrade until the contract has waited at least 7-days for finalization of all previously proposed state roots submitted through the old contract. Consider only configuring this state variable when `l2Outputs.length == 0`.

Succint: This concern is already outlined in the protocol's documentation.

Cantina Managed: Acknowledged.

3.2.5 Instant finalization is potentially dangerous in managing chain re-orgs

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: After migrating the `L2OutputOracle` contract to the new `OPSuccinctL2OutputOracle` implementation, the state is reinitialized with updated parameters and new state variables. `finalizationPeriodSeconds` is maintained for the duration of a single `finalizationPeriodSeconds` as all existing `L2Ouputs` must be optimistically proven as per the original implementation until an OP-Stack chain can make use of the `SP1Verifier` to prove state. However, in the event of a chain re-org on L2, a potentially invalid state has now been verified on L1 and users may have already proven/executed a withdrawal transaction.

Recommendation: `finalizationPeriodSeconds` must be configured such that each newly proposed L2 output has reached a degree of finality allowing for withdrawal execution on the `OptimismPortal` contract. This can follow ETH2's designated finality of *two* epochs (~13 minutes).

This should give sufficient time for the `challenger` to delete L2 outputs that have been affected by the re-org but not yet finalized.

Succinct: Fixed in PR 266 by setting default `finalizationPeriod` of 1 hour in the documentation.

Cantina Managed: Verified fix.