Optimism Security Audit Report

By Kyle Charbonnet - Contract Worker for the Ethereum Foundation

June 28, 2021

Overview

Executive Summary

The Ethereum Foundation has requested a security audit of Optimism's optimistic rollup implementation. The audit focused on the smart contracts, L2Geth, message-relayer service and batch-submitter service. The audit found 1 major level issue:

1. Insufficient validations ensuring transaction monotonicity on layer 2.

This issue will be explained in much more detail later in the report. The overall code quality was great with best practices being followed. A standout was the exceptional use of inline comments for explaining complex or subtle parts of code.

Background

Optimism has created an implementation of the optimistic rollup - a layer 2 scaling solution. It's implementation revolves around making transactions replayable and deterministic on both layer 1 and layer 2. Optimism achieved this by making slight modifications to the EVM, thus creating the OVM - Optimistic Virtual Machine. They made slight modifications to Geth to create L2Geth, which implements the OVM.

The main difference between the EVM and OVM is that the OVM has removed/modified some context dependent opcodes. This is to ensure that each transaction's outcome does not rely on information that differs between layer 1 and layer 2. For example, the TIMESTAMP opcode has been modified to use the initial layer 2 transaction timestamp. This allows the transaction to be replayed on layer 1 at a later time, but still use the initial layer 2 timestamp.

All transactions on layer 2 are executed in the OVM rather than the EVM. So, when transactions need to be replayed on layer 1, there are a set of smart contracts that facilitate the execution through the OVM. The only time a transaction will need to be replayed on layer 1 is during a fraud proof.

Optimism currently uses a 7 day fraud proof window. So, a fraud proof may be initialized and completed within 7 days of the relevant transaction. If completed successfully, the resultant state roots from that transaction onward are erased and the sequencer will have their bond slashed. The sequencer will then recompute all of those state roots. It is important to note that at launch, the Optimism team is using a centralized sequencer owned by them to ensure a stable release. However, there are plans to decentralize this role in the near future.

System Components

The system's main components are the smart contracts, L2Geth, and the services. There are multiple smart contracts: some deployed to layer 1, some deployed to layer 2, and others deployed to both. These smart contracts are responsible for storing transaction and state data on layer 1, facilitating fraud proofs, implementing opcode changes for the OVM, and allowing cross-chain transactions.

L2Geth is run by two types of nodes - sequencers and verifiers. Users send the sequencer node transactions, and since L2Geth runs the OVM, the sequencer is able to execute these transactions and update the layer 2 state. The sequencer then publishes the transaction data and state data to layer 1. The verifier nodes can follow either layer 1 or layer 2 for any new layer 2 transaction data that the sequencer has published or processed. They will then execute those transactions themselves to ensure that the state root posted by the sequencer is correct. If it is not, they will begin a fraud proof.

Optimism uses multiple TypeScript services to aid the sequencer, enact the relayer, and support cross chain data updates. The batch-submitter services are only run by the sequencer and they are responsible for posting the transaction and state root data to layer 1. The message-relayer service is run by anyone who wants to relay messages from layer 2 to layer 1. Lastly, the data-transport-layer indexes layer 2 transaction data that is on layer 1. Verifiers will use this service to get updates on new transactions that they must process themselves.

Coverage

Resources Examined

Optimism has a monorepo on github that contains the smart contracts, L2Geth, and services. This repo was the codebase under review:

- https://github.com/ethereum-optimism/optimism

Unlike common practice, I periodically pulled in the latest changes and simply looked at the develop (main) branch. This review ended by commit:

- 31f517a2a2fb5e36bb9521d3837556d5cafeaff5

The important documentation used to understand the system were:

- Optimism Documentation: https://community.optimism.io/docs/#for-casually-interested-readers
- Paradigm's Optimism Explainer: https://research.paradigm.xyz/optimism

Areas of Concern

Optimistic Rollup Concerns

The optimistic rollup has a certain set of security concerns that all implementations will likely face. The following areas were specifically targeted during this audit due to the nature of the optimistic rollup protocol:

- The security and reliability of the process of identifying a fraudulent state
- Security of transferring tokens and ETH between layer 1 and layer 2
- Sufficient data is posted to layer 1 such that the system doesn't enter an unknown state
- Protection against layer 2 transaction censorship

Optimism Strengths

It is helpful to understand Optimism's security strengths before looking at the areas of concern for Optimism's implementation of the optimistic rollup.

The most significant security strength Optimism has is the reuse of Geth. Geth is the dominant Ethereum protocol implementation and has been operating since 2015. These years of experience have given Geth time for many bugs to surface and be patched. Time in production use is one of the best ways to ensure that software is secure. Therefore, Optimism's reuse of Geth and the EVM greatly increases the percentage of well tested and secure code. Of course it is still important to audit the entire codebase since underlying assumptions used to build Geth may no longer apply to L2Geth.

Another strength of Optimism is the replayability of layer 2 transactions on layer 1. This feature stems directly from the reuse of L2Geth and the EVM. Since transactions are replayable on both layer 1 and layer 2, the fraud proof is able to prove to the Ethereum network that a state root posted by the sequencer is fraudulent. This prevents the need for an entirely new consensus mechanism to determine when a state root is fraudulent. So, the primary focus for security here is ensuring that the OVM does truly support deterministic transactions on both layer 1 and layer 2.

Optimism-Specific Concerns

Along with the optimistic rollup concerns, the following list of areas (not exhaustive) were investigated during this audit:

- No differences between execution of layer 2 transactions on layer 1 versus layer 2. Specific areas:
    - Gas charges/refunds
    - Pre-transaction state data - transactions on layer 2 use the L2Geth database for state data, but on layer 1 they use a smart contract
    - Reverts
    - OVM specific opcodes that differ from their EVM counterparts

- Proper bridge contract security for preventing fake withdrawls of ETH/tokens onto layer 1 or layer 2
- Sequencers are careful not to break rules that will slow or freeze its process of adding new transactions to layer 1 - ex. Skipping enqueued transactions.
- Verifier careful not to skip any transaction
- Standard security best practices for the smart contracts involved
- Fraud Proof security against attackers once it has been created

Vulnerabilities

Vulnerability 1: Insufficient Validations for Transaction Monotonicity

Location
https://github.com/ethereum-optimism/optimism/blob/31f517a2a2fb5e36bb9521d3837556d5cafeaff5/packages/contracts/contracts/optimistic-ethereum/OVM/chain/OVM_CanonicalTransactionChain.sol#L459-L519

Background
All layer 2 transactions have their data stored on layer 1 through the CanonicalTransactionChain (CTC) smart contract. These transactions are applied to the layer 2 state in order, so it is expected that the timestamps and block numbers of these transactions are in a non-decreasing order.

The sequencer must apply 2 types of transactions to the layer 2 state: transactions in the CTC queue and transactions sent directly to the sequencer. These 2 types of transactions have different rules regarding their timestamps and block numbers.

When the sequencer calls appendSequencerBatch(), it provides a list of BatchContexts that define the number of sequencer transactions to add followed by the number of queue transactions to add directly after the sequenced transactions. The timestamp and block number for these sequencer transactions are determined by the timestamp and block number of the BatchContexts. The sequencer is free to choose these timestamps within certain limits, such as ensuring that these sequencer transactions have non-decreasing timestamps/block numbers. However, the timestamp and block number for the queue transactions are set to the timestamp and block number at the time those transactions were added to the queue via enqueue().

Description
It is currently possible that transactions can be appended to the CanonicalTransactionChain contract with unordered timestamps and/or block numbers. In other words, there is a way to violate monotonicity and to break the non-decreasing order of transaction timestamps/block numbers.

The CTC contract performs various validations to ensure that transactions are not appended out of order, such as comparing each BatchContext timestamp with the timestamp of the previous BatchContext. However, one comparison that is missed is between the timestamp of queued transactions and the BatchContext that follows them. Since queue transactions don't share the same timestamp as the BatchContext before or after them, and there is no comparison here, transactions can be appended out of order.

Therefore, the sequencer could maliciously or accidently append transactions with unordered timestamps/block numbers. This has the potential to break any smart contracts that are time sensitive and rely on guaranteed non-decreasing transaction timestamps/block numbers.

Suggested Solution
A simple solution is to add one validation to the CTC appendSequencerBatch() function that ensures the timestamp/block number of the next BatchContext is greater than the timestamp/block number of the previous queue transaction (if there was at least one).

This vulnerability won't be exploitable at launch since the sequencer will be centralized by the Optimism team to start. Additionally, the sequencer software that Optimism uses does have this necessary validation in place to ensure unordered transactions don't occur. However, it is still important to add it to the smart contract so no accidents can occur.

Status
In Progress - Internal issue created

Final Remarks

Overall, the reuse of Geth and the EVM are Optimism's strong points from a security perspective. While it introduces some complexities, reusing these software and having a sandboxed environment to execute transactions on both layer 1 and layer 2 makes the system easier to understand. This helps immensely when auditing since it is easier to understand what the system should be doing and this should not be undervalued.

An important 'todo' is Optimism's test net inclusion of gas fees. The Optimism team has had their Kovan testnet open for a while now, and are now beginning to require gas fees. This testing will provide a lot more confidence in the system as the gas fee system on layer 2 can be somewhat complex. This would greatly solidify the test coverage of the Optimism optimistic rollup.