Pepe Unchained Security Review

Pashov Audit Group

Conducted by: 0xdeadbeef, Shaka, ast3ros

November 6th - November 10th

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **celestiaorg/optimism** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Pepe Unchained

Pepe Unchained Blockchain forks Optimism, an Ethereum layer-2 blockchain that enhances scalability and reduces transaction costs, making decentralized applications faster and more affordable. Paired with Celestia's modular data availability network, it empowers developers to launch scalable, customizable blockchains efficiently.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - 8c535c615fea063b85257335128e97b73589b9fa

Scope

The following repositories were in scope of the audit:

- https://github.com/celestiaorg/optimism/releases/tag/v1.3.0-OP_v1.9.2-CN_v0.15.0
- https://github.com/Uniswap/v3-core/tree/d8b1c635c275d2a9450bd6a78f3fa2484fef73eb
- https://github.com/Uniswap/v3-periphery/tree/0682387198a24c7cd63566a2c58398533860a5d1

7. Executive Summary

Over the course of the security review, 0xdeadbeef, Shaka, ast3ros engaged with Pepe Unchained to review Pepe Unchained. In this period of time a total of **2** issues were uncovered.

Protocol Summary

Protocol Name	Pepe Unchained
Repository	https://github.com/celestiaorg/optimism
Date	November 6th - November 10th
Protocol Type	Layer 2

Findings Count

Severity	**Amount**
Low	2
Total Findings	**2**

Summary of Findings

ID	Title	Severity	Status
[L-01]	Missing validation for daConfig when creating new op-node	Low	Acknowledged
[L-02]	Redundancy when AltDA is enabled in op-batcher driver.sendTransaction	Low	Acknowledged

8. Findings

8.1. Low Findings

[L-01] Missing validation for daConfig when creating new op-node

When creating a new op-node, all the configuration is validated except for the `daConfig`.

```
// Check verifies that the given configuration makes sense
func (cfg *Config) Check() error {
        ...
        if err := cfg.AltDA.Check(); err != nil {
                return fmt.Errorf("altDA config error: %w", err)
        }
        if cfg.AltDA.Enabled {
                log.Warn(
    "Alt-DAModeisaBetafeatureoftheMITlicensedOPStack.Whileithasreceivedinitialreviewfr
    itisstillundergoingtesting,
    andmayhavebugsorotherissues."
  )
        }
        if err := cfg.DaConfig.Check(); err != nil { // @audit no validate config
                return fmt.Errorf("da config error: %w", err)
        }
        return nil
}
```

The `Check()` function for daConfig is empty:

```
func (c CLIConfig) Check() error {
        return nil
}
```

It's recommended to validate the config such as `RpcURL` or `FallbackMode` in the `daConfig` struct.

[L-02] Redundancy when AltDA is enabled in op-batcher driver.sendTransaction

When AltDA is enabled in `op-batcher/batcher/driver.go`, the current flow of sendTransaction is as follows: the original transaction data is sent to the AltDA server, which returns a commitment. This commitment is then sent to Celestia.

```
if l.Config.UseAltDA {
                        comm, err := l.AltDA.SetInput(ctx, data)
                        if err != nil {
                                l.Log.Error("Failed to post input to Alt DA", "error",
                                // requeue frame if we fail to post to the DA Provider
                                l.recordFailedTx(txdata.ID(), err)
                                return nil
                        }
                        l.Log.Info("Set AltDA input", "commitment", comm, "tx", txdata
                        // signal AltDA commitment tx with TxDataVersion1
                        data = comm.TxData()
                }
                candidate, err = l.celestiaTxCandidate(data)
                if err != nil {
                        l.Log.Error("celestia: blob submission failed", "err", err)
                        candidate, err = l.fallbackTxCandidate(ctx, txdata)
                        if err != nil {
                                l.Log.Error("celestia: fallback failed", "err", err)
                                l.recordFailedTx(txdata.ID(), err)
                                return nil
                        }
                }
```

We have two scenarios:

`Double commitment storage case`: When AltDA is used and the AltDA commitment submission to Celestia succeeds, the commitment (not the transaction data) is stored in Celestia. This creates redundancy as the commitment is stored in both AltDA and Celestia. So when the roll-up node tries to reconstruct the L2 blocks from the L1 data, because the first byte is `0xce`, it would retrieve the data from Celestia (not AltDA). However, this data is the AltDA commitment rather than the actual transaction data. link

`AltDA Bypass case`: If the Celestia submission fails while attempting to store the AltDA commitment, then L2 transaction data is sent to L1 instead of AltDa commitment even though the AltDa works correctly. This means AltDa is completely bypassed.

It's recommended to store only in AltDA when it's enabled, without redundant Celestia storage.