Optimism
Security Review

Cantina Managed review by:

MiloTruck, Lead Security Researcher
Zigtur, Security Researcher

March 9, 2025

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

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 Feb 27th to Mar 1st the Cantina team conducted a review of optimism on commit hash 8d0dd96e. The team identified a total of **4** issues:

Issues Found

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

3 Findings

3.1 Medium Risk

3.1.1 Increase in calldata cost due to EIP-7623 affects L1 <> L2 transactions

Severity: Medium Risk

Context: OptimismPortal2.sol#L257-L259, CrossDomainMessenger.sol#L343-L360

Description: As part of the Pectra upgrade, EIP-7623 increases the cost of calldata for transactions. More specifically, the total gas used per transaction is now the maximum between the execution gas and the following:

```
10 * (zero_bytes_in_calldata + nonzero_bytes_in_calldata * 4)
```

However, Optimism's L1 and L2 contracts have not been updated to account for this change. Calldata cost is used in two contracts:

1. In `CrossDomainMessenger`, `baseGas()` calculates the amount of gas needed for `relayMessage()` to complete execution on the destination chain. This ensures messages sent through `CrossDomainMessenger` can be replayed and will never be stuck:

   ```
   // Calldata overhead
   + (uint64(_message.length) * MIN_GAS_CALLDATA_OVERHEAD)
   ```

2. In `OptimismPortal2`, `minimumGasLimit()` calculates the minimum gas limit for an L1 -> L2 transaction. This ensures the gas cost of deposit transactions scale with calldata size:

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

(1) does not affect L2 $\rightarrow$ L1 transactions as:

- EIP-7623 does not impact the inner workings of the EVM. It only impacts the whole transaction gas cost, and not the inner VM gas cost (named `execution_gas_used` in the EIP).
- For L2 $\rightarrow$ L1 transactions, `relayMessage()` is a sub-call from `OptimismPortal2`, so its calldata cost will remain at 4/16 gas per zero/non-zero calldata byte.

As such, the gas cost calculated with `baseGas()` will still be correct. However, L1 $\rightarrow$ L2 transactions are impacted by (1) as `relayMessage()` is called directly by the node. If a message's calldata is sufficiently long, `baseGas()` will return a gas limit lower than the floor cost stated in EIP-7623, causing the call to `relayMessage()` on L2 to fail due to insufficient gas.

(2) allows users to create deposit transactions with large calldata for less gas than what it actually costs on L2. This creates a risk of DOS as users can consume L2 resources for cheaper than intended.

Recommendation: The Optimism team has come up with the following fix:

- For (1), `baseGas()` needs to be updated to match the new `max(...)` computation in EIP-7623.
- For (2), change the calldata cost per byte from 16 to 40.

Optimism: Fixed in PR 14608.

Cantina Managed: Verified, the calculation in `CrossDomainMessenger.baseGas()` now matches the new gas formula in EIP-7623. The calldata cost in `OptimismPortal2.minimumGasLimit()` has also been increased from 16 to 40.

3.2 Low Risk

3.2.1 ETH transfers with `RECEIVE_DEFAULT_GAS_LIMIT` for EOAs is no longer safe after EIP-7702

Severity: Low Risk

Context: L1StandardBridge.sol#L109-L112, L2StandardBridge.sol#L79-L84

Description: `L1StandardBridge` has a receive function which allows EOAs to transfer ETH to L2 conveniently:

```
/// @notice Allows EOAs to bridge ETH by sending directly to the bridge.
receive() external payable override onlyEOA {
    _initiateETHDeposit(msg.sender, msg.sender, RECEIVE_DEFAULT_GAS_LIMIT, bytes(""));
}
```

Similarly, the receive function in `L2StandardBridge` allows for ETH transfers back to L1:

```
/// @notice Allows EOAs to bridge ETH by sending directly to the bridge.
receive() external payable override onlyEOA {
    _initiateWithdrawal(
        Predeploys.LEGACY_ERC20_ETH, msg.sender, msg.sender, msg.value, RECEIVE_DEFAULT_GAS_LIMIT, bytes("")
    );
}
```

As seen from above, the minimum gas limit for the ETH transfer on the destination chain (i.e. `_minGasLimit`) is specified as `RECEIVE_DEFAULT_GAS_LIMIT`, which is 200,000 gas. However, with EIP-7702, this is no longer safe. Previously, EOAs were guaranteed to not have code, so 200k gas would definitely be sufficient. Now, if an EOA has code on the receiving chain, it is possible for the call to run out-of-gas when transferring ETH.

Recommendation: Consider marking the receive function in both contracts as legacy and leaving a warning explaining why they are not safe to use for EOAs with code.

Optimism: Acknowledged.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 `EOA.isSenderEOA()` does not check the caller's code size

Severity: Informational

Context: EOA.sol#L13-L19, tx_setcode.go#L36-L42

Description: In `EOA.isSenderEOA()`, if the caller is not `tx.origin`, it is determined to be an EOA by checking if the first three bytes of code at the caller's address is equal to `0xEF0100`:

```
// If the sender is not the origin, check for 7702 delegated EOAs.
assembly {
    let ptr := mload(0x40)
    mstore(0x40, add(ptr, 0x20))
    extcodecopy(caller(), ptr, 0, 0x20)
    isEOA_ := eq(shr(232, mload(ptr)), 0xEF0100)
}
```

However, the function does not check if the caller's code size is equal to 23. This is how execution clients determine if the code at an address should be delegated as per EIP-7702, such as in geth:

```
// ParseDelegation tries to parse the address from a delegation slice.
func ParseDelegation(b []byte) (common.Address, bool) {
    if len(b) != 23 || !bytes.HasPrefix(b, DelegationPrefix) {
        return common.Address{}, false
    }
    return common.BytesToAddress(b[len(DelegationPrefix):]), true
}
```

Recommendation: Check that the caller's code size is equal to 23 as such:

```
  assembly {
      let ptr := mload(0x40)
      mstore(0x40, add(ptr, 0x20))
      extcodecopy(caller(), ptr, 0, 0x20)
-     isEOA_ := eq(shr(232, mload(ptr)), 0xEF0100)
+     isEOA_ := and(
+         eq(shr(232, mload(ptr)), 0xEF0100),
+         eq(extcodesize(caller()), 23)
+     )
  }
```

Optimism: Fixed in PR 14608.

Cantina Managed: Verified, `isSenderEOA()` now checks if the caller's code size is 23.

3.3.2 Multiple zero code length checks are no longer safe

Severity: Informational

Context: OPContractsManager.sol#L1025-L1028, DeputyPauseModule.sol#L229-L233

Description: The `DeputyPauseModule` and `OPContractsManager` contracts implement zero code length checks on addresses that will be used for future delegate calls. Due to EIP-7702, these checks are no longer safe. They allow an EOA to be set as the delegatee, providing this EOA the capability to modify the implementation code through EIP-7702 delegations.

Recommendation: The zero code length checks should be modified into EOA checks. This can be done by checking that the bytecode is not of length 23 and doesn't start with the magic `0xef0100`.

Optimism: Acknowledged.

Cantina Managed: Acknowledged. The entities allowed to set the implementations are trusted.