Coinbase Verifying Paymaster
Security Review

Cantina Managed review by:
Riley Holterhus, Lead Security Researcher
Rustyrabbit, Security Researcher

September 9, 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

Coinbase's Verifying Paymaster is an ERC-4337 compliant paymaster, designed for use within the Coinbase Developer Platform. It uses signature-based validation and supports ERC20 token fee payments.

From Sep 4th to Sep 6th the Cantina team conducted a review of verifying-paymaster on commit hash d28a4064. The team identified a total of **5** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 2
- Gas Optimizations: 0
- Informational: 3

3 Findings

3.1 Low Risk

3.1.1 `_calculateTokenCost()` rounding

Severity: Low Risk

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

Description: This issue was raised by the Coinbase team and has been included for tracking purposes.

The `_calculateTokenCost()` function converts an ETH amount into an equivalent ERC20 token amount using a provided `tokenExchangeRate`. This function is implemented as follows:

```
function _calculateTokenCost(uint256 gasCost, uint256 tokenExchangeRate) internal pure returns (uint256) {
    return (gasCost * tokenExchangeRate) / 1e18;
}
```

Notice that this implementation rounds down, which favors the user by reducing the amount of ERC20 tokens they are charged. In situations where the token has a small `decimals()` value, the rounding error in dollar terms may be noticeable over time. Additionally, there are cases where rounding down results in zero token cost to the user. To avoid these issues, it may be preferable to always round up, which would be in favor of the paymaster.

Recommendation: Consider adjusting the `_calculateTokenCost()` function to round up rather than down. This can be achieved using Solady's `FixedPointMathLib.mulDivUp()` function or OpenZeppelin's `Math.mulDiv()` function that takes a rounding direction as input.

Coinbase: Addressed in PR 5.

Cantina Managed: Verified.

3.1.2 `_validatePaymasterUserOp()` uses ERC-4337 banned opcode

Severity: Low Risk

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

Description: To mitigate mempool DoS vulnerabilities, ERC-4337 forbids the use of certain opcodes during the validation step of a user operation. One such forbidden opcode is the `BASEFEE` opcode, which is the underlying opcode for Solidity's `block.basefee` global variable.

Currently, the `_validatePaymasterUserOp()` function in the `VerifyingPaymaster` contains one location where this forbidden opcode is used:

```
function _validatePaymasterUserOp(/* ... */) /* ... */ {
    // ...
    if (paymasterData.token != address(0)) {
        uint256 gasPrice = _min(userOp.maxFeePerGas, userOp.maxPriorityFeePerGas + block.basefee);
        postOpContext.postOpOverheadFee = (paymasterData.postOpGasCost * gasPrice);
        // ...
        }
    }
    // ...
}
```

Since this violates the ERC-4337 specification, there can be issues when user operations interacting with the `VerifyingPaymaster` are submitted to the general 4337 mempool.

Recommendation: Consider removing the use of `block.basefee`. Note that the code referenced above is intended to account for the gas costs associated with the `_postOp()` call. One potential solution to avoid using `block.basefee` is to set `userOp.maxFeePerGas` as an initial upper bound on the gas price, and then any overcharges can be reimbursed in the `_postOp()` function (which is a function that's allowed to access the `BASEFEE` opcode).

Coinbase: Addressed in PR 4.

Cantina Managed: Verified.

3.2 Informational

3.2.1 Code style changes

Severity: Informational

Context: Global Scope

Description: The team has implemented code styling changes after the audit, including cleaning up imports and reordering functions. This finding is included here for tracking purposes.

Recommendation: Implement the desired style changes.

Coinbase: Style changes applied in PR 3.

Cantina Managed: Verified.

3.2.2 `prepaymentRequired` considerations

Severity: Informational

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

Description: In the `VerifyingPaymaster`, the `paymasterData.prepaymentRequired` boolean controls when ERC20 transfers occur. If `paymasterData.prepaymentRequired == true`, an upper bound amount of ERC20 is transferred from the userOp sender during the verification step, with any overcharges refunded in the `postOp()` function. If `paymasterData.prepaymentRequired == false`, the ERC20 transfer is handled entirely within the `postOp()` function.

In an internal audit report provided before this audit, the Coinbase team identified a potential issue in scenarios where the ERC20 transfer occurs entirely in the `postOp()`. Specifically, if the userOp execution interferes with the paymaster's ability to transfer tokens (e.g. by depleting the sender's token balance or revoking approvals), the paymaster will be unable to reimburse itself in `postOp()` while still being responsible for the gas costs.

This issue has been mitigated by ensuring that userOps are fully simulated and that senders who repeatedly cause this problem are blocked, along with using the `paymasterData.prepaymentRequired == true` logic in scenarios where prepayment is possible. While the team is already aware of this issue, the following informational observations may be useful:

- If the paymaster reverts during the first `postOp()` call, it will also cause the `userOp` execution to revert. This might be considered useful in cases where a failed `transferFrom()` in the `postOp()` is due to malicious behavior from the user. By reverting, the execution of their `userOp` is blocked and they will not benefit from the free gas sponsorship.
- In the `paymasterData.prepaymentRequired == true` case, if a `userOp` intends to spend the contract's entire ERC20 balance, it might revert because the verification step deducts tokens before execution. This is similar to how EOAs can't spend their full ETH balance without reserving an amount for gas fees. Fortunately, any issues resulting from this behavior would likely be caught in off-chain simulation, and the `userOp` execution could be adjusted as needed.

No recommendation. This finding has been provided for informational purposes.

3.2.3 ERC-4337 v0.6 `postOp()` revert bug considerations

Severity: Informational

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

Description: The `VerifyingPaymaster` contract is designed to work with v0.6 of the ERC-4337 specification, which contains a known bug. This bug is within the `EntryPoint` contract, and is triggered when the first `postOp()` call reverts with less than 32 bytes of data. In this scenario, the entire transaction reverts instead of the intended behavior of gracefully catching the `postOp()` revert and continuing with a second `postOp()` call.

In the implementation of `postOp()` found in the `VerifyingPaymaster`, the following error would produce a 4-byte selector, which means it would trigger the bug in the `EntryPoint` contract:

```
if (!c.allowAnyBundler && !isBundlerAllowed[tx.origin]) {
    revert BundlerNotAllowed();
}
```

Additionally, note that the following external calls exist in the `postOp()` implementation, and these calls may also revert with data less than 32 bytes in length:

```
if (c.prepaidAmount == 0) {
    IERC20(c.token).safeTransferFrom(c.sender, c.receiver, actualTokenCost);
} else {
    IERC20(c.token).safeTransfer(c.sender, c.prepaidAmount - actualTokenCost);
    IERC20(c.token).safeTransfer(c.receiver, actualTokenCost);
}
```

As a result, these scenarios can trigger the bug in `EntryPoint` and cause the entire transaction to fail. This is most relevant for the bundler to consider, as the paymaster would not be charged for the revert, and the specification doesn't indicate that this should affect the paymaster's reputation.

Recommendation: Since preventing the `EntryPoint` bug from being triggered entirely may not be practical, consider documenting this behavior in the codebase to ensure it's clearly understood, especially for bundlers interacting with the `VerifyingPaymaster`.

Coinbase: Addressed in PR 6.

Cantina Managed: Verified. The Coinbase team has decided to mitigate the potential reverts that are less than 32 bytes in length. This has been accomplished by changing the `BundlerNotAllowed()` error to include the address of the offending bundler, and by switching from OpenZeppelin's `SafeERC20` to Solmate's `SafeTransferLib`, which uses revert error messages larger than 32 bytes.