Zyfi Paymasters
Security Review

Cantina Managed review by:
Riley Holterhus, Lead Security Researcher
Chris Smith, Security Researcher

February 29, 2024

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Medium Risk . . . 4
  - 3.1.1 `verifier` signatures can be replayed . . . 4
- 3.2 Low Risk . . . 5
  - 3.2.1 No way to reset a protocol to use the `defaultMarkup` . . . 5
  - 3.2.2 `withdrawETH` can be blocked . . . 5
  - 3.2.3 `renounceOwnership` implementation can block `onlyOwner` functions . . . 5
- 3.3 Gas Optimization . . . 6
  - 3.3.1 Gas savings . . . 6
- 3.4 Informational . . . 7
  - 3.4.1 Trust assumption considerations . . . 7
  - 3.4.2 Refunds can be inaccurate . . . 8
  - 3.4.3 Sponsorship refunds may fail . . . 8
  - 3.4.4 No events are emitted in `setMarkup()` . . . 9
  - 3.4.5 `setVerifier` lacks `address(0)`-check . . . 9

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

Zyfi is a set of tools that leverages zkSync's native account abstraction to provide a seamless on-chain experience to users. Zyfi offers a full-stack solution ranging from a front-end to an API that lets any dApp propose gasless transactions, enabling users to pay with any ERC-20 tokens. Protocols can also decide to sponsor part or all of a user transaction according to their custom off-chain logic.

From Feb 12th to Feb 13th the Cantina team conducted a review of zyfi-paymaster on commit hash e38bf16a. The team identified a total of **10** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 1
- Low Risk: 3
- Gas Optimizations: 1
- Informational: 5

3 Findings

3.1 Medium Risk

3.1.1 `verifier` signatures can be replayed

Severity: Medium Risk

Context: ERC20Paymaster.sol#L198-L208, ERC20SponsorPaymaster.sol#L268-L280

Description: In the Zyfi Paymaster flow, users pay Zyfi with their ERC20 tokens, and Zyfi pays for the users' transaction costs in return. To ensure that Zyfi actually agrees to the terms of this exchange, a signature from Zyfi's `verifier` account is validated in the `validateAndPayForPaymasterTransaction()` function:

```
if (
    !_isValidSignature(
        signedMessage,
        address(uint160(_transaction.from)),
        address(uint160(_transaction.to)),
        token,
        amount,
        expirationTime,
        _transaction.maxFeePerGas,
        _transaction.gasLimit
    )
) {
    magic = bytes4(0);
}
```

Notice that this code snippet only verifies the `from`, `to`, `maxFeePerGas` and `gasLimit` parameters of the `_transaction` object. This implies that a signature from the `verifier` can be replayed across several transactions, so long as each transaction involves the same user calling the same address with the same gas values.

This is undesirable, as it gives the Zyfi API less control over the spending of its paymasters. This is especially important in the `ERC20SponsorPaymaster`, because protocol sponsors are entrusting the `verifier` to manage their ETH spending on-chain, which is more difficult with potential signature replays.

Recommendation: Introduce a nonce system in each paymaster. This nonce would be provided in the transaction's `paymasterInput`, and each nonce would be invalidated by the paymaster after its first use. If this nonce is added to the hash computed by `_isValidSignature()`, all possibility of signature replay is prevented.

If a new nonce system increases transactions fees too much, consider incorporating the transaction's nonce value instead. For example, the `_transaction.nonce` value can be added to hash computed by `_isValidSignature()`, and zkSync's own nonce tracking system will prevent signature replays.

Zyfi: Fixed in PR 8. We added a `maxNonce` check in `ERC20SponsorPaymaster`. We favoured this approach as it saves us from a storage read and write and gives us flexibility to adapt the "strictness" of the check from the API side. For `ERC20Paymaster` we omitted the check as we consider any replayed transaction to be a fair exchange between the paymaster ETH and the user token.

Cantina Managed: Verified.

With the `ERC20Paymaster`, signature replay concerns have been acknowledged. It's encouraged that the Zyfi API only signs `expirationTime` values that are not far in the future. This will help maintain a fair exchange rate of ETH and ERC20 tokens on replayed signatures.

With the `ERC20SponsorPaymaster`, a `maxNonce` check has been added. It's encouraged that the Zyfi API signs these values such that very few signature replays can happen. To facilitate this, the API should be aware of the specifics of zkSync's nonce system (where nonces aren't guaranteed to be monotonically increasing).

3.2 Low Risk

3.2.1 No way to reset a protocol to use the `defaultMarkup`

Severity: Low Risk

Context: ERC20SponsorPaymaster.sol#L302-L304, ERC20SponsorPaymaster.sol#L391

Description: Once a protocol has a markup set, it is impossible to allow that protocol to use the default markup again due to the min of `50_00` in `setMarkup`. This would then require switching the protocol's address or manually resetting all "default" protocols everytime `setDefaultMarkup` is called.

Recommendation: If it is desirable to be able to set a protocol's markup and then return it to using the default, allow this case in `setMarkup`. For example:

```
function setMarkup(address _address, uint256 _newMarkup) public onlyOwner {
    if (_newMarkup != 0 && _newMarkup < 50_00 || _newMarkup > 200_00)
        revert Errors.InvalidMarkup();
    markups[_address] = _newMarkup;
}
```

Zyfi: Fixed in PR 5 and commit e4014dd7.

Cantina Managed: Verified.

3.2.2 `withdrawETH` can be blocked

Severity: Low Risk

Context: ERC20Paymaster.sol#L244-L248, ERC20SponsorPaymaster.sol#L339-L343

Description: Users submit their transactions to the sequencer either directly or through a forcing mechanism on L1. The sequencer then bundles those into blocks. Either through luck or in a worst case scenario through collusion with the sequencer a user could continue to submit transactions and front run calls to `withdrawETH` causing attempts to withdraw all the ETH in the contract to fail (since `validateAndPayForPaymasterTransaction` decrements the ETH in the contract) calls to it just before a call to `withdrawETH(toAddress, <Total-ETH>)` would cause the withdraw to revert.

Recommendation: Add a `withdrawAllETH()` function the owner can call that will send the full balance of ETH in the contract. This is similar to what was implemented in the example paymaster.

Zyfi: We plan to migrate to new versions of Paymasters instead of using upgrades. Part of this migration plan will be that the API stops serving transactions to an obsolete paymaster for the time before draining it which should eliminate the issue you described. The "precise" version was added to allow for possible rebalancing between paymasters, without interrupting service. However, we are okay adding `withdrawAllETH()` as a protection from this edge case. Fixed in PR 3.

Cantina Managed: Verified.

3.2.3 `renounceOwnership` implementation can block `onlyOwner` functions

Severity: Low Risk

Context: ERC20Paymaster.sol#L28, ERC20SponsorPaymaster.sol#L28

Description: OpenZeppelin's Ownable contract implements a `renounceOwnership` function that allows the owner to set `address(0)` as the new owner. In addition to blocking any future assignment of ownership. This would block the following functions:

- ERC20Paymaster.sol:
  - `setVerifier`
  - `withdrawETH`
  - `withdrawERC20`
  - `withdrawERC20Batch`
- ERC20SponsorPaymaster.sol:

- `setVerifier`
- `setVault`
- `withdrawETH`
- `withdrawERC20`
- `withdrawERC20Batch`
- `setMarkup`
- `setDefaultMarkup`

Recommendation: Since even when shutting down the contract, locking the deposited ETH and ERC20 payments is unwanted behavior that could be accidentally triggered, we would recommend overwriting `renounceOwnership`. At a minimum, it probably makes sense that it would withdraw the ETH to the current owner, but you may want to consider the best way to handle the ERC20 token balances too. This may simply be documenting the steps to occur before `renounceOwnership` is called.

Zyfi: We don't intend to call `renounceOwnership` on a working paymaster, and we can maintain ownership on a deprecated one. We overrode the function in PR 2.

Cantina Managed: Verified.

3.3 Gas Optimization

3.3.1 Gas savings

Severity: Gas Optimization

Context: ERC20Paymaster.sol#L50-L56, ERC20SponsorPaymaster.sol#L67-L73, SponsorshipVault.sol#L32-L35, SponsorshipVault.sol, ERC20Paymaster.sol#L41, ERC20SponsorPaymaster.sol#L38-L41

Description:

- **Modifiers:**
  - ERC20Paymaster.sol#L50-L56
  - ERC20SponsorPaymaster.sol#L67-L73
  - SponsorshipVault.sol#L32-L35

  Generally in Ethereum, there are some gas savings by moving modifier code from the modifier into a function. Since when a contract is deployed, the modifier code is copied into each function that uses it, this reduces the size of the code.

- **Safe/Unchecked Math**:
  - SponsorshipVault.sol

  In the `SponsorshipVault` all of the math operations (adding and subtracting from sponsor balances) should be able to be wrapped in `unchecked` blocks. Where the user is depositing or the Paymaster is refunding, that addition cannot approach `type(uint256).max` since it is dealing with ETH and `msg.value`. The subtraction functions will already revert with `Errors.NotEnoughFunds()` if the amount to be subtracted is greater than the balance of the `msg.sender`.

- **Constants:**
  - ERC20Paymaster.sol#L41
  - ERC20SponsorPaymaster.sol#L38-L41

  The zkSync Era's documentation on design recommendations specifically calls out a scenario where it is better to reuse contract code and provides the example that constants are the better thing to do with Ethereum, but passing constructor parameters instead in Era "*leads to substantial fee savings*".

Recommendation: We recommend you test the effects of these changes in the zkSync environment to ensure they provide gas savings in Era.

- Example code for modifiers:

```
modifier onlyBootloader() {
    _checkBootloader();
    // Continue execution if called from the bootloader.
    _;
}

function _checkBootloader() internal view {
    if (msg.sender != BOOTLOADER_FORMAL_ADDRESS) {
        revert Errors.NotFromBootloader();
    }
}
```

- Example for the `unchecked`:

```
function withdraw(uint256 amount) public {
    if (amount > balances[msg.sender]) revert Errors.NotEnoughFunds();
        unchecked {
            balances[msg.sender] -= amount;
        }
    (bool sent, ) = payable(msg.sender).call{value: amount}("");
    if (!sent) revert Errors.FailedWithdrawal();
    emit Withdrawn(msg.sender, amount);
}
```

- For the Constants: The added complexity of having to pass the `NOMINATOR`s and `version` seems a little awkward; however, we would recommend at least trying the change in your environment and evaluating if the gas savings warrants making the change (and whether the savings is only on contract deploy or also in transactions since the latter is much more important for this use case).

Zyfi: We didn't notice gas savings when changing the modifiers or the constant deployment, but we will keep them in mind for future iterations.

We implemented the unchecked sections in PR 6.

Cantina Managed: Acknowledged and Verified.

3.4 Informational

3.4.1 Trust assumption considerations

Severity: Informational

Context: ERC20SponsorPaymaster.sol

Description: In the Zyfi Paymaster flow, each transaction involves on-chain validation of a user signature (included in `_transaction.signature`) and a Zyfi `verifier` signature (included in `_transaction.paymasterInput`). This implies the exchange of funds is trustless for both the user and Zyfi.

In the `ERC20SponsorPaymaster` contract, there also exists the protocol sponsor. The protocol sponsor does not have a signature validated on-chain, and they instead rely on Zyfi's signature to appropriately use their funds.

If the Zyfi `verifier` were to become compromised, the sponsor could have its entire `SponsorshipVault` balance used in an undesired way. Indeed, by creating several wasteful transactions and forcing the sponsor to pay for them using a `sponsorshipRatio` of 100%, the `verifier` could quickly drain the sponsor's ETH. In this scenario, a portion of the wasted ETH would be returned to the paymaster in each transaction's operator refund, in which case the paymaster `owner` could rescue a portion of the funds. However, the recoverable ETH would likely be only a fraction of the sponsor's initial balance.

A scenario where the paymaster `owner` becomes compromised is slightly worse, since they can change the `verifier` to do the same griefing mentioned above, and would not be likely to return any of the refunds.

Recommendation: If it is important to validate the protocol sponsor's intentions on-chain, consider incorporating a separate protocol signature in the `ERC20SponsorPaymaster` validation. If this is undesirable for UX or gas-efficiency reasons, consider encouraging protocols to not store a significant amount of funds in the `SponsorshipVault`. Instead, protocols can reduce their trust requirements by keeping a smaller amount in the vault, and periodically topping up their balance as needed.

Zyfi: We are glad that our paymasters are confirmed to be trustless for users, as it's a cornerstone of our design. Based on current discussions, the trust assumptions for protocols are acceptable given the gas savings and simplicity they allow, but we will keep in mind the trustless alternatives for the future.

Cantina Managed: Acknowledged.

3.4.2 Refunds can be inaccurate

Severity: Informational

Context: ERC20Paymaster.sol#L158, ERC20SponsorPaymaster.sol#L208

Description: In zkSync, transaction refunds are based on two components:

1. The amount of gas that's still unused after the transaction execution completes.
2. The amount of ETH that's refunded from the zkSync operator, partially due to L1 gas prices affecting the correctness of the L2 `basefee` and `gasPricePerPubdata` calculations.

In the current zkSync transaction flow, the bootloader is only aware of the first component when it calls `postTransaction()` on the paymaster. This means that the `_maxRefundedGas` parameter can underestimate the amount of ETH being returned to the paymaster, and users may receive a smaller refund than they would otherwise expect.

Recommendation: Unfortunately, there is no easy fix for this issue. The zkSync operator refund amount is not accessible to the paymaster during the `postTransaction()` call. The refund is also not guaranteed, so it's not advisable for the paymaster to attempt to estimate it.

So, it is recommended to simply make note of this behavior, and potentially document it in the code comments. If the `_maxRefundedGas` does not become more accurate in the future, it may also be worth exploring a periodic manual refund mechanism. This would require admin intervention and would be based on off-chain calculations that have the full context of each transaction refund.

Zyfi: Acknowledged. We will monitor any changes on the zkSync stack that can help us improve the refund logic.

Cantina Managed: Acknowledged.

3.4.3 Sponsorship refunds may fail

Severity: Informational

Context: ERC20SponsorPaymaster.sol#L221-L229

Description: In the native zkSync AA control flow, paymasters are refunded excess ETH left over from the transaction's execution. The paymaster is notified of this refund in the `postTransaction()` function, and the `ERC20SponsorPaymaster` uses this function to implement the following sponsorship refund:

```
if (requiredETHProtocol > 0) {
    uint256 refundEthProtocol = (requiredETHProtocol *
        _maxRefundedGas) / _transaction.gasLimit;
    ISponsorshipVault(vault).refundSponsorship{
        value: refundEthProtocol
    }(protocolAddress);
}
```

Since the `postTransaction()` call happens *before* any ETH is refunded to the paymaster, it's possible that this code reverts due to insufficient funds. This is complicated further by the fact that `_maxRefundedGas` value is an argument from the bootloader that may be larger than the actual refund later received.

Recommendation: This issue could be partially mitigated by implementing a `balances` mapping that can be incremented on refunds and later withdrawn from. However, this would introduce added complexity, and would still not fully address discrepancies due to `_maxRefundedGas` differing from the actual refund.

So, it is instead recommended to simply make note of this behavior, possibly by documenting it in the comments of `postTransaction()`. Since the paymaster already needs to maintain a buffer of ETH for regular usage, this behavior should rarely lead to a revert.

Zyfi: As discussed, this should not be an issue as Zyfi already needs to maintain an ETH buffer for regular use.

Cantina Managed: Acknowledged.

3.4.4 No events are emitted in `setMarkup()`

Severity: Informational

Context: ERC20SponsorPaymaster.sol#L389-L394

Description: The `setMarkup()` function allows the `ERC20SponsorPaymaster` owner to set the markup value of a specific protocol address. No events are emitted when this function is called, so it is difficult for off-chain observers to monitor its usage.

Recommendation: Consider adding an event to be emitted in `setMarkup()`.

Zyfi: Fixed in commit 488b7352.

Cantina Managed: Verified.

3.4.5 `setVerifier` lacks `address(0)`-check

Severity: Informational

Context: ERC20Paymaster.sol#L58-L61, ERC20Paymaster.sol#L228-L231, ERC20SponsorPaymaster.sol#L75-L81, ERC20SponsorPaymaster.sol#L314-L317

Description: If the `verifier` is set to `address(0)` the signature validation in `validateAndPayForPaymasterTransaction` will always revert.

Recommendation: This may be desirable as a way of shutting off the paymaster contract, but assuming the preferred way of doing upstream from the paymaster, consider adding a `if (_verifier == address(0)) revert Errors.InvalidAddress();` check to the appropriate places.

Zyfi: Agreed. Our plan to shut off a paymaster is not dependent on this, so we will add the protection. Fixed in PR 4.

Cantina Managed: Verified.