# Farcaster Audit Report

Prepared by Cyfrin

Version 1.0

**Lead Auditor**

Hans

November 5, 2023

# Contents

# 1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

# 2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

# 3 Risk Classification

| | **Impact: High** | **Impact: Medium** | **Impact: Low** |
|---|---|---|---|
| **Likelihood: High** | Critical | High | Medium |
| **Likelihood: Medium** | High | Medium | Low |
| **Likelihood: Low** | Medium | Low | Low |

# 4 Protocol Summary

Farcaster is a sufficiently decentralized protocol for building social apps. Sufficient decentralization means that two people who want to communicate can always do so. It also means developers have permissionless access to public data on the network.

# 5 Audit Scope

The initial review was conducted for a PR #5 and all the Solidity files inside the `src` were in scope. The mitigation review was conducted for a PR #6 and all the findings were resolved by the Farcaster team.

# 6 Executive Summary

Over the course of 9 days, the Cyfrin team conducted an audit on the Farcaster smart contracts provided by Farcaster. In this period, a total of 6 issues were found.

**Summary**

| | |
|---|---|
| Project Name | Farcaster |
| Repository | protocol |
| Commit | ba0508fc7e43. . . |
| Audit Timeline | Oct 24th - Nov 5th |
| Methods | Manual Review |

## Issues Found

| | |
|---|---|
| Critical Risk | 0 |
| High Risk | 0 |
| Medium Risk | 4 |
| Low Risk | 2 |
| Informational | 0 |
| Gas Optimizations | 0 |
| Total Issues | 6 |

## Summary of Findings

| | |
|---|---|
| [M 7.1.1] A signer can't cancel his signature before a deadline. | Resolved |
| [M 7.1.2] In `IdRegistry`, a recovery address might be updated unexpectedly. | Resolved |
| [M 7.1.3] `IdRegistry.transfer/transferFor()` might be revoked by a recovery address. | Resolved |
| [M 7.1.4] A removal signature might be applied to the wrong `fid` | Acknowledged |
| [L 7.2.1] Inconsistent validation of `vaultAddr` | Acknowledged |
| [L 7.2.2] Lack of validations for some admin functions | Acknowledged |

# 7 Findings

## 7.1 Medium Risk

### 7.1.1 A signer can't cancel his signature before a deadline.

**Severity:** Medium

**Description:** After signing a signature, a signer might want to cancel it for some reason. While checking other protocols, a signer can cancel by increasing his nonce. In this protocol, we inherit from OpenZeppelin's Nonces contract and there are no ways to cancel the signature before a deadline.

**Impact:** Signers can't invalidate their signatures when they want.

**Recommended Mitigation:** Recommend adding a function like `increaseNonce()` to invalidate the past signatures.

**Client:** Fixed by adding a base `Nonces` contract that exposes an external `useNonce()` function, enabling the caller to increment their nonce. Commit: `0189a1f`

**Cyfrin:** Verified.

### 7.1.2 In `IdRegistry`, a recovery address might be updated unexpectedly.

**Severity:** Medium

**Description:** There are 2 functions to update a recovery address, `changeRecoveryAddress()` and `changeRecoveryAddressFor()`. As `changeRecoveryAddress()` doesn't reset a pending signature that would be used in `changeRecoveryAddressFor()`, the below scenario would be possible.

- Alice decided to set a recovery as Bob and created a signature for that.
- But before calling `changeRecoveryAddressFor()`, Alice noticed Bob was not a perfect fit and changed the recovery address to another one by calling `changeRecoveryAddress()` directly.
- But Bob or anyone calls `changeRecoveryAddressFor()` after that and Bob can change the owner as well.

Of course, Alice could delete the signature by increasing her nonce but it's not a good approach for users to be allowed to use the previous signature.

**Impact:** A recovery address might be updated unexpectedly.

**Recommended Mitigation:** We should include the current recovery address in the recovery signature. Then the previous signature will be invalidated automatically after changing the recovery.

**Client:** Fixed by adding the current recovery address to `CHANGE_RECOVERY_ADDRESS_TYPEHASH`. Commit: `7826446`

**Cyfrin:** Verified.

### 7.1.3 `IdRegistry.transfer/transferFor()` might be revoked by a recovery address.

**Severity:** Medium

**Description:** In every `fid`, there exists an owner and a recovery address, each possessing identical authority, enabling either one to modify the other. But while transferring the `fid`, it just changes the owner and this scenario might be possible.

- Consider Bob with a `fid(owner, recovery)` intending to sell it.
- After receiving some funds, he transfers his `fid` to an honest user using `transfer()`.
- When the honest user is going to update the recovery address, Bob calls `recover()` by front running and seizes the account.

- In contrast to ERC721, a recovery address acts like an approved user for the NFT, empowered to change ownership at any moment. Notably, this authority is cleared during the transfer to prevent subsequent updates by any prior approvals.

**Impact:** `IdRegistry.transfer/transferFor()` might be revoked by a recovery address.

**Recommended Mitigation:** Recommend adding a function like `transferAll()` to update both `owner/recovery`.

**Client:** Fixed by adding `transferAndChangeRecovery` and `transferAndChangeRecoveryFor` to `IdRegistry`. Commit: `d389f9f`

**Cyfrin:** Verified.

### 7.1.4 A removal signature might be applied to the wrong `fid`.

**Severity:** Medium

**Description:** A remove signature is used to remove a key from `fidOwner` using `KeyRegistry.removeFor()`. And the signature is verified in `_verifyRemoveSig()`.

```
function _verifyRemoveSig(address fidOwner, bytes memory key, uint256 deadline, bytes memory sig)
↪   internal {
    _verifySig(
        _hashTypedDataV4(
            keccak256(abi.encode(REMOVE_TYPEHASH, fidOwner, keccak256(key), _useNonce(fidOwner),
            ↪   deadline))
        ),
        fidOwner,
        deadline,
        sig
    );
}
```

But the signature doesn't specify a `fid` to remove and the below scenario would be possible.

- Alice is an owner of `fid1` and she created a removal signature to remove a `key` but it's not used yet.
- For various reasons, she became an owner of `fid2`.
- `fid2` has a `key` also but she doesn't want to remove it.
- But if anyone calls `removeFor()` with her previous signature, the `key` will be removed from `fid2` unexpectedly.

Once a key is removed, `KeyState` will be changed to `REMOVED` and anyone including the owner can't retrieve it.

**Impact:** A key remove signature might be used for an unexpected `fid`.

**Recommended Mitigation:** The removal signature should contain `fid` also to be invalidated for another `fid`.

**Client:** Acknowledged. This is an intentional design tradeoff that makes it possible to register a fid and add a key in a single transaction, without knowing the caller's assigned fid in advance. We accept that this has the consequence described in the finding, and users should interpret key registry actions as “add key to currently owned fid.”

Nonces provide some protection against this scenario: if Alice wants to revoke her previous signature intended for `fid1`, she can increment her nonce to invalidate the signature.

**Cyfrin:** Acknowledged.

## 7.2 Low Risk

### 7.2.1 Inconsistent validation of `vaultAddr`

In `KeyManager.setVault()` and `StorageRegistry.setVault()`, there is a validation for address(0) but we don't check in the constructors.

```
File: audit-farcaster\src\KeyManager.sol
        vault = _initialVault;
        emit SetVault(address(0), _initialVault);
...
    function setVault(address vaultAddr) external onlyOwner {
        if (vaultAddr == address(0)) revert InvalidAddress();
        emit SetVault(vault, vaultAddr);
        vault = vaultAddr;
    }

```

**Client:** After internal discussion, we've decided to remove payments from the `KeyGateway` altogether and rely on per-fid limits in the `KeyRegistry` for now. We're keeping the gateway pattern in place, which gives us the ability to introduce a payment in the future if it becomes necessary.

We don't intend to redeploy the StorageRegistry with this deployment, but we will add this validation in the next version of the storage contract.

Commit: `11e2722`

**Cyfrin:** Acknowledged.

### 7.2.2 Lack of validations for some admin functions

In `KeyManager.setUsdFee()` and `StorageRegistry.setPrice()`, there are no upper limits.

While the protocol owner is regarded as a trusted party, it's still kind of an inconsistent implementation because there are min/max limits for `fixedEthUsdPrice` in `StorageRegistry.setFixedEthUsdPrice()`.

```
File: audit-farcaster\src\KeyManager.sol
    function setUsdFee(uint256 _usdFee) external onlyOwner {
        emit SetUsdFee(usdFee, _usdFee);
        usdFee = _usdFee;
    }

File: audit-farcaster\src\StorageRegistry.sol
    function setPrice(uint256 usdPrice) external onlyOwner {
        emit SetPrice(usdUnitPrice, usdPrice);
        usdUnitPrice = usdPrice;
    }
```

**Client:** After internal discussion, we've decided to remove payments from the `KeyGateway` altogether. (See the response to 7.2.1 for more details).

We don't intend to redeploy the StorageRegistry with this deployment, but we will add this validation in the next version of the storage contract.

Commit: `11e2722`

**Cyfrin:** Acknowledged.

# 8 Appendix

## 8.1 Reported Issues

The Farcaster team discovered some vulnerabilities after the audit had started. The Cyfrin team has included the self-reported issues in the review.

### 8.1.1 `KeyManager._ethUsdPrice()` returns the wrong price at the refresh block

**Description:** `KeyManager._ethUsdPrice()` returns the current price at the refreshing block.

```
File: audit-farcaster\src\KeyManager.sol
    function _ethUsdPrice() internal view returns (uint256) {
        uint256 fixedEthUsdPrice = storageRegistry.fixedEthUsdPrice();
        if (fixedEthUsdPrice != 0) return fixedEthUsdPrice;
        return storageRegistry.ethUsdPrice();
    }
```

As a result, in `KeyManager.addFor()`, the registration fee might be calculated wrongly.

**Client:** After internal discussion, we've decided to remove payments from the `KeyGateway` altogether. (See the response to 7.2.1 for more details).

Commit: `11e2722`

**Cyfrin:** Verified.