Veil Cash Security Review

Pashov Audit Group

Conducted by: zark, Udsen, eeyore, peanuts

February 12th 2025 - February 15th 2025

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 **veildotcash/veil_contracts_audit** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Veil Cash

Veil Cash is a fork of Tornado Cash, and is deployed on the Base Layer 2 (L2) blockchain. It leverages zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge) to enable users to achieve on-chain privacy and anonymity. Key changes from Tornado Cash are that Veil uses a proxy contract for deposits, upgrades are managed via VeilValidator.sol, and it includes mechanisms for on-chain user verification and whitelisting specific depositors.

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 - 326f141c0a0e8ebd14dccd3d93ecd61ae48e7e69

fixes review commit hash - e67267a65f5c17ffbf8305e284be2f75283fb4c5

Scope

The following smart contracts were in scope of the audit:

- `Veil_0005_ETH`
- `Veil_001_ETH`
- `Veil_005_ETH`
- `Veil_01_ETH`
- `Veil_1_ETH`
- `VeilValidatorV5`
- `iVerify`
- `verify`

7. Executive Summary

Over the course of the security review, zark, Udsen, eeyore, peanuts engaged with Veil Cash to review Veil Cash. In this period of time a total of **3** issues were uncovered.

Protocol Summary

Protocol Name	Veil Cash
Repository	https://github.com/veildotcash/veil_contracts_audit
Date	February 12th 2025 - February 15th 2025
Protocol Type	Privacy service

Findings Count

Severity	**Amount**
Low	3
Total Findings	**3**

Summary of Findings

ID	Title	Severity	Status
[L-01]	Depositor count can accidentally increase	Low	Resolved
[L-02]	Incorrect event emissions	Low	Resolved
[L-03]	Using flashloans to bypass balance requirement	Low	Acknowledged

8. Findings

8.1. Low Findings

[L-01] Depositor count can accidentally increase

In VeilValidatorV4, the `veilManager` can set allowed depositors. If the `veilManager` accidentally calls the sets the same `_depositor` as true more than once, the `depositorCount` will increase

```
function setAllowedDepositor
   (address _depositor, bool _isAllowed, string memory _details) public {
        if (msg.sender != veilManager) revert OnlyVeilManager();
>       allowedDepositors[_depositor] = _isAllowed;
        depositorDetails[_depositor] = _details;
>       if (_isAllowed) {
            depositorCount++;
        } else {
            depositorCount--;
        }
        emit DepositorStatusChanged(_depositor, _isAllowed, _details);
    }
```

To prevent such an issue, check that the depositor is set or unset before increasing or decreasing the count. Something like:

```
function setAllowedDepositor
   (address _depositor, bool _isAllowed, string memory _details) public {
        if (msg.sender != veilManager) revert OnlyVeilManager();

       if (_isAllowed) {
+           if(allowedDepositors[_depositor] != _isAllowed){
            depositorCount++;
        }
        } else {
+        if(allowedDepositors[_depositor] != _isAllowed){
            depositorCount--;
        }
        }

        allowedDepositors[_depositor] = _isAllowed;
        depositorDetails[_depositor] = _details;
        emit DepositorStatusChanged(_depositor, _isAllowed, _details);
    }
```

[L-02] Incorrect event emissions

`VeilValidatorV4::deposit005ETH` function incorrectly emits the `Deposited` event with the wrong `poolSize`. The `005` pool corresponds to pool ID 4, but the event is emitted with ID 0 instead.

```
- emit Deposited(msg.sender, 0, depositAmount, fee);
+ emit Deposited(msg.sender, 4, depositAmount, fee);
```

Also, `VeilValidatorV4::setRewardsTracker`, `VeilValidatorV4::setVeilVerifiedOnchain` and `Veil_005_ETH::updateValidatorContract` are important state changing functions, but they do not emit events to log the updates they perform.

Finally, `VeilValidatorV4::VeilTokenAmountSet` event and `Veil_005_ETH::UpdateVerifiedDepositor` event are unused and they can be safely removed to reduce contract size and gas costs.

[L-03] Using flashloans to bypass balance requirement

Depositors are required to hold a specific amount of `VEIL` tokens to be able to deposit into each pool. Each pool has its own required amount, and this check is enforced by the `VeilValidatorV4::_hasVeil` function:

```
function _hasVeil(address _depositor, uint8 _poolSize) internal view returns
  (bool) {
     return veilToken.balanceOf(_depositor) >= poolVeilAmount[_poolSize];
}
```

However, a user can bypass this requirement with a flash loan, borrowing `VEIL` tokens just before depositing and repaying them within the same block. This lets him meet the `_hasVeil` check without any real commitment, undermining the deposit requirement.

To prevent this exploit, consider using a time based validation mechanism to ensure depositors maintain the required `VEIL` balance for a enough duration before the deposit.