Vaultody, Wallet Factory Security Audit

Report Version 1.0

May 2, 2024

Conducted by:
Atanas Dimulski, Independent Security Researcher

Contents

1 About Dimulski

Dimulski is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews.

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 trained experts try to find as many vulnerabilities as possible. I 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 Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 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 are not so critical.

3.2 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.

3.3 Action required for severity levels

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Vaultody engaged Dimulski to review their wallet factory smart contract protocol from April 26, 2024, to April 29, 2024.

Overview

Project Name	Vaultody, Wallet Factory
Repository	https://bitbucket.org/menadev/vaultody-gas-tanker/src/master/
Commit hash	3cdd14f38f62c2bcadd2ec687fa74ecd27258712
Resolution	e8740c1689178fcea802a492649c7715fa65d2d1
Methods	Manual review

Timeline

-	April 26, 2024	Audit kick-off
v0.1	April 29, 2024	Preliminary report
v1.0	May 2, 2024	Mitigation review

Scope

src/CustodialWallet.sol
src/CustodialWalletFactoryV2.so

Issues Found

Critical	0
High	0
Medium	1
Low	4

5 Findings

5.1 Medium

5.1.1 The approve() function will revert for non standard ERC20 tokens

Severity: Medium

Description:
The approve() function in the CustodialWallet.sol is intended to allow the of owner the contract, to approve other addresses to spend the tokens contained within the contract.

```
    function approve(
        address tokenAddress,
        uint256 contractType,
        address spender,
        uint256 amount,
        uint256 tokenId
    ) public virtual onlyOwner {
        if (contractType == 0) {
            bool approval = IERC20(tokenAddress).approve(spender, amount);
            require(approval, "Approval was unsuccessful");
        } else if (contractType == 1) {
            IERC721(tokenAddress).approve(spender, tokenId);
        } else if (contractType == 2) {
            IERC1155(tokenAddress).setApprovalForAll(spender, true);
        } else {
            revert("Unsupported contract type");
        }
    }
```

However not all ERC20 tokens, return boolean on approve, which could cause token approvals to fail. For example the USDT token approve() function, is implemented in the following way:

```
    function approve(address _spender, uint _value) public onlyPayloadSize(2 * 32) {

        // To change the approve amount you first have to reduce the addresses`
        //  allowance to zero by calling `approve(_spender, 0)` if it is not
        //  already 0 to mitigate the race condition described here:
        //  https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
        require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));

        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
    }
```

As can be seen from the above code snippet the USDT approve() function does not return a boolean value when called, and requires the spender allowance to be 0, in order to mitigate race conditions. Whenever the approve() function is called on a USDT, or another weird ERC20 token, that doesn't return a boolean value, the function will revert, severely limiting the functionality of the contract.

Recommendation:
Instead of IERC20(tokenAddress).approve(spender, amount) use Openzeppelin SafeERC20.forceApprove()

Resolution: Resolved.

5.2 Low

5.2.1 Unsafe ownership transfer

Severity: Low

Description:
The CustodialWallet.sol contract, inherits from the CustodialOwnable.sol contract, which transfers ownership in the following way:

```
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(
            newOwner != address(0),
            "Ownable: new owner is the zero address"
        );
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
```

It is possible that the new owner is set to an address which is not controlled by the previous owner. Thus making the entire contract obsolete.

Recommendation:
Implement an ownership transfer process similar to Ownable2Step.sol.transferOwnership()

Resolution: Resolved.

5.2.2 Missing remove allowance functionality for ERC1155 tokens

Severity: Low

Description:
The CustodialWallet.sol contract is missing a remove allowance functionality for ERC1155 tokens, in case an address that has been previously given allowance is compromised, it is not possible for the owner of the CustodialWallet.sol contract instance to remove the given allowance for ERC1155 tokens.

Recommendation:
Modify the approve() function to the following:

```
    function approve(
        address tokenAddress,
        uint256 contractType,
        address spender,
        uint256 amount,
        uint256 tokenId,
        bool approval
    ) public virtual onlyOwner {
        ...
        else if (contractType == 2) {
            IERC1155(tokenAddress).setApprovalForAll(spender, approval);
        }
        ...
    }
```

Resolution: Resolved.

5.2.3 Missing address(0) check in CustodialWalletFactoryV2.create() function

Severity: Low

Description:
The create() function in the CustodialWalletFactoryV2et.sol contract is missing a address(0) check, thus a user can supply an address(0) by mistake, as parameter and create a wallet that can't be used.

Recommendation:
Add a custom error error InvalidAddress(); in the CustodialWalletFactoryV2.sol contract, and add the following check in the create() function:

```
    if(owner == address(0)) revert InvalidAddress();
```

Resolution: Acknowledged.

5.2.4 Unused variables in the CustodialWalletFactoryV2.sol contract

Severity: Low

Description:
In the CustodialWalletFactoryV2et.sol contract the following variables are not used anywhere in the contract:

```
        uint256 private constant _MAX_ARRAY_BOUNDS = 2000;
        uint256 private constant _MAX_ARRAY_CALCULATE_BOUNDS = 10_000;
        mapping(bytes32 => address) public wallets;
```

Recommendation:
Remove the above mentioned variables, as they are not used anywhere in the contract.

Resolution: Acknowledged.