PUBLIC

Code Assessment

of the ERC20 Pods Smart Contracts

December 19, 2022

Produced for

by

Contents

1 Executive Summary

Dear 1inch team,

Thank you for trusting us to help 1inch with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of ERC20 Pods according to Scope to support you in forming an opinion on their security risks.

1inch implements an extension for OpenZeppelin's `ERC20` implementation, that allows users to register so-called `Pods`, that are called on a balance update.

The most critical subjects covered in our audit are functional correctness, integration with OpenZeppelin's `ECR20` token implementation, and access control. Security regarding all the aforementioned subjects is good.

The general subjects covered are code complexity, documentation and event handling. Security regarding all the aforementioned subjects is improvable. Code complexity is improvable due to the custom `AddressArray` implementation. Documentation is non-existing.

In summary, we find that the codebase provides a satisfactory level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	7
• Code Corrected	3
• Risk Accepted	1
• Acknowledged	2
• No Response	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the `contracts` folder of the ERC20 Pods repository. No documentation for the smart contracts reviewed was available. Following files from the repository contracts folder were part of the assessment scope:

```
interfaces/IERC20Pods.sol
interfaces/IPod.sol
ERC20Pods.sol
Pod.sol
ReentrancyGuard.sol - (file added in Version 2)
```

On top of that, we reviewed the `AddressArray` and `AddressSet` libraries that are used by `ERC20Pods`, the scope for those libraries is limited to their use within the `ERC20Pods` contract.

We expect the `ERC20Pods` contract to be only implemented on top of OpenZeppelin's `ERC20` implementation, version `0.4.8`.

The table below indicates the code versions relevant to this report and when they were received.

ERC20 Pods

V	Date	Commit Hash	Note
1	22 November 2022	5fc07f7e711d1d74b37cbe7120a9f3117e12b4bd	Initial Version
2	14 December 2022	f975e2eaec9714e66163c1826044f722660a14a2	Version 2

AddressArray and AddressSet

V	Date	Commit Hash	Note
1	22 November 2022	92a4754b880dbababcb08cc30050c57bdd0573da	Initial Version

For the solidity smart contracts, the compiler version `0.8.17` was chosen.

2.1.1 Excluded from scope

Any contracts not mentioned above, mock and testing contracts that might rely on the scoped contracts are not part of the scope. Imported libraries are assumed to behave according to their specification and are not part of the assessment scope.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

1inch offers a functionality extension for ERC20 that allows the token to notify another contract, called `Pod` that some internal balances have changed. With this extension comes the abstract `Pod` contract that should be implemented for other contracts to profit from this extension. The `ERC20Pods` records for each user, which pod was registered, `AddressSet` is used to this end.

AddressArray

This library contains logic for managing lists of `address` that are in the form of `mapping(uint256 => uint256)`, stored in a `struct`. The length of the list is stored in the 96 MSB of the first mapping's entry, then the addresses are stored in the 160 LSB of each mapping entry, starting from 0 up until `list's length - 1`. The available functions are:

- `length`: returns the length of the list
- `at`: returns the address stored at the provided index. The function returns `address(0)` if the index >= list's length.
- `get`: returns a memory array containing the stored address
- `push`: adds an address at the end of the list and returns the new length
- `pop`: removes the last element of the list. The function reverts if the list is already empty.
- `set`: updates the address at the provided index. The function reverts if the provided index is out-of-bounds.

AddressSet

This library is built on top of the functionality provided by `AddressArray` to implement a set of addresses. The `struct` used here includes an array and a lookup table storing the position of each address in the array. The function made available by this library are:

- `length`: returns the cardinality of the set
- `at`: returns the address stored at the provided index. The function returns `address(0)` if the index >= list's length.
- `contains`: returns `true` if the set contains the queried address.
- `add`: adds the provided address to the set if not already included, returns `true` if the address was not already in the set, and `false` if the set already contained the set. The new length is stored in the lookup table.
- `remove`: removes the provided address from the set, returns `true` on success, and `false` if the provided address is not in the set. Note that this may alter the position of the last address in the list.

Pod

This abstract contract implements the interface `IPod`. A concrete implementation of this contract should define the function `updateBalances(address from, address to, uint256 amount)` that can trigger some action inside the `Pod` contract, e.g. update internal balances. The `Pod` must be initiated with a token address at construction and the contract provides the `onlyToken` modifier, which reverts if the `msg.sender` is not the token.

ERC20 Pods

This abstract contract is an extension for ERC20 that allows address to register `Pods`. The pods will be called via the internal function `_updateBalances`, that will trigger `IPod.updateBalances` on the `Pod` upon a token transfer or pod management. The contract will keep a set of the registered pods per address. The internal function `_updateBalances` does not care about the success or failure of the call to `IPod.updateBalances`. The contract receives a maximum number of pods each account is allowed to have, this limit is set at deployment and cannot be changed later. Each call to `IPod.updateBalances` has a maximum gas allocation of `200_000`.

The pods management functions are:

- `addPod(address pod)`: will add the pod provided as argument to the set of pods of the caller. The function will revert if the set already contains the pod, if `pod == address(0)`, or if the pods limit is exceeded. When a pod is added, `IPod.updateBalances` is called on the pod contract to update the balance.
- `removePod(address pod)`: will remove the pod provided as argument from the set of pods of the caller. The function will revert if the set does not contain the pod. When a pod is removed, `IPod.updateBalances` is called on the pod contract to reduce the balance.
- `removeAllPods`: will remove all the pods from the caller.

Upon a token transfer, the callback `_afterTokenTransfer` will be triggered on the contract. The function will first iterate through the sender's pods, updating the balances with the following rule:

- if both share a pod, the balances update is such that `from=sender` and `to=receiver`, like a token transfer.
- if the sender is in a pod that is not shared with the receiver, the balances update is such that `from=sender` and `to=address(0)`, like a token burn.
- if the receiver is in a pod that is not shared with the sender, the balances update is such that `from=address(0)` and `to=receiver`, like a token mint.

2.2.1 Trust model

The `ERC20Pods` contract should not be used on top of `ERC20` with special behaviors like, tokens with fees or rebalancing tokens.

We assume that the constructor parameters upon deployment (`podsLimit_` and `podCallGasLimit_`) are choosen correctly. Incorrect parameters may break the ERC20Pods contract. Amongst others this applies if the pods to be updated exceed the gas available. While the code of Version 2 prevents initializing `podsLimit_` with 0 (which would inhibit the pod functionality) there is nothing preventing to set a too low or too high `podCallGasLimit` which would inhibit successful calls.

2.2.2 Changes in Version 2

- Instead of the previously hardcoded `200_000` gas, the amount of gas for the calls to the pods is now specified as constructor argument.
- Function `_afterTokenTransfer` now features a reentrancy protection. Also, the functions `balanceOf` and `podBalanceOf` revert if the reentrancy protection is active.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	4

5.1 No Documentation

Security Low Version 1

No documentation is available for ERC20Pods. This abstract contract is intended to be used by third parties hence documentation is vital to avoid issues. For authors of pods it must be clearly documented what they have to take into account and what they can rely on, such as:

- Failed calls to `Pod.updateBalances()` are ignored, consequently authors of pods must be aware of the consequences for their pods
- Amount of gas available
- When exactly the token triggers `Pod.updateBalances()`: Upon non-zero token transfers and when the pod is added/removed from an account having non-zero balance. Misunderstandings by a developer of a Pod may lead to correctness issues.

The trust model should be clearly specified, including:

- How exactly pods are trusted / untrusted
- Whether only trusted parties can add/remove pods to/from an account. If this holds, the docs should clearly state that a developer extending ERC20Pods must adhere to this

Furthermore not all token holders, e.g. contracts can call `addPod()` themselves. The documentation may elaborate on this topic, e.g., what can be assumed / what the limitations are.

5.2 Consistency on Zero Amount Transfers

Design Low Version 1 Acknowledged

The ERC20 standard specifies `Note Transfers of 0 values MUST be treated as normal transfers and fire the Transfer event.`. For consistency, it may make sense to inform the registered pods about 0 balance actions and 0 amount transfers. If this behavior is desired the way it is, it should be mentioned somewhere so that Pod developer are aware that 0 balance or 0 amount transfers are not notified to the Pod.

Acknowledged:

1inch acknowledged the issue and decided to leave the code as it is.

5.3 Side Effects of `_updateBalances()`

Security | Low | Version 1 | Risk Accepted

While the gas check prevents direct reentrancy into the token on functions changing the balance, `200_000` gas is enough to make some other state changes that could affect to-be-updated pods. Notably upon updating the first pod A, this contract may interact with another pod B which is to be updated later in the sequence and hence does not yet know about these balance changes pod A currently executing already is aware of. While we have not uncovered any direct issue, a badly designed or adversarial pod could be problematic. No trust model nor specification covering this scenario is available.

Risk accepted:

1inch is aware of and accepts the risk.

Version 2 introduced a reentrancy guard. Note that this can be leveraged by a pod to detect such a scenario and revert. While the state of the reentrancy guard itself cannot be querried direcly, public functions `balanceOf` and `podBalanceOf` now feature the `nonReentrant(View)` modifier and will revert if called in such a situation. Hence in the scenario described above pod B could call `balanceOf()` and be protected.

5.4 Zero Address Consistency in `AddressArray`

Design | Low | Version 1 | Acknowledged

When querying an index that is out-of-bounds with `AddressArray.at`, the function does not revert and returns `address(0)`. Thus, it is not possible to distinguish between an `address(0)` that would effectively be part of the array, and an out-of-bounds access.

Acknowledged:

1inch is aware of the issue and states that in the current use case, no pod with `address(0)` can be added. While this is true in the case of `ERC20Pods`, it can still be an issue for other contracts using the library.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3

- ERC20Pods podsLimit Sanitization Code Corrected
- Missing Events Code Corrected
- Operations Order on Pod Removal Code Corrected

6.1 ERC20Pods `podsLimit` Sanitization

Design Low Version 1 Code Corrected

The `podsLimit` in the `ERC20Pods`'s constructor is never sanitized. If `podsLimit` is zero, the functionality added by `ERC20Pods` cannot be used, so it would not make sense to allow setting `podsLimit=0`.

Code corrected:

The constructor of the `ERC20Pods` contract now checks that `podsLimit` is not zero.

Note that in Version 2 the constructor takes a second parameter `podCallGasLimit_` which is not sanitized. Unsuitable values could make the ERC20Pods unusable.

6.2 Missing Events

Design Low Version 1 Code Corrected

Typically, events help track the state of the smart contract. To be able to reconstruct the state offchain, events should be emitted when users add and remove pods.

Code corrected:

Two events `PodAdded` and `PodRemoved` have been added and are emitted whenever a pod is added/removed.

6.3 Operations Order on Pod Removal

Design Low Version 1 Code Corrected

When a pod is removed with `removePod`, it is first removed from the internal address set, and then the balances are updated. But when calling `removeAllPods`, the balances are updated before the pod is removed from the address set. Thus, there are two different behaviors for the same action and the potential for inconsistencies arises.

Code corrected:

The function `removeAllPods` now follows the order of `removePod` by first removing the pod from the address set and then update the balances.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Failed Call to Pod Is Silent

Note Version 1

`ERC20Pods._updateBalances()` calls the pod with a fixed amount of gas. If this call fails, the execution nonetheless continues normally in order to not block the ERC20. Users must be aware that a failed call to a pod is silent and will not emit any event.

7.2 Integrations May Break Due to Gas Requirements

Note Version 1

A transfer of an ERC20Pods may require significantly more gas than the transfer of a normal ERC20. This especially applies when sender and receiver are connected to multiple distinct pods. Moreover, the current abstract contract ERC20Pods allows an user to register any arbitrary pod for his address.

In the worst case each of the pods uses the full 200'000 gas available. When sender and receiver have distinct pods this amounts to 2 * `podsLimit` gas.

Integrations must take this into account in order to avoid running into problems such as, but not limited to:

An example could be a liquidation of a position where in an extreme case multiple different ERC20Pods where each sender/receiver is connected to several pods must be transferred. The liquidation may not be possible due to the gas requirement exceeding the block gas limit.

7.3 Not Exactly _POD_CALL_GAS_LIMIT Available

Note Version 1

A Pod cannot rely on having exactly 200'000 gas available upon being called. While it is taken into account that a maximum of 63/64 of the remaining gas can be passed to the call, due to the overhead between the check and the call:

```
if lt(div(mul(gas(), 63), 64), _POD_CALL_GAS_LIMIT) {
        mstore(0, exception)
        revert(0, 4)
    }
        pop(call(_POD_CALL_GAS_LIMIT, pod, 0, ptr, 0x64, 0, 0))
    }
```

in a corner case scenario the call may receive slightly less gas.

7.4 Order of Pods in AddressSet

Note Version 1

Users msut be aware that upon pod removal, the order in the pods in the `AddressSet` may change, so two calls to `podAt` with index X, with a call to `removePod` in-between, may not yield the same result.

7.5 Unaffected Elements of the Output Memory Array on `AddressArray.get()`

Note Version 1

When providing an output memory array to the function `AddressArray.get(Data storage self, address[] memory output)`, users must be aware that if `length(self) < output.length()`, only the `length(self)` first elements of `output` will be overwritten, leaving the remaining elements of `output` untouched.