OpenZeppelin | security

0x Settler Audit

April 8, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-02-21 To 2024-03-22
Languages	Solidity

Total Issues	9 (4 resolved, 2 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	4 (0 resolved, 2 partially resolved)
Notes & Additional Information	5 (4 resolved)

Scope

We audited the 0xProject/0x-settler repository at commit f18e966.

In scope were the following files:

```
src
├── allowanceholder
│   ├── AllowanceHolder.sol
│   ├── AllowanceHolderBase.sol
│   ├── AllowanceHolderContext.sol
│   ├── AllowanceHolderOld.sol
│   ├── IAllowanceHolder.sol
│   ├── TransientStorage.sol
│   ├── TransientStorageBase.sol
│   ├── TransientStorageLayout.sol
│   └── TransientStorageMock.sol
├── core
│   ├── Basic.sol
│   ├── MakerPSM.sol
│   ├── OtcOrderSettlement.sol
│   ├── Permit2Payment.sol
│   ├── SettlerErrors.sol
│   ├── UniswapV2.sol
│   └── UniswapV3.sol
├── deployer
│   ├── Deployer.sol
│   ├── Feature.sol
│   ├── Nonce.sol
│   └── TwoStepOwnable.sol
├── proxy
│   └── ERC1967UUPSUpgradeable.sol
├── utils
│   ├── CheckCall.sol
│   ├── FreeMemory.sol
│   ├── MultiCall.sol
│   ├── Panic.sol
│   ├── ProxyMultiCall.sol
│   ├── Revert.sol
│   └── UnsafeMath.sol
├── Context.sol
├── IERC20.sol
├── ISettlerActions.sol
├── Settler.sol
└── SettlerAbstract.sol
```

System Overview

0x Settler provides various settlement flows for users during the token settlement phase. With the addition of the allowance holder as an intermediary step, users can now utilize transient storage along with Permit2. This allows users to sign transactions granting the Settler the ability to execute operations via the allowance holder. This consumes token permits after users' transactions which makes token transfers more secure by not holding any funds, allowances or token permits, thereby further minimizing gas costs and custody requirements for such flows. This gives users the ability to conduct flows such as Uniswap V2 swaps, Uniswap V3 VIP swaps, Curve swaps, OTC orders, multi-collateral Dai (DSS) and meta-transactions, all while minimizing risk, custody requirements, and gas costs.

However, it should be noted that the codebase does make extensive use of assembly for both low-level memory manipulation and high-level logic. Since the use of assembly discards many safety features provided by Solidity, it should usually be limited to small, well-defined code blocks. This would make the codebase more robust, readable, and extensible. On the other hand, this may not be practical without markedly modifying the codebase. As such, we recommend significantly expanding the test suite to ensure consistency with the specification and validating all expected behavior.

While the implementation is clean and efficiently optimized, it is hard to comprehend it due to the lack of documentation. Assembly is, needless to say, more difficult to understand than Solidity. Thus, the code would benefit from detailed technical documentation which is an important aspect for both users and developers. Code can become indecipherable without it and hinder users' understanding. This increases the chances of misinterpreting the code which can lead to potentially dangerous issues with the future implementations of the Settler contracts.

Security Model and Trust Assumptions

Users should take caution when interacting with the Settler contracts without 0x's additional tooling or their front end. Due to the use of permit signatures, any misconfiguration can lead to transactions causing assets to be left behind in the contracts, which assets can then be freely moved by any malicious user. It should also be noted that there is an assumption that all signatures are being properly constructed either with 0x's tooling or their front end. Furthermore, since allowance holder contracts use `tx.origin`, they might behave unexpectedly if the caller is ERC-4337 account abstraction enabled.

Privileged Roles

Overall, the Settler contracts are decentralized and do not have privileged roles. However, there are some Deployer contracts present with the intention of being the single source of deployment for all versions of 0x V5 settlement contracts which allows for deterministic addresses to be pre-computed. Ownership and authorization to deploy through the Deployer contracts would be achieved through a multisig wallet.

Deployment Assumptions

The idea behind deploying `AllowanceHolderOld.sol` to networks that have not yet upgraded is to be able to use the `TSTORE/TLOAD` opcodes on these networks. Since the contracts are using version `0.8.24` to compile, the contracts can be compiled to use the `PUSH0` opcode. However, there are some networks that have not been upgraded to use the `PUSH0` as well. This will prevent the contracts from being deployed on such networks. The same can be said for their use of the new `MCOPY` opcode. It is assumed that these facts are kept in mind when choosing which networks the Settler contracts will be deployed to.

Error-Prone Inheritance Pattern

Within the Deployer contacts, there is an error-prone inheritance pattern - specifically in the `TwoStepOwnable.sol`, `Permit2Payment.sol`, and `ERC1967UUPSUpgradeable.sol` contracts. This increases the chances of introducing bugs that could sneak into production and

cause issues. Thus, it is important for the 0x developers to fully document the most sensitive parts of these contracts and take extra caution when iterating on the current 0x V5 settlement contracts.

Low Severity

L-01 Missing Docstrings

Throughout the codebase, there are several parts that do not have docstrings, particularly `internal` functions. While the Solidity NatSpec does not require docstrings for `internal` functions, much of the codebase uses low-level assembly. This intrinsically makes it harder to comprehend the code due to the lack of documentation. Some of the files with missing docstrings are:

- `AllowanceHolder.sol`
- `AllowanceHolderBase.sol`
- `AllowanceHolderContext.sol`
- `Basic.sol`
- `CheckCall.sol`
- `Context.sol`
- `Deployer.sol`
- `MultiCall.sol`
- `OtcOrderSettlement.sol`
- `Panic.sol`
- `Permit2Payment.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially resolved in pull request #101.*

L-02 Incomplete Docstrings

There are several parts of the codebase that have incomplete docstrings for either function parameters or return values:

- The upgrade function in `ERC1967UUPSUpgradeable.sol`
- The upgradeAndCall function in `ERC1967UUPSUpgradeable.sol`

- The exec function in `IAllowanceHolder.sol`
- The transferFrom function in `IAllowanceHolder.sol`
- The PERMIT2_TRANSFER_FROM function in `ISettlerActions.sol`
- The METATXN_PERMIT2_TRANSFER_FROM function in `ISettlerActions.sol`
- The METATXN_SETTLER_OTC_PERMIT2 function in `ISettlerActions.sol`
- The UNISWAPV3_PERMIT2_SWAP_EXACT_IN function in `ISettlerActions.sol`
- The METATXN_UNISWAPV3_PERMIT2_SWAP_EXACT_IN function in `ISettlerActions.sol`
- The UNISWAPV2_SWAP function in `ISettlerActions.sol`
- The sellTokenForTokenToUniswapV3 function in `UniswapV3.sol`

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of any contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially resolved in pull request #99. The 0x Project team stated:*

> *The "functions" defined in `ISettlerActions` aren't real functions.*

L-03 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled.

Throughout the codebase, there are multiple floating pragma directives, with most of the contracts using `solidity ^0.8.24`. This could either introduce old compiler bugs or expose the codebase to undiscovered vulnerabilities in recently released compiler versions.

Consider using a fixed pragma directive.

Update: *Acknowledged, not resolved.*

L-04 Duplicated Imports

Within `Deployer.sol`, the `isNull` function is imported from both the `Feature.sol` file and the `Nonce.sol` file.

Consider renaming the `isNull` functions in both `Feature.sol` and `Nonce.sol` files to improve the overall clarity and readability of the codebase.

Update: *Acknowledged, not resolved. The 0x Project team stated:*

> *The two `isNull` implementations cannot be confused for each other because distinct user-defined types are not convertible.*

Notes & Additional Information

N-01 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact.

Consider adding a NatSpec comment containing a security contact above the contract definitions. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved in pull request #100.*

N-02 Unused or Unnecessary Code

Throughout the codebase, there are instances of unused code or variables:

- The `Permit2BatchPaymentAbstract` contract in `Permit2Payment.sol`
- The `Permit2BatchPayment` contract in `Permit2Payment.sol`
- The `FeeTokenMismatch` error in `Permit2Payment.sol`
- The `Permit2Payment.sol` import in `UniswapV3.sol` is already inherited via the `SettlerAbstract` contract.

- The `isMultiHop` named return variable for the `_isPathMultiHop` function in `UniswapV3.sol`

Consider addressing the above in order to make the codebase more performant and optimized in terms of gas consumption.

Update: *Resolved in pull request #96.*

N-03 Inconsistent Use of Named Returns

The `Deployer` contract has inconsistent usage of named returns in its functions.

Consider being consistent with the use of named returns throughout the codebase.

Update: *Acknowledged, not resolved.*

N-04 Gas Optimization

Potential gas cost improvements were found throughout the codebase:

- In line 172 of `OtcOrderSettlement.sol`, a strict inequality would save 3 gas units and remove the redundant case when `takerAmount == maxTakerAmount` as it will not do an unnecessary `MSTORE` (another 3 gas) when setting `takerAmount` to `maxTakerAmount`.
- Splitting revert statements will save gas by not having a boolean operator in the `if` statement.
  - Line 203 of `Deployer.sol`
  - Line 256 of `Deployer.sol`
  - Line 40 of `ERC1967UUPSUpgradeable.sol`
  - Line 220 of `Permit2Payment.sol`
- Removing assert checks from constructors will save gas on deployment. These checks can be moved to the test suite to ensure that further installments are properly checked. Although this saves gas, it could be problematic if the wrong value is not caught, and depending on the developers' needs, it can be optional.
  - Line 27 of `AllowanceHolderOld.sol`
  - Line 26 of `TransientStorageMock.sol`
  - Lines 62-63 of `OtcOrderSettlement.sol`

    - Line 138 of `Deployer.sol`
    - Line 153 of `Deployer.sol`
    - Line 41 of `TwoStepOwnable.sol`
    - Line 83 of `TwoStepOwnable.sol`
    - Line 163 of `TwoStepOwnable.sol`
    - Lines 86-87 of `ERC1967UUPSUpgradeable.sol`
    - Line 227 of `ERC1967UUPSUpgradeable.sol`
    - Line 251 of `ERC1967UUPSUpgradeable.sol`

- In lines 136-137 of `OtcOrderSettlement.sol`, two separate calls are made to different `_transferFrom` functions. In the first one, the `_isForwarded()` parameter is hard set to `false`. In the second one, there is a check being performed on the sender to see whether it is `_isForwarded()` and reverts if `true`. For this optimization, if the intention is to not allow a `forwarded` sender, it should check this in the first function and fail early to save gas. Otherwise, consider hard setting `forwarder` to `false` in the second function as well.

To reduce the gas consumption during code execution, consider refactoring the code to be more performant.

Update: *Resolved in pull request #97. The 0x Project team stated:*

> *Setting `takerAmount = maxTakerAmount` should not be compiling an `MSTORE` instruction (unless `takerAmount` has spilled into memory); it should be compiling to `SWAPN POP` instead. The asserts in the constructor should compile-out and produce no code either at deploy-time or runtime; what actually happens when one of the asserts is violated is that you get a compiler error because immutables remain unset in the constructor.*

N-05 Typographical Errors

There are a few typos throughout the codebase:

- In line 485 of `README.md`, "we can one or more transfers" should be "we can do one or more transfers".
- In line 485 of `README.md`, "Allowing us to take either a buy token feel" should be "buy token fee".
- In line 104 of `Settler.sol`, "could interaction with" should be "could interact with".
- In line 66 of `OtcOrderSettlement.sol`, "transfering" should be "transferring".
- In line 37 of `UniswapV2.sol`, "explicity" should be "explicitly".

- In line 12 of `CheckCall.sol`, "succeded" should be "succeeded".

Consider fixing the above typographical errors.

Update: *Resolved in pull request #98.*

Conclusion

The auditors were not able to identify any critical or high-severity issues during this audit, indicating an overall healthy protocol design and that the client prioritizes security when working with assembly. Several minor vulnerabilities were found that would help increase the code's quality. Although the codebase is highly optimized, the auditors were still able to identify several opportunities for improvement. Many of the 0x team’s optimizations have resulted in assembly implementations of highly complex and critical parts of the code. However, much of the codebase has minimal documentation. Providing documentation, particularly for these areas, would greatly increase the code's quality. Communication with the 0x team has been fruitful and all the auditors' questions were answered promptly and thoroughly.