0X-Settler Intent Upgrade

Smart Contract Security Assessment

January 23, 2025

SUMMARY

ID	Description	STATUS
L1	Slippage check is optional in _checkSlippageAndTransfer	Resolved
A1	Compiler bugs	Info

ABSTRACT

Dedaub was commissioned to perform a security audit of an upgrade to the **0x-Settler** smart contracts which aimed to build an intent protocol on top of its existing meta-transaction/routing functionality.

Overall we found one low severity security issue and no major ones in the code. It should be noted that the core routing and meta transaction logic were out of scope and assumed to be safe (this code was also previously audited).

BACKGROUND

The 0x Settler protocol is designed to facilitate secure and gas-efficient asset swaps across a range of blockchain protocols. This is achieved by offering users the ability to execute meta-transactions, wherein users can specify a list of actions they want the protocol to execute on their behalf (generally different swaps).

Due to the generality of the system, the protocol leverages the Permit2 contract to handle asset permissioning, as it enables short-lived non-reusable permissions, avoiding dangling approvals. When submitting a transaction to the protocol the user also signs off on a witness parameter, which includes the tolerated slippage and the list of actions the protocol is to perform on behalf of the user.

We were tasked with auditing a PR which aimed to introduce an *intent* based protocol on top of this existing metatransaction functionality. The core of the PR is the relaxation of the witness parameter, with the new version only requiring the user to sign off on the slippage, allowing the submitter of the transaction to decide the actions (so long as the slippage is still satisfied, else the contract reverts).

The primary new contract is the `SettelerIntent` contract, which is based on the existing `SettlerMetaTxn` contract. The functionality is nearly identical, less the aforementioned changes to the witness. The PR also adds an `Intent.sol` contract for each supported chain, as the protocol integrates additional dApps for each chain. The logic for each integration is inherited from the existing `MetaTxn.sol`, with the bulk of the code in `Intent.sol` aiming at addressing compilation issues.

SETTING & CAVEATS

This audit report mainly covers the changes at the audit_dedaub_jan25 tag of the **Ox-Settler** repository.

2 auditors worked on the codebase for 2 days (each) on the following contracts:

```
src
├── Settler.sol
├── SettlerAbstract.sol
├── SettlerBase.sol
├── SettlerIntent.sol
├── SettlerMetaTxn.sol
├── chains
│   └── \<Chain>
│       ├── Common.sol
│       ├── Intent.sol
│       ├── MetaTxn.sol
│       └── TakerSubmitted.sol
└── core
    ├── Permit2Payment.sol
    └── univ3forks
        └── SolidlyV3.sol
```

We do not list the entire directory structure due to brevity, with `\<Chain>` representing a placeholder for the following: Arbitrum, Avalanche, Base, Blast, Bnb, Gnosis, Ink, Linea, Mainnet, Mantle, Mode, Optimism, Polygon, Scroll, Sepolia, Sonic, Taiko, WorldChain.

We emphasize that this was a “delta” audit, considering only the new code implementing intent-based transactions, and potential issues that can arise by allowing actions to be freely chosen. The core functionality of the protocol was assumed to be correct. Note that any hypothetical issue in the core functionality would likely be magnified by the use of intents, since the actions can now be controlled by a malicious entity.

The audit’s main target is security threats, i.e., what the community understanding would likely call "hacking", rather than the regular use of the protocol. Functional correctness (i.e. issues in "regular use") is a secondary consideration. Typically it can only be covered if we are provided with unambiguous (i.e. full-detail) specifications of what is the expected, correct behavior. In terms of functional correctness, we often trusted the code’s calculations and interactions, in the absence of any other specification. Functional correctness relative to low-level calculations (including units, scaling and quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues affecting the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third-party attacker to drain a portion of the system’s or users’ funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third-party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: • User or system funds can be lost when third-party systems misbehave. • DoS, under specific conditions. • Part of the functionality becomes unusable due to a programming error.
	Examples:

LOW	• Breaking important system invariants but without apparent consequences. • Buggy functionality for trusted users where a workaround exists. • Security issues which may manifest when the system evolves.

Issue resolution includes “dismissed” or “acknowledged” but no action taken, by the client, or “resolved”, per the auditors.

CRITICAL SEVERITY

[No critical severity issues]

HIGH SEVERITY

[No high severity issues]

MEDIUM SEVERITY

[No medium severity issues]

LOW SEVERITY

• LOW | Resolved

L1 | Slippage check is optional in _checkSlippageAndTransfer

`SettlerBase::_checkSlippageAndTransfer` allows both `minAmountOut` and `buyToken` to be zero, in which case the whole slippage test and the transfer are skipped.

`SettlerBase::_checkSlippageAndTransfer():80`

```
function _checkSlippageAndTransfer(AllowedSlippage calldata slippage)
internal {
  (address recipient, IERC20 buyToken, uint256 minAmountOut) =
  (slippage.recipient, slippage.buyToken, slippage.minAmountOut);
  // dedaub: slippage check is optional
  if (minAmountOut != 0 || address(buyToken) != address(0)) {
     ...
  }
}
```

This is useful for meta-transaction swaps in which the slippage check is combined with the final action of the swap and the tokens are directly sent from the DEX to the taker.

However, for intent-based swaps, skipping the check is completely unsafe since the only witness provided by the user is the slippage check so without it anyone could easily steal the funds without returning anything to the intended recipient.

As a consequence, we recommend strengthening the slippage sanity checks by requiring **both** `minAmountOut` and `buyToken` to be non-zero.

OTHER / ADVISORY ISSUES

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

• ADVISORY | Info

A1 | Compiler bugs

The code is compiled with Solidity Version `0.8.25`, which at the time of writing has no known bugs.

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Security Suite.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our

foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.