Solid World

Smart Contract Security Assessment

May 12, 2023

ABSTRACT

Dedaub was commissioned to perform a security audit of several new components of the Solid World protocol. The core Solid World Protocol, which has been audited by Dedaub before, is a set of mechanisms (e.g., tokenization, pooling, and ecosystem reward distribution to stakeholders/liquidity providers) that aim to create and incentivize a liquid market for forward carbon offset agreements.

The most critical subjects covered by this audit are a liquidity deployer non-custodial contract, a KYC and blacklist solution that has been integrated in the existing contracts and the timelocking of certain protocol functionality.

The code and accompanying artifacts (e.g., test suite, documentation) have been developed with high professional standards. No security issues/threats that could lead to theft or loss of user funds were identified by the audit.

The main finding of the audit is a potential DOS attack to the LiquidityDeployer contract that would prevent it from deploying the deposited liquidity to Gamma's Hypervisor. Nevertheless, all other functions of the contract such as withdrawals would continue to work as expected.

SETTING & CAVEATS

The scope of the audit includes changes to the core contracts of the Solid World protocol and the new LiquidityDeployer contract. The test suite was consulted during the audit but was not part of it. The full list of audited files can be found below:

```
solid-world-dao-contracts
└── contracts
    ├── CollateralizedBasketTokenDeployer.sol
    ├── CollateralizedBasketToken.sol
    ├── ForwardContractBatchToken.sol
    ├── LiquidityDeployer.sol
```

```
├── libraries
│   ├── liquidity-deployer
│   │   ├── LiquidityDeployerDataTypes.sol
│   │   └── LiquidityDeployerMath.sol
│   ├── manager
│   │   ├── CollateralizationManager.sol
│   │   ├── DecollateralizationManager.sol
│   │   └── RegulatoryComplianceManager.sol
├── compliance
│   ├── Blacklist.sol
│   ├── KYCRegistry.sol
│   ├── RegulatoryCompliant.sol
│   └── VerificationRegistry.sol
├── SolidStaking.sol
├── SolidWorldManager.sol
├── SolidWorldManagerStorage.sol
└── TimelockController.sol
```

The audit report covers commit hash a0c22bf81b50a5218f98148e6285fd825f092602 of the at the time private repository solid-world-dao-contracts.

Two auditors worked on the codebase for 3 days.

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.
LOW	Examples: • 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:

ID	Description	STATUS
M1	LiquidityDeployer functionality can get DOSed	**WON'T FIX**

The LiquidityDeployer contract tracks the users that have deposited funds into it and certain functionality, i.e., function `deployLiquidity`, loops over them to compute the deployable liquidity and the LP tokens owed to each depositor. To be more exact, the array of the depositors is traversed 3 times during the execution of `deployLiquidity`, once in each of the `_computeAvailableLiquidity`, `_prepareDeployment` and `_prepareLPTokensOwed` functions.

LiquidityDeployer::deployLiquidity

```
function deployLiquidity() external nonReentrant {
    (
        lastAvailableLiquidity[config.token0],
        lastAvailableLiquidity[config.token1]
    ) = _computeAvailableLiquidity();

    // Dedaub: The called function traverses the depositors array.
    (
        lastTotalDeployedLiquidity[config.token0],
        lastTotalDeployedLiquidity[config.token1]
    ) = _computeTotalDeployableLiquidity();
```

```
    // Dedaub: The called function traverses the depositors array.
    _prepareDeployment();
    _allowUniProxyToSpendDeployableLiquidity();
    uint lpTokens = _depositToUniProxy();

    // Dedaub: The called function traverses the depositors array.
    _prepareLPTokensOwed(lpTokens);
    // Dedaub: Code omitted for brevity.
}
```

Thus, a malicious actor could create multiple depositor addresses that deposit tiny amounts to the LiquidityDeployer contract to make the execution of deployLiquidity infeasible due to exceeding the set gas constraints. The LiquidityDeployer expects users to deposit at least one of two tokens, a collateralized basket token (CBT) issued by Solid World or USDC. Even though the Solid World team will apply KYC to the owners of CBTs they cannot control USDC depositors and thus such an attack is entirely feasible. This DOS attack presents no obvious economic benefit to the attacker but could still be carried out to sabotage the proper operation of the system. On the bright side, such an attack will only block the liquidity deployment functionality of the contracts, while withdrawals of user funds will work as expected. However, a new LiquidityDeployer contract will have to be deployed to carry out the initial deployment of liquidity on Gamma's Hypervisor.

LOW SEVERITY:

ID	Description	STATUS
L1	Existing storage layout should be respected by future upgrades to the contract	**WON'T FIX**

The SolidWorldManager contract is the only core contract that is upgradeable (the other one is the VerificationRegistry). This means that care should be taken when changing its storage layout. More specifically, the order and types of storage variables (SolidWorldManagerStorage) should remain unchanged and any new variables should be defined after all the existing ones. If the previous storage layout is not respected, it can cause data inconsistencies and unexpected behavior when interacting with the upgraded contract. We are emphasizing this as we noticed that an addition of new storage variables to the SolidWorldManagerStorage did not follow the aforementioned best practices. Of course, the contracts have not been deployed or upgraded in a production setting yet but we feel that the developers should be well aware of this potential issue in the future.

CENTRALIZATION ISSUES:

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. Even if the owner is reputable, users are more likely to engage with a protocol that guarantees no catastrophic failure even in the case the owner gets hacked/compromised. We list issues of this kind below. (These issues should be considered in the context of usage/deployment, as they are not uncommon. Several high-profile, high-value protocols have significant centralization threats.)

[NO CENTRALISATION ISSUES]

OTHER / ADVISORY ISSUES:

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

ID	Description	STATUS
A1	SolidStaking's Stake event does not capture the msg.sender	**WON'T FIX**

The SolidStaking `Stake` event captures the `recipient` account but not the `msg.sender`, thus this piece of information is not recorded if the `recipient` is not also the `msg.sender`.

ID	Description	STATUS
A2	LiquidityDeployer::getTokenDepositors can be optimized to save gas	**WON'T FIX**

The function `LiquidityDeployer::getTokenDepositors` copies the depositors array from storage to memory by performing a loop over each element of the array instead of just returning the array.

LiquidityDeployer::getTokenDepositors

```
function getTokenDepositors() external view
    returns (address[] memory tokenDepositors)
{
    tokenDepositors = new address[](depositors.tokenDepositors.length);
    for (uint i; i < depositors.tokenDepositors.length; i++) {
        tokenDepositors[i] = depositors.tokenDepositors[i];
    }
}
```

By changing the code to:

```
function getTokenDepositors() external view
    returns (address[] memory tokenDepositors)
{
    return depositors.tokenDepositors;
}
```

the cost of calling `getTokenDepositors` is reduced by 33% and the deployment cost of the LiquidityDeployer is reduced by ~1.5%.

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 Watchdog.

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.