Audit Firm: Solidity Lab

Client Firm: DIVA Protocol

Prepared By: said017, WangChao, kodak_rome, Emmalien

Delivery Date: April 20th, 2023

Client Firm engaged Solidity Lab to review the security of its Smart Contract system. From March 14th 2023 to March 20th 2023, a team of 4 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Notice that the examined smart contracts are not resistant to external/internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

Project Overview

Project Name	**RaisinLabs**
Language	Solidity
Codebase	https://github.com/GuardianAudits/DivaAudit (https://github.com/GuardianAudits/DivaAudit)
Commit	5d0c7f6d854b65c2f723ac6bceea6bbd4edef37e (https://github.com/GuardianAudits/DivaAudit)

Delivery Date	**March 20th 2023**
Audit Methodology	Static Analysis, Manual Review

Vulnerability Level	**Total**	**Pending**	**Declined**	**Acknowledged**	**Partially Resolved**
Critical	0	0	0	0	0
High	2	2	0	0	0
Medium	2	2	0	0	0
Low	2	2	0	0	0

Audit Scope & Methodology

Scope

ID	**File**	**Checksum**
A	diva-contracts/contracts/*	-
B	oracles/contracts/*	-

Methodology

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by community auditors.

Vulnerability Classifications

Vulnerability Level	Classification
Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
Low	Minor deviation from best practices.

Protocol Graph (Optional)

IMAGE

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Receiver of settlement fee can be wrong in certain condition if fallback data provider executing `setFinalReferenceValue()`	Logic Error	HIGH	Pending
H-02	Receiver of treasury fee can be wrong in certain condition if remove liquidity function is executed	Logic Error	HIGH	Pending
M-01	Offer Taker using EIP712 can't remove his liquidity via valid `fillOfferRemoveLiquidity()` in certain case with ERC20 collateral with blacklist	DOS	MEDIUM	Pending
M-02	Diva new owner can effectively have access to `DivaDevelopmentFund withdraw()` and `withdrawDirectDeposit()` right after elected.	Centralization Risk	MEDIUM	Pending
L-01	unpauseReturnCollateral() will extend pause delay time even when it already unpaused	Centralization risk	LOW	Pending
L-02	Griefer can challenge final reference value and prolonged the settlement process	DOS	LOW	Pending

High

H-01 Receiver of settlement fee can be wrong in certain condition if fallback data provider executing `setFinalReferenceValue()`

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/GovernanceFacet.sol#L165-L169 (https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/GovernanceFacet.sol#L165-L169)

Description:

`updateFallbackDataProvider()` is called by owner to change `gs.fallbackDataProvider`, and valid to execute 60 days after this function called.

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/SettlementFacet.sol#L480-L490
(https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/SettlementFacet.sol#L480-L490)

```
        uint256 _startTimeNewFallbackDataProvider = block.timestamp

        // Store start time and new fallback data provider
        gs.startTimeFallbackDataProvider = _startTimeNewFallbackDat
        gs.fallbackDataProvider = _newFallbackDataProvider;
```

Inside `LibDIVA` library, `_getCurrentFallbackDataProvider()` is called to check if the protocol should use the `gs.fallbackDataProvider` only if current `block.timestamp` passed the `_startTimeNewFallbackDataProvider` (already more than 60 days) :

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibDIVA.sol#L918-L927
(https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibDIVA.sol#L918-L927)

```
    function _getCurrentFallbackDataProvider(
        LibDIVAStorage.GovernanceStorage storage _gs
    ) internal view returns (address) {
        // Return the new fallback data provider if `block.timestam
        // the activation time, else return the current fallback da
        return
            block.timestamp < _gs.startTimeFallbackDataProvider
                ? _gs.previousFallbackDataProvider
                : _gs.fallbackDataProvider;
    }
```

However, while `_setFinalReferenceValue()` will check if the caller is the current data provider by using `LibDIVA._getCurrentFallbackDataProvider(_gs)`, but setting the fee receiver with direct `_gs.fallbackDataProvider` inside `_confirmFinalReferenceValue()` call.

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/SettlementFacet.sol#L480-L490
(https://github.com/GuardianAudits/DivaAudit/blob/main/diva-

contracts/contracts/facets/SettlementFacet.sol#L480-L490)

```
                // Check that the `msg.sender` is the fallback data
                if (msg.sender != LibDIVA._getCurrentFallbackDataPr
                    revert NotFallbackDataProvider();

                _confirmFinalReferenceValue(
                    _poolId,
                    _pool,
                    _treasury,
                    _gs.fallbackDataProvider, // @audit - this dire
                    _finalReferenceValue,
                    _gs
                );
```

This can cause issue in scenario where the fallback data provider is changed by calling `updateFallbackDataProvider()` . while the new data provider can't send `setFinalReferenceValue()` within 60 days duration, the new fallback data provider will get the settlement fee immediately (not after 60 days), even though the one execute is the previous fallback data provider.

Recommendation:

Update the parameter inside `_confirmFinalReferenceValue()` using value from `LibDIVA._getCurrentFallbackDataProvider(_gs)` check :

```
                // Check that the `msg.sender` is the fallback data
                if (msg.sender != LibDIVA._getCurrentFallbackDataPr
                    revert NotFallbackDataProvider();

                _confirmFinalReferenceValue(
                    _poolId,
                    _pool,
                    _treasury,
                    LibDIVA._getCurrentFallbackDataProvider(_gs), /
                    _finalReferenceValue,
                    _gs
                );
```

Resolution:

H-02 Receiver of treasury fee can be wrong in certain condition if remove liquidity function is executed

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibDIVA.sol#L780-L785 (https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibDIVA.sol#L780-L785)

Description:

`updateTreasury()` is called by owner to change `gs.treasury`, but valid to use this update treasury 2 days after this function called.

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/GovernanceFacet.sol#L196-L204 (https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/GovernanceFacet.sol#L196-L204)

```
1          // Store current treasury address in `previousTreasury` var
2          gs.previousTreasury = gs.treasury;
3
4          // Set time at which the new treasury address will become a
5          uint256 _startTimeNewTreasury = block.timestamp + 2 days;
6
7          // Store start time and new treasury address
8          gs.startTimeTreasury = _startTimeNewTreasury;
9          gs.treasury = _newTreasury;
```

Inside `LibDIVA` library, `_getCurrentTreasury()` is used to get the valid treasury by checking the `gs.startTimeTreasury` against current `block.timestamp` :

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibDIVA.sol#L929-L940 (https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibDIVA.sol#L929-L940)

```
    function _getCurrentTreasury(LibDIVAStorage.GovernanceStorage s
        internal
        view
        returns (address)
    {
        // Return the new treasury address if `block.timestamp` is
        // the activation time, else return the current treasury ad
        return
            block.timestamp < _gs.startTimeTreasury
                ? _gs.previousTreasury
                : _gs.treasury;
    }
```

However, when remove liquidity is called, either from `LiquidityFacet` functions or from `EIP712RemoveFacet` functions, it will eventually call `LibDIVA` 's `_removeLiquidityLib()` function and treasure fee allocated directly with `LibDIVAStorage._governanceStorage().treasury` :

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibDIVA.sol#L780-L785 (https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibDIVA.sol#L780-L785)

```
        _allocateFeeClaim(
            _removeLiquidityParams.poolId,
            _pool,
            LibDIVAStorage._governanceStorage().treasury, // @audit
            _protocolFee
        );
```

This will cause the new treasury get the fee even before the 2 days delays period.

Recommendation:

Use the valid treasury inside `_removeLiquidityLib()` :

```
        _allocateFeeClaim(
            _removeLiquidityParams.poolId,
            _pool,
            _getCurrentTreasury(LibDIVAStorage._governanceStorage()
            _protocolFee
        );
```

Resolution:

Medium

M-01 Offer Taker using EIP712 can't remove his liquidity via valid `fillOfferRemoveLiquidity()` in certain case with ERC20 collateral with blacklist

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibEIP712.sol#L859-L863 (https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibEIP712.sol#L859-L863)

Description:

When maker create an EIP712 signature of `OfferRemoveLiquidity`, taker should be able to remove his liquidity via calling `fillOfferRemoveLiquidity()`. However, in certain case, if used collateral have blacklist mechanism and the maker is blacklisted after the taker take the offer, he can't execute the `fillOfferRemoveLiquidity()`, because the transfer collateral to the maker will fail :

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibEIP712.sol#L859-L863 (https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/libraries/LibEIP712.sol#L859-L863)

```
1          LibDIVA._returnCollateral(
2              _pool,
3              _offerRemoveLiquidity.maker,
4              _collateralAmountRemovedNetMaker
5          );
```

Consider this scenario, taker create the offer because he see that the maker also have EIP712 signature of `OfferRemoveLiquidity` in case the taker want to remove his liquidity from the Offer.

But the taker can't remove liquidity via EIP712 signature of `OfferRemoveLiquidity` now since the maker is blacklisted from the ERC20 collateral.

Recommendation:

Consider have separate state to track the maker and taker EIP712 collateral amount and implement pull over push method.

Resolution:

M-02 Diva new owner can effectively have access to `DivaDevelopmentFund withdraw()` and `withdrawDirectDeposit()` right after elected.

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/DIVADevelopmentFund.sol

(https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/DIVADevelopmentFund.sol)

Description:

While Diva's new elected owner can make a change of protocol parameters (such as fee, treasuery, fallback data provider) and only take effect after certain time, the new elected owner can have access to `withdraw()` and `withdrawDirectDeposit()` of `DivaDevelopmentFund` right after elected.

This can potentially be issue if some malicious owner is elected, and there is no way to prevent it from accessing the funds.

Recommendation:

Consider to have some delay after the new elected owner to have access of `DivaDevelopmentFund withdraw()` and `withdrawDirectDeposit()`, Also set some delay of every `withdraw()` and `withdrawDirectDeposit()` call to have some safety measure.

Resolution:

Low

L-01 unpauseReturnCollateral() will extend pause delay time even when it already unpaused

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/GovernanceFacet.sol#L234

(https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/GovernanceFacet.sol#L234)

Description:

`unpauseReturnCollateral()` function called to set `gs.pauseReturnCollateralUntil` with `block.timestamp` and unpause protocol functionality that depend of this check. However, calling `unpauseReturnCollateral()` while the protocol is already unpaused can extend the pause `pauseReturnCollateral()` function time check to 2 days.

Which undesirable and could potentially make the `pauseReturnCollateral()` function unavailable when it is needed.

Recommendation:

Consider make sure the protocol is in unpause state before updating the `gs.pauseReturnCollateralUntil` :

```
    function unpauseReturnCollateral() external override onlyOwner
        // Get reference to relevant storage slot
        LibDIVAStorage.GovernanceStorage storage gs = LibDIVAStorag
            ._governanceStorage();

        // check first if the contract is paused, otherwise don't u
        if (gs.pauseReturnCollateralUntil > block.timestamp) {
            gs.pauseReturnCollateralUntil = block.timestamp;
        }

        // Log the updated `pauseReturnCollateralUntil` timestamp
        emit ReturnCollateralUnpaused(
            msg.sender,
            gs.pauseReturnCollateralUntil
        );
    }
```

Resolution:

L-02 Griefer can challenge final reference value and prolonged the settlement process.

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/SettlementFacet.sol#L365-L368 (https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/SettlementFacet.sol#L365-L368)

Description:

When the data provider submit final reference value and challenge is allowed via `setFinalReferenceValue()` , it can be challenged by calling as long as still in challenge period time by calling `challengeFinalReferenceValue()` . However, the

required value to successfully call `challengeFinalReferenceValue()` is very low (only require to have non zero balance of short and long token).

https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/SettlementFacet.sol#L365-L368 (https://github.com/GuardianAudits/DivaAudit/blob/main/diva-contracts/contracts/facets/SettlementFacet.sol#L365-L368)

```
        if (
            IPositionToken(_pool.shortToken).balanceOf(msg.sender)
            IPositionToken(_pool.longToken).balanceOf(msg.sender) =
        ) revert NoPositionTokens(); // the value requirement is to
```

Griefer can mint dust amount of liquidity, since there is no minimum collateral check, or buy dust amount of long and short token, then initiate `challengeFinalReferenceValue()` to make the position challenged state.

Although the impact is minimal since the data provider only need to resubmit final reference value again with the same value or set challenge to false, but still the extra step needed from the usual.

Recommendation:

Consider to add non zero minimal long and short token owned to initiate `challengeFinalReferenceValue()` .

```
        if (
            IPositionToken(_pool.shortToken).balanceOf(msg.sender)
            IPositionToken(_pool.longToken).balanceOf(msg.sender) =
        ) revert NoPositionTokens();
```

Resolution:

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Solidity Lab to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk.
Solidity Lab’s position is that each company and individual are responsible for their own due diligence and continuous security. Solidity Lab’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Solidity Lab is subject to dependencies and under continuing
development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results.
The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Solidity Lab does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About

Solidity Lab is a community of practicing auditors guided by Guardian Audits.

To learn more, visit https://lab.guardianaudits.com (https://lab.guardianaudits.com)

To view the Solidity Lab audit portfolio, visit https://github.com/GuardianAudits/LabAudits![(https://github.com/GuardianAudits/LabAudits!%5B)]