MIDGAR

SMART CONTRACT SECURITY AUDIT OF

1DFX

TABLE OF CONTENTS

Project Summary

Security Firm: Midgar Pte. Ltd.

Prepared By: VanGrim, EVDoc

Client Firm: Liquid Lab Company Ltd

Final Report Date: 1 October 2024

Liquid Lab Company Ltd engaged Midgar to review the security of its smart contracts related to the 1DFX platform. From the **8th of July to the 11th of August**, a team of two (2) auditors reviewed the source code in scope. All findings have been recorded in the following report.

Please refer to the complete audit report below for a detailed understanding of risk severity, source code vulnerability, and potential attack vectors.

Project Name	1DFX
Language	Solidity
Codebase	https://git.liquid-lab.io/1dfx/1dfx-ddex-contracts/ https://git.liquid-lab.io/1dfx/1dfx-core-contracts/
Commit	Initial: 48ed2c6a2767cfbf5ed9c192632d140dbdeb4b8a 4c67dccf76af7b91fb8d0713f7b959bc702bf25e Final: e1b5a491fd6d001044db2b44b3405ea448d62a68 22d8cb8cb0eaf49c004dc29c876d784ceae4d5b6
Audit Methodology	Static Analysis, Manual Review, Fuzz Testing, Invariant Testing
Review Period	8 July - 11 August 2024
Resolved	1 October 2024

Project Overview

1DFX is a cutting-edge decentralized finance (DeFi) protocol designed to revolutionize the financial ecosystem by leveraging blockchain technology and smart contracts. It addresses key challenges in traditional finance, such as lack of transparency, inefficiency, and centralized control, by providing a robust, transparent, and efficient platform for financial transactions and asset management.

1DFX offers comprehensive deliverable management, including assets, referential assets, index assets, and perpetual futures. The protocol ensures accurate classification, tracking, and management of each deliverable, providing users with a reliable financial ecosystem. It incorporates a rule-based framework for managing deliverables, ensuring all transactions adhere to predefined rules, enhancing security and integrity.

Audit Scope

1DFX-Core-Contracts

ID	File	SHA-1 Checksum
ACE	AccessControlEnumerableFacet.sol	**c1f5d4d081f733f113a1757d76ec1e3e69fb52e6**
DCF	DiamondCutFacet.sol	**d28e5bad38c6a0e7bf3ebdf129f31dce94862d62**
DLF	DiamondLoupeFacet.sol	**00a46fc2bc32e59f9934467d49209d51c7c7c851**
PAF	PausableFacet.sol	**2258924fafcfd98274d4d9673c32df60a93f509c**
TLF	TimeLockFacet.sol	**2b7aac323c52d0b4c50a1a8809251715ccf7de54**
LCE	LibAccessControlEnumerableFacet.sol	**7d51210dec3cef7099ff59fea44aeef967b02e3e**
LID	LibDiamond.sol	**2b7ca1a32e29920326431f067248dfc05ac6689e**
LTL	LibTimeLock.sol	**193aea7fc53f6450d5c8d7a72c31f6035769f81a**
ODFXT	OneDfxToken.sol	**0203c3c677cf5b0138fa00d6748b7ae2a6e5b1f5**

1DFX-core-contracts

ID	File	SHA-1 Checksum
ONS	OnlySelf.sol	**e79e5c209287c72b1acb361814f0b222e1a1d88b**
PAU	Pausable.sol	**222ba7036966562aa5b24cc205f03e85df384529**
REG	ReentrancyGuard.sol	**2f4f18093233a7eb8b800baec2f9ba9fa620165c**
ODI	OneDfxInitializer.sol	**e27e48a10f463cc68fc30b3f84437a705b88d511**
BIT	Bits.sol	**8d5d21d7ca53b99edbe45c33c298797f88cc4fea**
CON	Constants.sol	**b22a9c677e58f12a035eb547a90a366c0cd20f78**
STU	StringUtils.sol	**0555822b342307e4a1b3d398741fbbf8f13a1d7f**
UNC	UncheckedCounter.sol	**aadadd48ab4dec6df0bf1297fb31aa5716585fa6**
ODFX	OneDfx.sol	**f3eb784cbfff979d7c6e6c568216beedce24b146**

Audit Scope

1DFX-ddex-contracts

ID	**File**	**SHA-1 Checksum**
CPF	ChainlinkPriceFacet.sol	**924f22775e78eabb7d9c1ca5a22d260c57f5e6f6**
PFF	PriceFacadeFacet.sol	**86be2f157917a0f981ae9e4beddfe05efd87f8ff**
RDDRF	ReferenceDataDeliverableRulesFacet.sol	**c8a2c9a8f0e064435561bd0bfc2df1f072cd70bb**
RDDF	ReferenceDataDeliverablesFacet.sol	**78e3bbf9624ccdf8916f7a9fccddd4c941ad4c83**
RDLP	ReferenceDataLiquidityPoolFacet.sol	**08919039dd5d34d3e5b0f375951ead7ea69bc336**
RDLF	ReferenceDataLookupFacet.sol	**5b1be785e4a9c394fff0edb8d22bd0e231dfd00b**
TLP	TranchedLiquidityPoolFacet.sol	**b70514d16081e0192ffdd71aabed05f5ef037c88**
LCP	LibChainlinkPrice.sol	**4898c4f1b2a568d11fafc1ff86e7eb93b4772323**

1DFX-ddex-contracts

ID	File	SHA-1 Checksum
LDT	LibDeliverableTransfer.sol	**f7a3c21481b381b4796cf939341ee23efb4d7c44**
LPF	LibPriceFacade.sol	**acbed9b7f8b1d631a5d22cb279fb325580267bc3**
LRDDR	LibReferenceDataDeliverableRules.sol	**6c393807cf7a139203975204488cc72125df6d0e**
LRDD	LibReferenceDataDeliverables.sol	**82a2d0c6a66d8fc3548bb38eccc2fd8d962466c5**
LRDLP	LibReferenceDataLiquidityPool.sol	**b50692d9d4e9930fe987dc94d8f461584db158fd**
LRDL	LibReferenceDataLookup.sol	**55a29e1d693f51f0cd412426add6b884e505c997**
LTLP	LibTranchedLiquidityPool.sol	**7653f631c19f27e13f23ebec2670e1044862e4e4**
LTLPS	LibTranchedLiquidityPoolStorage.sol	**443b56c62ef3980a014d985b21bc1d7e3ce70f53**

1DFX-ddex-contracts

ID	File	SHA-1 Checksum
ONC	OnchainConstants.sol	**e2e4ff107c829e5e2d48932232a4b9db8baeb04e**
GLOBAL	-	**-**

Vulnerability Summary

Vulnerability Level	Total	Acknowledged	Resolved
🔴 Critical	1	0	1
🟠 High	4	3	1
🟡 Medium	9	3	6
🟢 Low	8	1	7
🔵 Informational	2	2	0

Findings & Resolutions

ID	Title	Severity	Status
LRDDR-1	Function reverts when setting new liquidity pool target weight rules due to array modification	🔴 **Critical**	**Resolved**
GLOBAL-1	Future upgrades may be difficult or impossible	🟠 **High**	**Resolved**
LTLP-1	Math not rounding in protocol favor	🟠 **High**	**Acknowledged**
LTLP-2	LPs could exploit the phase transition to make risk-free profits	🟠 **High**	**Acknowledged**
LTLP-3	Depositing token X and withdrawing token Y should not be allowed	🟠 **High**	**Acknowledged**
LRDLP-1	A `basePrice` of 0 should not be allowed to prevent any division by zero	🟡 **Medium**	**Resolved**
LTLPS-1	Incorrect configuration of `TrancheValueRule` or `TrancheDepositRule` could DoS deposits	🟡 **Medium**	**Resolved**
LTLPS-2	`updateTranche()` is vulnerable to front-running attacks	🟡 **Medium**	**Acknowledged**
LTLPS-3	Updating a tranche's deliverables to assets never deposited would prevent any LP withdrawals	🟡 **Medium**	**Resolved**
LRDD-1	Incorrect Lpt address when setting the `Deliverable` could lead to a loss of funds for LPs	🟡 **Medium**	**Resolved**

Findings & Resolutions

ID	Title	Severity	Status
LRDDR-2	Impossible to set liquidity pool target weight rules after a certain amount of deliverables	● **Medium**	**Acknowledged**
LRDDR-3	Possible to overwrite `positionFeeRule` due to incorrect validation	● **Medium**	**Resolved**
RDDF-1	Missing validations of added/removed assets risk leading to incorrect `feeBps`	● **Medium**	**Resolved**
LPF-1	Lack of price freshness check in `LibPriceFacade.sol` allows a stale price to be used	● **Medium**	**Acknowledged**
ODFX-1	Missing input validation checks on contract initialize/constructor	● **Low**	**Resolved**
LTLP-4	`withdrawFromTranche()` triggers a `TrancheValueBelowMinimumAllowed` error incorrectly	● **Low**	**Resolved**
LTLP-5	Allowing `minAcceptableAssetQuantity` to 0 could inflate the price of Lpt starting from phase 2	● **Low**	**Resolved**
LRDD-2	When an index deliverable is created, `baseDeliverable` must be different from `quoteDeliverable`	● **Low**	**Resolved**
RDLPF-1	Lack of method to update/get `TrancheDefinition`	● **Low**	**Resolved**
LRDDR-4	Redundant `deliverableId` checks	● **Low**	**Acknowledged**

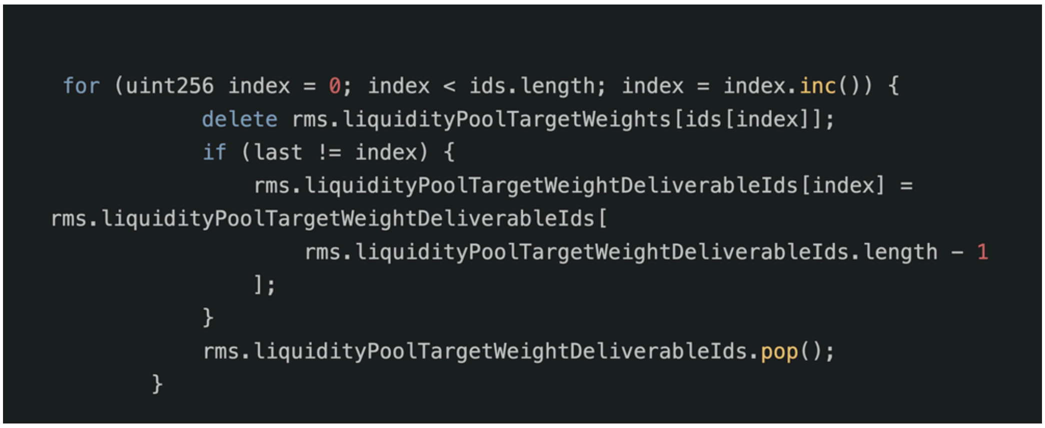
Findings & Resolutions

ID	**Title**	**Severity**	**Status**
LRDDR-5	Mistakenly returning the `b.breachedMaxExposureOpenFeeBps` twice	🟢 **Low**	**Resolved**
GLOBAL-2	Typo suggestions	🟢 **Low**	**Resolved**
INFO-1	Make sure to call `setRoleAdmin` first when adding the `AccessControlEnumerableFacet`	🔵 **Informational**	**Acknowledged**
INFO-2	Wrong decimals for ‘ETH-T’ mock ERC20	🔵 **Informational**	**Acknowledged**

LRDDR-1	**Function reverts when setting new liquidity pool target weight rules due to array modification**		
Asset	**LibReferenceDataDeliverableRules.sol:L605-617**		
Status	**Resolved**		
Rating	**Severity: Critical**	**Impact: High**	**Likelihood: High**

Description (POC)

The protocol sets its liquidity pool target weight rules by calling the function `setLiquidityPoolTargetWeightRule()` with an array containing deliverable IDs and target weight basis points (bps). However, when trying to adjust an already existing liquidity pool target weight rule (due to newly added or removed assets, for example), the function will always revert due to an array out-of-bounds issue. The revert happens in the following code snippet inside the `removeLiquidityPoolTargetWeightRule()` function:

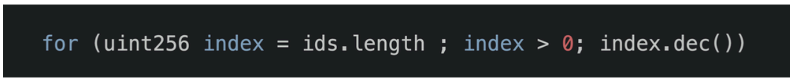


```
    for (uint256 index = 0; index < ids.length; index = index.inc()) {
            delete rms.liquidityPoolTargetWeights[ids[index]];
            if (last != index) {
                rms.liquidityPoolTargetWeightDeliverableIds[index] =
  rms.liquidityPoolTargetWeightDeliverableIds[
                    rms.liquidityPoolTargetWeightDeliverableIds.length - 1
                ];
            }
            rms.liquidityPoolTargetWeightDeliverableIds.pop();
        }
```

Due to removing elements from the end of the `rms.liquidityPoolTargetWeightDeliverableIds` array and trying to access the indices incrementally in the loop (starting with index 0), the function will revert halfway through the loop when it tries to access an index that has been removed. In summary, this means that it will never be possible to change an already existing target weight rule composition.

Recommendation

Consider changing the for-loop to iterate in reverse to avoid an array out-of-bound issue.



```
  for (uint256 index = ids.length ; index > 0; index.dec())
```

Resolution

This issue is resolved as of commit 73e319bff7cba476366bdfd3e7d71cdce7ad4849

GLOBAL-1	Future upgrades may be difficult or impossible		
Asset	1dfx-ddex-contracts		
Status	Resolved		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The project uses nested structures to store data, which may complicate or make future upgrades impossible. In extreme cases, upgrades could lead to data inconsistency and improper system operation.

As stated in this article by Nick Mudge, the creator of EIP-2535 Diamonds:

```Do not put structs directly in structs unless you don't plan on ever adding more state variables to the inner structs. You won't be able to add new state variables to inner structs in upgrades without overwriting existing state variables.```

Recommendation

To enable safe extension of inner structures in future upgrades, avoid directly nesting structures. Instead, use mappings, which allow extending structures without the risk of overwriting existing state variables.

Alternatively, avoiding the addition of variables to structs during upgrades would prevent such complications.

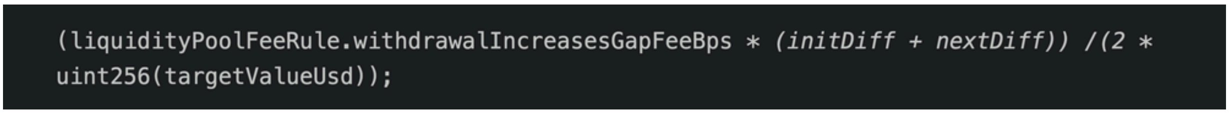
Resolution

This issue is resolved as of commit 59761a154800c1c00189ce060c6e56216d4e4de3

LTLP-1	**Math not rounding in protocol favor**		
Asset	**LibTranchedLiquidityPool.sol: L537-540**		
Status	**Acknowledged**		
Rating	**Severity: High**	**Impact: Medium**	**Likelihood: High**

Description

The `feeInc` calculation in `_getFeeBps` function uses rounding in favor of the user instead of the protocol, giving away a small amount of fees that can accumulate over time.
`_getFeeBps()` is used in `LibTranchedLiquidityPool` code to know how much fees will be charged when a user deposits or withdraws.
This function rounds down when calculating `feeInc` in case of `avgDiff < uint256(targetValueUsd)`, providing slightly less fees than expected:



```
(liquidityPoolFeeRule.withdrawalIncreasesGapFeeBps * (initDiff + nextDiff)) /(2 *
uint256(targetValueUsd));
```

Proof of Concept

Let's take the example of a pool with a single tranches and two assets: `USDC` and `USDT`, where 1 USDC = $1 and 1 USDT = $1. The `LiquidityPoolTargetWeight` for `USDC` and `USDT` is 50% each. Imagine the pool is well-balanced with 1000 USDC and 1000 USDT in the tranche. Each target weight is achieved.

Let's consider that `withdrawalIncreasesGapFeeBps` is set to 15 and `depositIncreasesGapFeeBps` is also set to 15.

Bob deposits 1000 USDC into the tranche 1. Here are the various parameters:

- targetValueUsd` = 1000 USD
- `nextValueUsd` = 2000 USD
- `initDiff` = 0 USD
- `nextDiff` = 1000 USD
- `avgDiff` = 500
- `feeInc` = (15 * (0 + 500)) / 2000 = 3.75 => rounded down to 3
- value returned from `_getFeeBps()` = 15 + 3 = 18 instead of 19 if rounded up

Recommendation

Use a math library like `FixedPointMathLib` from solady ("@solady/utils/FixedPointMathLib.sol")

Resolution

The functionality with respect to defining and then incentivising behaviour towards the target pool composition will be completely revisited in phase 2. Fees will not be charged in phase 1, so it has been decided that this does not require to be addressed.

Acknowledged and closed

LTLP-2	**LPs could exploit the phase transition to make risk-free profits**		
Asset	**LibTranchedLiquidityPool.sol L130-153**		
Status	**Acknowledged**		
Rating	**Severity: High**	**Impact: Medium**	**Likelihood: High**

Description

When a deposit is made in a tranche, the number of LP tokens to mint is calculated using the following formula: ```(assetValueInUsdAfterFee * 1e8) / lptPrice. ```

The `lptPrice` is returned by the internal function `_getLptPrice()`, which returns the `basePrice` during phase 1.
In phase 2, the exchange rate will be determined based on `trancheValueUsd` and the `totalSupply`.

During phase 1, the `basePrice` will be 1, so the amount of LP tokens to mint will depend solely on the value of the deposited asset in USD. However, in phase 2, this amount will be determined by both the value of the deposited asset and the `lptPrice`, which will no longer be equal to 1. The `lptPrice` will be determined by dividing the `trancheValueUsd` by the `totalSupply`.
Since fees will have accumulated during phase 1, the `trancheValueUsd` will be greater than the `totalSupply`
This means that LPs could deposit just before the transition from phase 1 to phase 2 and withdraw in phase 2 an amount greater than their initial deposit, giving them a risk-free profit.

When a withdrawal is made, the `assetValueToWithdrawInUsd` will be higher when transitioning to phase 2 due to the fees accumulated during phase 1, ensuring a profit for LPs who deposited in phase 1.

A stepwise jump in the `lptPrice` will encourage such risk-free strategies, leading to a decline in the LP token price at the beginning of phase 2

Recommendation

Consider pausing deposits and withdrawals before transitioning to phase 2. It would also be preferable not to disclose the exact timing of the pause. This should limit withdrawals at the beginning of phase 2, thereby reducing the decline in the LP token price

Resolution

With the changes to change from chainlink to a new Price Store implementation, the lptPrice will be calculated each time. The base price will only be used for the first deposit into the tranche.
Therefore, this issue will not be relevant

Acknowledged and closed

LTLP-3	Depositing token X and withdrawing token Y should not be allowed		
Asset	LibTranchedLiquidityPool.sol L229-233		
Status	Acknowledged		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description (POC)

If a deliverable asset loses a significant portion of its value, LPs who deposited this token will have an incentive to withdraw another `deliverableAsset` to recover the value of their deposit.

This practice could continue until either the minimum tranche value in USD is reached or the tranche contains insufficient asset value. Once either of these conditions is met, no LP will be able to withdraw from this tranche.

Recommendation

Consider creating an LPT for each `deliverableAsset` asset rather than creating an LPT for each `tranche` (since each tranche contains multiple deliverable assets with varying prices).
Another solution would be to store the deposited quantities by token for each LP in a mapping and only allow withdrawals of a token if the LP has previously deposited an equal or greater amount of that same token.

Resolution

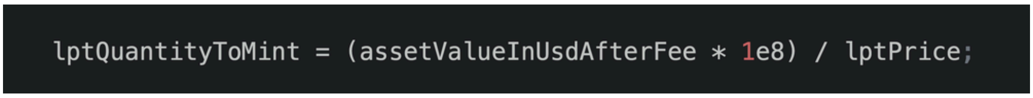
The functionality to allow a deposit of token X and to withdraw token Y is temporary functionality that will only be available in phase 1 where only a deposit and stake product is being offered. In phase 1, only tokens that are highly correlated will be accepted in the LP. The business acknowledges the risk of a temporary price decorrelation between the assets accepted. However, the capacity to take advantage of potential price decorrelations will be limited due to: tranche caps, maximum deposit limits, cooldown period for deposits, cooldown period for withdrawals, withdrawal limits for periods of time .In the next phase where pool composition will be strictly defined, a user will not be able to make a withdrawal to a specified asset. The pool will decide what assets the user receives based on the composition of the pool, the target weights, and the risk the pool carries for different assets.

Acknowledged and closed

LRDLP-1	A `basePrice` of 0 should not be allowed to prevent any division by zero		
Asset	LibReferenceDataLiquidityPool.sol		
Status	Resolved		
Rating	Severity: Medium	Impact: High	Likelihood: Low

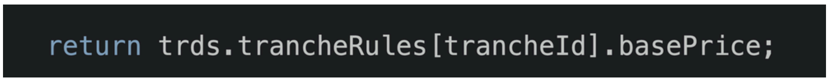
Description (POC)

When a tranche is added or updated, a base price of 0 should not be allowed. If the base price is 0, any call to the `depositIntoTranche()` function would result in a panic revert due to division by zero. `calculateLpTokensToMint()` determines the amount to mint by dividing `assetValueInUsdAfterFee` by `lptPrice`: ```lptQuantityToMint = (assetValueInUsdAfterFee * 1e8) / lptPrice; ```



```
lptQuantityToMint = (assetValueInUsdAfterFee * 1e8) / lptPrice;
```

`_getLptPrice()` determines the price during phase 1 based on the `basePrice`:



```
return trds.trancheRules[trancheId].basePrice;
```

Recommendation

Ensure that the `basePrice` is different from 0 when adding or updating a tranche

Resolution

This issue is resolved as of commit 71b1b695f464ac00716967366673449ba518ea58

LTLPS-1	**Incorrect configuration of `TrancheValueRule` or `TrancheDepositRule` could DoS deposits**		
Asset	**LibTranchedLiquidityPoolStorage.sol**		
Status	**Resolved**		
Rating	**Severity: Medium**	**Impact: High**	**Likelihood: Low**

Description (POC)

When the tranche value rules or deposit value rules are updated by the admin, it is allowed to provide values that could cause a Denial of Service (DoS) for the `depositIntoTranche()` function.

The transaction will revert during the internal call to the `_validateDeposit()` function with a `TrancheDepositAboveMaxValue()` custom error or a `TrancheDepositBelowMinValue()` custom error.

Setting `minTrancheValue` greater than `maxTrancheValue`, or `minDepositValue` greater `maxDepositValue` would prevent any deposits, regardless of the amount.

In the same way, setting `maxTrancheValue` less than `minDepositValue` would cause the same issue for deposits.

Recommendation

Consider verifying that `minTrancheValue < maxTrancheValue` and `minDepositValue < maxDepositValue` when `setTrancheValueRule()` and `setTrancheDepositRule()` are called.

Resolution

This issue is resolved as of commit 9c44d796c9539f78a6348e09b1257acad45607ad and ee9bb8db7169cc690a8a707bc370570c723cf84450

LTLPS-2	`updateTranche()` is vulnerable to front-running attacks		
Asset	LibTranchedLiquidityPoolStorage L89-108		
Status	Acknowledged		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description (POC)

When a tranche is updated, such as modifying the `basePrice`, nothing prevents a liquidity provider (LP) from front-running the transaction by depositing if the `basePrice` is being increased. Front-running an admin's transaction that raises the `basePrice` would ensure an immediate profit for the front-runner.

Recommendation

Changes such as increasing the `basePrice` should only be made when deposits are paused. Consider implementing a `whenPaused()` modifier for the `updateTranche()` function to ensure that deposits and withdrawals are paused before modifying any price-related parameters.

Resolution

With the changes to change from chainlink to a new Price Store implementation, the lptPrice will be calculated each time. The base price will only be used for the first deposit into the tranche. Therefore, this issue will not be relevant.

Acknowledged and closed.

LTLPS-3	Updating a tranche's deliverables to assets never deposited would prevent any LP withdrawals		
Asset	LibTranchedLiquidityPoolStorage L191-213		
Status	Resolved		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description (POC)

If the deliverables in the tranche participation rule are updated to tokens that have never been deposited in the tranche, LPs will no longer be able to withdraw. New LPs would need to deposit using the updated tokens for the existing LPs to be able to withdraw.

Recommendation

Ensure that the tranche definition rule is never updated with only deliverables that have never been deposited to prevent LPs from being unable to withdraw.
In any case, there must always remain an asset that has been used for deposit in the tranche. Confirm this by checking the `trancheTokenQuantities` mapping.

Resolution

This issue is resolved as of commit 75de2a76c5bc4e2581e693d8fb2618f774c310ed

LRDD-1	Incorrect Lpt address when setting the `Deliverable` could lead to a loss of funds for LPs		
Asset	LibReferenceDataDeliverables.sol L109-120		
Status	Resolved		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description (POC)

When an asset deliverable is added by calling the `addAssetDeliverable()` function with an `AssetDeliverable` as an argument, if the asset is a `COLLECTIVE_INVESTMENT_TOKEN`, there is no check in `_addDeliverable()` to confirm that it is a `OneDfxToken`.

If the token address in the `Deliverable` struct is not a `OneDfxToken`, and if this asset is set as an Lpt deliverable when creating a tranche, any LP depositing into the tranche will not have any tokens minted and will lose their deposit.

Recommendation

Consider adding a function called `supportsInterface()` to the `OneDfxToken` contract that verifies compatibility by checking for the presence of required methods. You could call this function when adding an asset deliverable as `COLLECTIVE_INVESTMENT_TOKEN` to ensure the token is a valid `OneDfxToken` that will be correctly minted upon deposit

Resolution

This issue is resolved as of commit 55a07c9d973497d3d590ec460766cccd707269e3

LRDDR-2	Impossible to set liquidity pool target weight rules after a certain amount of deliverables		
Asset	LibReferenceDataDeliverableRules.sol		
Status	Acknowledged		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The way to set the target weight for all deliverables at the moment is through the function `setLiquidityPoolTargetWeightRule()`. However, after a certain amount of assets it will be impossible to set the target weight rules due to the transaction running out of gas. As there's currently no other way to set the target weight rules for all deliverables, this essentially means that the protocol will not be able to adjust target weight rules.

Recommendation

Consider refactoring the code to reduce gas consumption. Simplify the logic of `_validateLiquidityPoolTargetWeights()` also consider using an Enumerable set or a mapping to avoid nesting for-loops when checking for duplicate ids.

Resolution

As we there are only 2 assets in phase 1 and we will not use target weights in phase 1, we will change this in phase 2.

Acknowledged and closed.

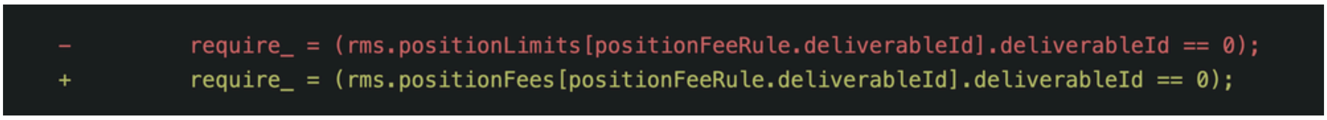
LRDDR-3	Possible to overwrite positionFeeRule due to incorrect validation		
Asset	LibReferenceDataDeliverableRules.sol L244		
Status	Resolved		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

In the `addPositionFeeRule()` function, we're checking the wrong mapping when validating if the mapping storage is empty or not. Instead of the `rms.positionFees` mapping the validation happens in the `rms.positionLimits`. This could potentially lead to a situation where the deliverable ID does not revert even if the position rule has been initialised already, and thus risks getting overwritten.

Recommendation

Consider replacing the following lines of code inside of the `addPositionFeeRule()` function



```diff
-        require_ = (rms.positionLimits[positionFeeRule.deliverableId].deliverableId == 0);
+        require_ = (rms.positionFees[positionFeeRule.deliverableId].deliverableId == 0);
```

Resolution

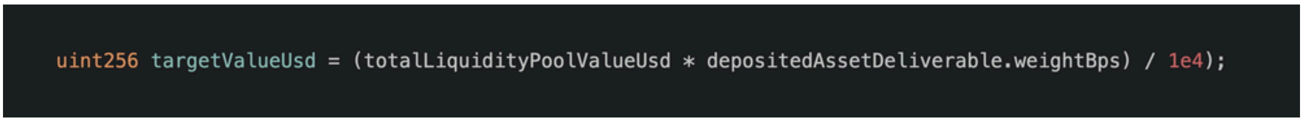
This issue is resolved as of commit e46971c1f574e5a4edb8d6a5c31fc6b5fa2a11a8

RDDF-1	Missing validations of added/removed assets risk leading to incorrect feeBps		
Asset	ReferenceDataDeliverablesFacet.sol		
Status	Resolved		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

When an asset is added or removed in the `ReferenceDataDeliverablesFacet` contract there are currently no validations happening to ensure that the newly added/removed asset are also adjusted for in the target weight rules for the liquidity pool.

Since the fee basis points are based on the target weights to set the fee structure, this means that the output from the function `getFeeBps()` will be incorrect (for example, the following line of code will return 0), meaning that the fee basis point will be fixed rather than dynamic:



```
uint256 targetValueUsd = (totalLiquidityPoolValueUsd * depositedAssetDeliverable.weightBps) / 1e4);
```

Recommendation

Consider calling the function `setLiquidityPoolTargetWeightRule()` when adding/removing asset.

Resolution

This issue is resolved as of commit 200d71cee0bf078137f8420b1aa95806cc533d97

LPF-1	Lack of price freshness check in LibPriceFacade.sol allows a stale price to be used		
Asset	LibPriceFacade.sol		
Status	Acknowledged		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

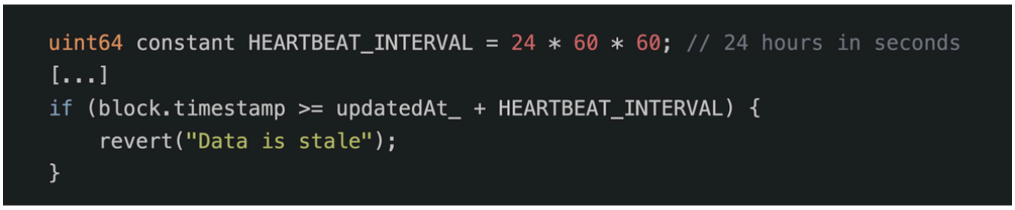
In the `getPriceFromCacheOrOracle()` function, callback prices for a given deliverable are stored in the `cachePrice` variable, and price information from Chainlink is stored in the `priceInfo` variable. The `getPriceFromChainlink()` function returns three variables: `price`, `decimals`, and `updatedAt`. The `tokenPrice` is set as the price from Chainlink if it is fresher than the cached price.

If `pfs.callbackPrices[deliverableId]` has not been initialized, the default values for `cachePrice.timestamp` and `cachePrice.price` will be 0. As a result, the 'else' block will be executed because `cachePrice.timestamp` (which is 0) will be less than `priceInfo.oracleUpdatedAt`.
In this case, there is no freshness check in the current implementation of `LibPriceFacade.sol::getPriceFromCacheOrOracle()`. This could lead to stale prices being used.

In addition, in situations where the cache price has been set but the oracle goes offline or the market value drops drastically for any given token - the cached price means that malicious users will still be able to utilise a previous cached token price and sell their "useless" tokens regardless of how stale the price is.

Recommendation

Implement an additional validation that reverts stale prices. When working with oracles, taking the heartbeat into consideration is considered best practice. This is a safety check to ensure that not all cached prices will/can be used indefinitely regardless of the actual status of the token. Optimally this should be implemented in the getPriceFromChainLink() function inside of LibChainLinkPrice.sol. For example:



```
uint64 constant HEARTBEAT_INTERVAL = 24 * 60 * 60; // 24 hours in seconds
[...]
if (block.timestamp >= updatedAt_ + HEARTBEAT_INTERVAL) {
    revert("Data is stale");
}
```

Resolution

This code will not be used due to the change from chainlink to a new Price Store implementation.

Acknowledged and closed.

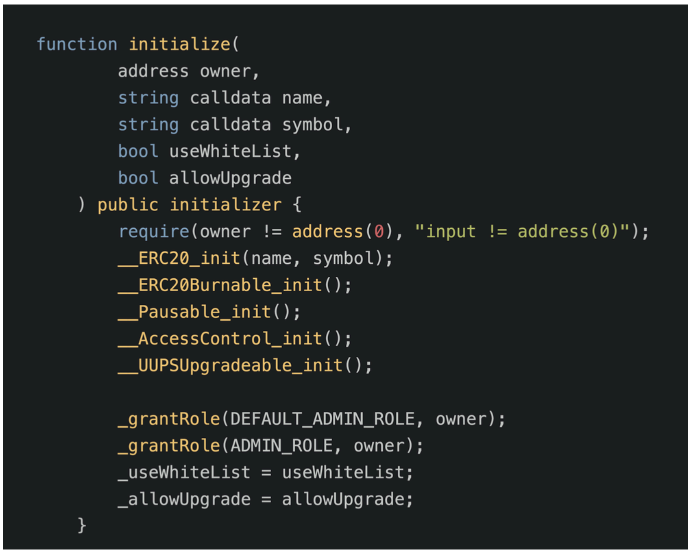
ODFX-1	Missing input validation checks on contract initialize/constructor		
Asset	OneDfxToken.sol & OneDfx.sol		
Status	Resolved		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Contract initialize/constructor input parameters should always be validated to prevent the creation/initialization of a contract in a wrong/inconsistent state.

Recommendation

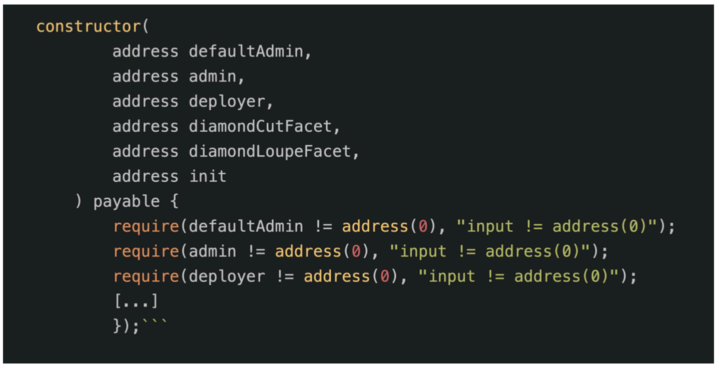
Consider implementing the following changes.
`OneDfxToken.sol`



```
function initialize(
        address owner,
        string calldata name,
        string calldata symbol,
        bool useWhiteList,
        bool allowUpgrade
    ) public initializer {
        require(owner != address(0), "input != address(0)");
        __ERC20_init(name, symbol);
        __ERC20Burnable_init();
        __Pausable_init();
        __AccessControl_init();
        __UUPSUpgradeable_init();

        _grantRole(DEFAULT_ADMIN_ROLE, owner);
        _grantRole(ADMIN_ROLE, owner);
        _useWhiteList = useWhiteList;
        _allowUpgrade = allowUpgrade;
    }
```

`OneDfx.sol`



```
constructor(
        address defaultAdmin,
        address admin,
        address deployer,
        address diamondCutFacet,
        address diamondLoupeFacet,
        address init
    ) payable {
        require(defaultAdmin != address(0), "input != address(0)");
        require(admin != address(0), "input != address(0)");
        require(deployer != address(0), "input != address(0)");
        [...]
        });```
```

Resolution

This issue is resolved as of commit 9a2e278fe6829dbd316f8a7318d5917d55ca1d87

LTLP-4	`withdrawFromTranche()` triggers a `TrancheValueBelowMinimumAllowed` error incorrectly		
Asset	LibTranchedLiquidityPool.sol L427		
Status	Resolved		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

When `calculateAssetQuantityToWithdraw()` is internally called from the `withdraw()` function, it performs some validation by calling `_validateWithdrawal()`.
This internal function checks that the withdrawal does not cause the tranche value in USD to fall below the `minTrancheValue` threshold.

However, `_validateWithdrawal()` is called with the argument `assetValueToWithdrawInUsd`. Since the fees remain in the tranche, this function should be called with `assetValueToWithdrawAfterFeeInUsd`.

User transactions should not revert as long as the required minimum values are met.

Recommendation

Call `_validateWithdrawal()` after calculating `assetValueToWithdrawAfterFeeInUsd` to validate the withdrawal with this amount.

Resolution

This issue is resolved as of commit 0a65d23c26cf1e1bb75a9d0e033aec2334ea9863

LTLP-5	**Allowing `minAcceptableAssetQuantity` to 0 could inflate the price of LPT starting from phase 2**		
Asset	**LibTranchedLiquidityPool.sol**		
Status	**Resolved**		
Rating	**Severity: Low**	**Impact: Low**	**Likelihood: Low**

Description (POC)

Allowing `minAcceptableAssetQuantity` to 0 burns LPTs without reducing the `trancheValueUsd`.

In phase 2, when the exchange rate is no longer fixed, this would lead to an artificial increase in the Lpt price. Multiple such withdrawals over time could accumulate, impacting new LPs but favoring the old ones.

Recommendation

Consider not allowing users to set `minAcceptableAssetQuantity` to 0 when calling `withdrawFromTranche()` function.

Resolution

This issue is resolved as of commit 0f3a4cbc4686f0a071714d03f19068b021720140

LRDD-2	When an index deliverable is created, `baseDeliverable` must be different from `quoteDeliverable`		
Asset	LibReferenceDataDeliverables.sol		
Status	Resolved		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

An index deliverable with the same `baseDeliverable` and `quoteDeliverable` would not be appropriate because the `baseDeliverable` must be quoted against another deliverable. For example, `EUR-USD-INDEX` or `BTC-USD-INDEX`.

The `addIndexDeliverable()` function from `LibReferenceDataDeliverables` internally calls `_checkIndexSpec()` and `_addDeliverable()` before referencing the index according to the `IndexDeliverable` struct provided as an argument. The `_checkIndexSpec()` function only verifies that the `baseDeliverableId` and `quoteDeliverableId` exist.

Therefore, it is possible to create a `BTC-BTC-INDEX`. Creating such an index should not be allowed.

Recommendation

Consider checking that `baseDeliverable` and `quoteDeliverable` are different within `_checkIndexSpec()` function.

Resolution

This issue is resolved as of commit 933f808da7643a4c7595de64233d85c53345af99

<table>
<tr><td><b>RDLPF-1</b></td><td colspan="3"><b>Lack of method to update/get `TrancheDefinition`</b></td></tr>
<tr><td><b>Asset</b></td><td colspan="3"><b>ReferenceDataLiquidityPoolFacet.sol</b></td></tr>
<tr><td><b>Status</b></td><td colspan="3"><b>Resolved</b></td></tr>
<tr><td><b>Rating</b></td><td><b>Severity: Low</b></td><td><b>Impact: Low</b></td><td><b>Likelihood: Low</b></td></tr>
</table>

Description

There are no methods to update or get the tranche definition in `ReferenceDataLiquidityPoolFacet.sol`. The only way to update or get a tranche definition is by calling the `updateTranche()` or `getTranche()` functions.

In the case of updating the tranche definition, only the `basePrice` is modifiable. To modify it, all tranche parameters need to be provided and `updateTranche()` must be called.

Recommendation

Consider implementing `updateTrancheDefinition()` and `getTrancheDefinition()` functions.

Resolution

This issue is resolved as of commit b0083dc81760dfdde497a68c92bfe678af40a1de

LRDDR-4	Redundant `deliverableId` checks		
Asset	LibReferenceDataDeliverableRules.sol		
Status	Acknowledged		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

There are a few redundant validations that checks that the `deliverableId != 0` within the following functions:

- addPositionLimitRule()
- addPositionFeeRule()
- addLiquidityPoolFeeRule()
- addLiquidityParticipationRule()
- addPerpetualFundingRule()

The above-mentioned functions are all called from the ReferenceDataDeliverableRulesFacet.sol that does a separate validation check through the function `checkDeliverableIdisValid()` where the deliverable ID is validated to be non-zero.

Recommendation

Consider removing the deliverableId != 0 checks within the above-mentioned functions.

Resolution

As there is no harm in the extra check and removing it will mean an update to some tests we will live with this issue.

Acknowledged and closed.

LRDDR-5	**Mistakenly returning the `b.breachedMaxExposureOpenFeeBps` twice**		
Asset	**LibReferenceDataDeliverableRules L321**		
Status	**Resolved**		
Rating	**Severity: Low**	**Impact: Low**	**Likelihood: Low**

Description

The `getPositionFeeRule()` is accidentally returning the same value, `b.breachesMaxExposureOpenFeeBps` twice.

Recommendation

It should return `breachesMaxExposureCloseFeeBps` instead.

Resolution

This issue is resolved as of commit 27c1882020e346453e53236d01e27f3067f12cc9

GLOBAL-2	Typo suggestions		
Asset	1dfx-ddex-contracts		
Status	Resolved		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

- In the `AssetType` enum within the `IReferenceDataDeliverables` interface, there is a typographical error. One of the elements is incorrectly named `STOP` instead of the correct `SPOT`. This typo could lead to confusion and potential issues when referencing this specific asset type.
- `_checkBlackoutPeriod()` has a parameter called `lastWithdrawalAt`. However, this parameter takes both the `lastDepositedAt` and `lastWithdrawalAt`. A more appropriate naming might be: `lastTransactedAt` to include both the timestamps for deposits and withdrawals.
- `confirmTriggerPrice()` has a variable called oracle price. However, it is a bit misleading since the fetched price could be from the cache. A better name would be `cacheOrOraclePrice` or use `_price` to distinguish between internal variables and external ones.

Resolution

This issue is resolved as of commit 979c363879ddb956da3726bcf07075fc11b056f8

INFO-1	**Make sure to call `setRoleAdmin` first when adding the `AccessControlEnumerableFacet`**		
Asset	**1dfx-core-contracts**		
Status	**Acknowledged**		
Rating	**Severity: Low**	**Impact: Low**	**Likelihood: Low**

Description

Since the `AccessControlEnumerableFacet.sol` will be added through the `DiamondCutFacet.sol` at a later stage it is important to note that the `setRoleAdmin()` needs to be called first when adding this facet. This is due to the many functions inside of the `AccessControlEnumerableFacet.sol` that relies on the admin being set. An omission of the above could lead to the access control not working properly (since the admin address would be 0).

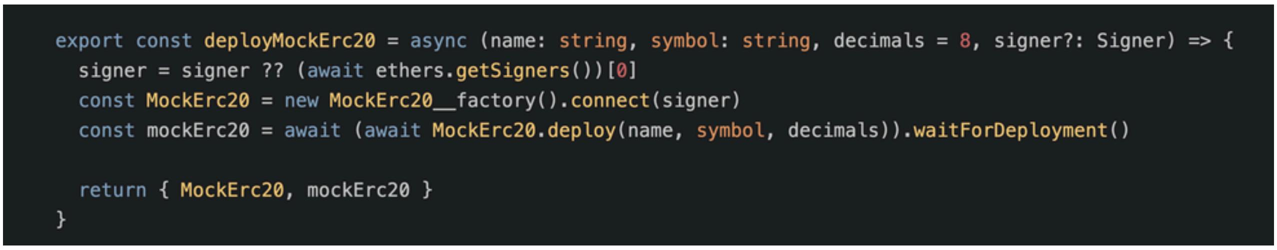
Resolution

Acknowledged and closed.

INFO-2	Wrong decimals for ‘ETH-T’ mock ERC20		
Asset	deployer.ts		
Status	Acknowledged		
Rating	Severity: Low	Impact: Low	Likelihood: Low

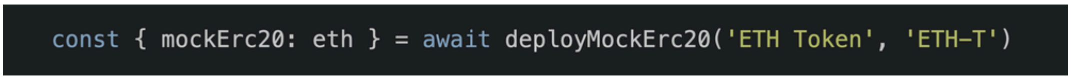
Description

In the deployment script, the ERC20 ‘ETH-T’ (WETH) is deployed with 8 decimals.



```
export const deployMockErc20 = async (name: string, symbol: string, decimals = 8, signer?: Signer) => {
  signer = signer ?? (await ethers.getSigners())[0]
  const MockErc20 = new MockErc20__factory().connect(signer)
  const mockErc20 = await (await MockErc20.deploy(name, symbol, decimals)).waitForDeployment()

  return { MockErc20, mockErc20 }
}
```



```
const { mockErc20: eth } = await deployMockErc20('ETH Token', 'ETH-T')
```

On `MAINNET`, the WETH token has 18 decimals. Ensure that the same amount of decimals are used to avoid confusion when testing.

Resolution

Acknowledged and closed.

Appendix

Test Suites: Invariants and Properties

During the security review, Midgar conducted an extensive series of tests on the client's codebase using the Echidna testing tool. The focus was on fuzzing and invariants testing to verify that the properties and invariants within the codebase remained robust under a variety of randomized conditions. This rigorous testing approach was designed to ensure the codebase's resilience and reliability by simulating a wide range of inputs and scenarios.

Over the course of the review, Echidna performed 1,000,000 test runs, systematically exploring potential vulnerabilities and ensuring that the codebase adhered to its specified behaviors.

ID	Description	Run Count	Passed
DEPOSIT_01	Deposit mints LPT to the sender	**1,000,000+**	✅
DEPOSIT_02	Deposit transfers tokens to the protocol	**1,000,000+**	✅
DEPOSIT_03	Deposit increases the tranche value	**1,000,000+**	✅
DEPOSIT_04	Deposit increases the total pool value	**1,000,000+**	✅
WITHDRAW_01	Withdraw deducts LPT from the sender	**1,000,000+**	✅
WITHDRAW_02	Withdraw removes tokens from the protocol	**1,000,000+**	✅
WITHDRAW_03	Withdraw decreases the tranche value	**1,000,000+**	✅
WITHDRAW_04	Withdraw decreases the total pool value	**1,000,000+**	✅
TOKENS_01	Total supply of LPT is equal to the sum of balances of all users	**1,000,000+**	✅

ID	Description	Run Count	Passed
FEES_01	After a deposit, asset sent in USD should be greater than the Lpt minted in USD	**1,000,000+**	✅
FEES_02	After a withdrawal, Lpt burnt in USD should be greater than the amount withdrawn in USD	**1,000,000+**	✅
POOL_01	If the price of assets increase, the total pool value in USD should increase	**1,000,000+**	✅
POOL_02	If the price of assets decrease, the total pool value in USD should decrease	**1,000,000+**	✅
POOL_03	The total pool value in USD should be the sum of the USD amounts of the protocol's balances	**1,000,000+**	✅
DOS	Denial of Service	**1,000,000+**	✅

Vulnerability Classification

The risk matrix below has been used for rating the vulnerabilities in this report. The full details of the interpretation of the below can be seen here.

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

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 aspiring auditors.

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 Midgar 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. Midgar’s position is that each company and individual are responsible for their own due diligence and continuous security. Midgar’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 Midgar are subject to dependencies and are 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 to internal/external exploits. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Midgar does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Midgar

Midgar is a team of security reviewers passionate about delivering comprehensive web3 security reviews and audits. In the intricate landscape of web3, maintaining robust security is paramount to ensure platform reliability and user trust. Our meticulous approach identifies and mitigates vulnerabilities, safeguarding your digital assets and operations. With an ever-evolving digital space, continuous security oversight becomes not just a recommendation but a necessity. By choosing Midgar, clients align themselves with a commitment to enduring excellence and proactive protection in the web3 domain.

To book a security review, message https://t.me/vangrim1.

MIDGAR