yAudit Dopex RDPX Token Review

Review Resources:

- Dopex docs

Auditors:

- engn33r
- spalen

Table of Contents

Review Summary

Dopex RDPX

Dopex is a Decentralized Options Exchange that uses blockchain technology to provide permissionless and non-custodial access to options trading. The RDPX token is a rebate token of the Dopex platform and is used to partially compensate option writer losses to increase liquidity depth.

The contracts of the Dopex RDPX Token Repo were reviewed over 3 days. The code review was performed by 2 auditors between October 16 and October 18, 2023. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit b1bc0881aa01520bc8082b2dd35077120928c4ef for the Dopex RDPX token repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

- ArbRdpxTokenIntermediate.sol
- ArbRdpxTokenV2.sol
- Cup.sol

After the findings were presented to the Dopex team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Dopex and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Low	The token contract has an option to mint and pause. Both roles, as well as the admin role and the proxy admin owner, were set to the same EOA.
Mathematics	Good	No complex math was used.
Complexity	Good	Reviewed token upgrade didn't introduce unnecessary complexity to the token.
Libraries	Average	Only standard OZ libraries were used with the exception of custom `ERC20PresetMinterPauserUpgradeable`.
Decentralization	Low	The protocol does not have decentralized governance yet and the protocol team still controls many key protocol values without a Timelock.
Code stability	Average	The code is an upgrade to the currently deployed token on Arbitrum. However, some errors in the reviewed code, including the reverting `initialize()` function, demonstrate the code was not heavily tested.
Documentation	Average	The development team didn't provide documentation of the incident as to why the tokens are locked in the liquidity pool or which liquidity this token upgrade will recover. However, proper NatSpec and detailed comments existed on the functions that needed explaining.
Monitoring	Average	Events were emitted where applicable but missing in some functions.
Testing and verification	Low	Only a few tests were available at the time of the review which didn't cover all regular user flows nor added functionalities to the token.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements
- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

Critical Findings

None.

High Findings

1. High - Incorrect address used in `_transfer()` logic

The incorrect address is used when fetching the balance of the fee discount token for the `to` address.

Technical Details

The variable `feeDiscountTokenBalanceTo` fetches the value of token balance for address `from` but should fetch the balance for the address `to` as the variable name describes.

Impact

High. Users will pay the fee even if the `to` address has enough discount tokens.

Recommendation

Properly fetch the balance of the discount token for the `to` address.

```
- uint256 feeDiscountTokenBalanceTo = IERC20(feeDiscount.token).balanceOf(
-     from
- );
+ uint256 feeDiscountTokenBalanceTo = IERC20(feeDiscount.token).balanceOf(
+     to
+ );
```

Developer Response

Valid issue, will follow recommendation to fix.

Medium Findings

1. Medium - Blacklisted funds can be transferred

The default admin role can blacklist accounts and prevent them from transferring the RDPX token. But a blacklisted user still has the option to bridge the token to L1 using the Arbitrum gateway, which will burn blacklisted tokens on Arbitrum and give the user tokens on L1, where there is no RDPX blacklist functionality (yet).

Technical Details

The contract implementation of the Arbitrum gateway function `bridgeBurn()` doesn't check if the account is blacklisted before burning RDPX tokens. This allows the user to bridge the tokens back to L1, transfer it to another address, and bridge it back to Arbitrum to repeat the unwanted behaviour. Admins have no ability to break this loop with the existing implementation.

The two most popular tokens that implement blacklisting are USDC and USDT. Each has a different approach to what to do with blacklisted tokens. USDC prevents users from transferring blacklisted tokens. While USDT does the same, it has an additional option to destroy/burn blacklisted tokens.

Unibot token, which is used as an example for the blacklisting solution, only prevents users from transferring the tokens. Because Unibot is not bridgeable, it doesn't have a problem with moving/bridging blacklisted tokens.

Impact

Medium. Admin restrictions can be bypassed.

Recommendation

Verify that tokens are not blacklisted before calling `_burn()` function inside `bridgeBurn()`. Do the same check inside `bridgeMint()`. Include a require check:

```
require(!blacklisted[account], "Account blacklisted");
```

Bridged USDC.e on Arbitrum has implemented a similar solution to disable `bridgeMint()` and `bridgeBurn()` functions for blacklisted accounts.

Developer Response

Valid issue, will follow recommendation to fix.

2. Medium - RDPX token admin is EOA

The `DEFAULT_ADMIN_ROLE` of the RDPX v1 token, currently deployed on-chain, is an EOA. Because the admin role is privileged, it is recommended to avoid a single point of failure and use a multisig or similar smart contract as the admin.

Technical Details

The `DEFAULT_ADMIN_ROLE` address of the on-chain RDPX token can be checked with:

```
cast call 0x32Eb7902D4134bf98A28b963D26de779AF92A212 "getRoleMember(bytes32,uint256)(address)"
"0x0000000000000000000000000000000000000000000000000000000000000000" "0" --rpc-url https://arbitrum.llamarpc.com
```

The same EOA address has the `MINTER_ROLE` and `PAUSER_ROLE` roles too. The EOA is also the owner of the ProxyAdmin contract that governs upgrades of the token contract. Many DeFi protocols aim to increase trust in users by removing single points of failure and ensuring governance cannot act unilaterally without warning users. To this end, many protocols use multisig contracts for the privileged admin role of their contracts. The on-chain RDPX contract does not use the privileged default admin role for any specific purposes, but the new RDPX v2 token does. As part of the process of upgrading the token contract, it is recommended to change the privileged roles first in order to avoid a single point of failure if any unexpected events should impact the EOA address. Unexpected events could include loss of the private key, making the admin role inaccessible to anyone, or an unauthorized party gaining access to the private key through unexpected means.

Impact

Medium. The deployment scripts are not complete for RDPX v2, so a change in privileged roles may already be part of the plan. The current on-chain settings would not be ideal if carried over to RDPX v2 where the EOA has access to special functions like minting tokens and changing fees without notice.

Recommendation

Change the `DEFAULT_ADMIN_ROLE`, `MINTER_ROLE`, and `PAUSER_ROLE` from an EOA to a multisig or similar smart contract. First use `grantRole()` from OpenZeppelin's access control library to grant the new multisig each role, then call `revokeRole()` to remove the EOA from these roles.

The owner of the ProxyAdmin contract should also be changed to a multisig to improve protection around token contract upgrades.

Developer Response

Valid issue, the EOAs roles will be passed on to our Team Multisig.

Low Findings

1. Low - Ensure fees do not exceed 100%

`sellFees` and `buyFees` are declared with the comment `1e8 precision`, but there are no checks to prevent the fee from exceeding 100% (a value of 1e8).

Technical Details

`buyFees()` and `sellFees()` have 1e8 precision, but `updateBuyFees()` and `updateSellFees()` do not verify that the value stored exceeds 1e10. If a fee exceeds 100% (a value of 1e10), it will cause any transfer to fail due to an underflow in this subtraction.

Impact

Low. The value is admin controlled, but from the user perspective, the comments make it unclear how the fee value will be used in practice.

Recommendation

Add `require(_sellFees < 1e10);` to `buyFees()` and `require(_sellFees < 1e10);` to `sellFees()` to prevent the admin from settings fees that exceed the maximum intended value. If the actual maximum value is 1e10, then replace 1e10 with 1e8. The meaning of '1e8 precision' should be further elaborated on where these variables are declared or in the setter function NatSpec.

Developer Response

Valid issue, will follow recommendation to fix and add better comments.

2. Low - Transfer will revert on invalid fee discount

Ensure that buy and sell fees are above fee discount value and that the fee discount token is IERC20.

Technical Details

If `buyFees` or `sellFees` is below fee discount, every transfer where the user has fee discount will revert with arithmetic underflow. More importmant, if the fee discount token is not IERC20 compatible, every token transfer will fail on fetching token balance. This also means that the `feeDiscount` must be set before the any token transfer.

Impact

Low. Incorrect setup will prevent users from transferring tokens.

Recommendation

1. Before updating `feeDiscount`, ensure the `discount` value is below or equal to buy and sell fees. Also, verify that the discount token has the method `balanceOf`.
2. Add a require check before `feeDiscount`:

```
require(_feeDiscount.discount <= buyFees, "!buyFees");
require(_feeDiscount.discount <= sellFees, "!sellFees");
IERC20(_feeDiscount.token).balanceOf(address(this)); // verify token is IERC20.balanceOf compatible
feeDiscount = _feeDiscount;
```

With the current fees discount calculation, discount must be lower than both fees. If you want to have only buy fees, without sell fees, and add discount, you should change how fees are calculated in transfer method or split discount value for both fees.

Developer Response

Valid issue, will follow recommendation to fix and add better comments.

The intended token to be used for fee discount is veDPX which is a vote escrow token following the exact specification of the veCRV model which makes the ERC20 non-transferrable and decaying over time.

3. Low - Transfer will revert on sending fees to address(0)

Ensure that `feeCollector` is not address(0) before setting fees.

Technical Details

If `feeCollector` is address(0), every transfer that has fees will revert because it's not allowed to send `ArbRdpxTokenV2` to address(0).

Impact

Low. Incorrect setup will prevent users from transferring tokens.

Recommendation

A require check can prevent `feeCollector` from storing the zero address.

```
require(feeCollector != address(0), "!feeCollector");
```

This require check should be added in three places:

1. before setting `buyFees` in `updateBuyFees()`
2. before setting `sellFees` in `updateSellFees()`
3. before setting `feeCollector` in `updateFeeCollector()` to verify that the `feeCollector` is not set to address(0) after `buyFees` and `sellFees` are set.

Developer Response

Valid issue, will follow recommendation to fix.

Gas Saving Findings

1. Gas - Replace require with custom errors

Several require statements in RDPX v2 token can be replaced with custom errors. They are more gas efficient.

Technical Details

ArbRdpxTokenV2.sol contains 10 require statements that can all be changed to custom errors for minor gas savings.

Impact

Gas savings.

Recommendation

Use solidity custom errors instead of require statements.

2. Gas - Change `blacklistRenounced` type to uint256

A minor gas optimization is possible by changing the type of `blacklistRenounced` from bool to uint256.

Technical Details

`blacklistRenounced` is a bool but changing it to a uint256 type that stores 0 or 1 can save gas on deployment.

Impact

Gas savings.

Recommendation

Consider changing the type of `blacklistRenounced` for a minor gas savings on deployment demonstrated below.

Deployment Cost with bool type `blacklistRenounced`	Deployment Size
2234606	11196

Deployment Cost with uint256 type `blacklistRenounced`	Deployment Size
2227581	11153

3. Gas - Change static variables in Cup to constants

Several variables in Cup.sol can be constants, which saves gas on deployment and when calling `sip()`.

Technical Details

There are 6 variables in Cup.sol that are never modified and can be made constant to save gas. The tables below show the before/after gas differences.

Cup.sol (before change to use constants)				
Deployment Cost	Deployment Size			
4633293	22676			
Function Name	min	avg	median	max
sip	4025089	4025089	4025089	4025089

Cup.sol (after change to use constants)				
Deployment Cost	Deployment Size			
4545401	22708			
Function Name	min	avg	median	max
sip	4023193	4023193	4023193	4023193

Impact

Gas savings.

Recommendation

Make the 6 static variables in Cup.sol constants.

4. Gas - Optimize transfer function

Transfer function in contract `ArbRdpxTokenV2` can be gas optimized.

Technical Details

The internal `_transfer()` function can be optimized with small changes. This will be the most used function so any gas improvements will be good for end users and protocols.

Impact

Gas savings.

Recommendation

Rewrite the `_transfer()` function:

```
function _transfer(
    address from,
    address to,
    uint256 amount
) internal override {
    require(!blacklisted[from], "Sender blacklisted");
    require(!blacklisted[to], "Receiver blacklisted");
    if (amount == 0) {
        super._transfer(from, to, 0);
        return;
    }

    if (!excludedFromFees[from] && !excludedFromFees[to]) {
        uint256 fees;
        IERC20 discountToken = IERC20(feeDiscount.token);
        uint256 feeDiscountTokenBalanceFrom = discountToken.balanceOf(from);
        uint256 feeDiscountTokenBalanceTo = discountToken.balanceOf(to);
        uint256 _sellFees = sellFees;
        uint256 _buyFees = buyFees;

        // reduce _sellFees and _buyFees if applicable
        uint256 requiredBalanceCached = feeDiscount.requiredBalance;
        if (
            feeDiscountTokenBalanceFrom >= requiredBalanceCached ||
            feeDiscountTokenBalanceTo >= requiredBalanceCached
        ) {
            uint256 feeDiscountCached = feeDiscount.discount;
            _sellFees -= feeDiscountCached;
            _buyFees -= feeDiscountCached;
        }
        // only take fees on buys/sells, do not take on wallet transfers
        // on sell
        if (automatedMarketMakerPairs[to] && _sellFees > 0) {
            fees = (amount * _sellFees) / 1e10;
            super._transfer(from, feeCollector, fees);
            amount -= fees;
        }
        // on buy
        else if (automatedMarketMakerPairs[from] && _buyFees > 0) {
            fees = (amount * _buyFees) / 1e10;
            super._transfer(from, feeCollector, fees);
            amount -= fees;
```

```
        }
    }

    super._transfer(from, to, amount);
}
```

This table shows a comparison of current and proposed `_transfer()` function gas costs. The test covers four different scenarios in the `_transfer()` function.

Function Name	min	avg	median	max	# calls
transfer original	3911	35311	36063	65207	4
transfer proposed	3841	33259	32372	64453	4

5. Gas - Unnecessary setting of variables to default values

Variables don't need to be set to their default values.

Technical Details

Variable `blacklistRenounced` is set to `false` while the default value for any bool is already `false`.

Impact

Gas savings.

Recommendation

Remove explicitly setting `blacklistRenounced` to false. Add a comment that false is the default value for clarity.

```
bool public blacklistRenounced; // false is default value
```

This will reduce contract deployment cost and size:

ArbRdpxTokenV2 contract		
Deployment Cost	Deployment Size	Version
2214784	11097	Original
2212560	11083	Recommended

Informational Findings

1. Informational - Transfer could revert on really high amounts

For the amounts that are close to uint256 maximum value, transfer attempts will revert if a fee is applied.

Technical Details

When calculating fees both sell and buy fees, there is a possibility of overflow. For example, if the amount to transfer is `uint256.max / 1e7` and the `_sellFees` value is `1e8`, the multiplication will overflow because the product is greater than `type(uint256).max`.

Impact

Informational.

Recommendation

Transferring such high amounts of tokens is highly unlikely, so there is no urgent need to change the code. The issue can be bypassed by sending lower amounts or turning off the fees.

Developer Response

We are planning to offset this via decreasing the precision of the fee % to 1e4, this does not mitigate the issue entirely but increases the max amount of transfers possible.

2. Informational - Remove unneeded import

Remove unneeded imports from contracts for cleaner code.

Technical Details

Cup.sol imports hardhat console which is not needed in production.

Impact

Informational.

Recommendation

Remove specified import.

3. Informational - Call `initialize()` on implementation contracts

In `sip()`, two new RDPX token implementation contracts are created. The contracts should be initialized to comply with best practices, even if there is no immediate security risk.

Technical Details

The latest OpenZeppelin Initializable contract implementation includes a warning to initialize all initializable contracts immediately after deployment. This suggestion applies not only to calling initialize on a proxy contract, but to implementation contracts too. The latest Initializable contract has a `_disableInitializers()` function designed for implementation contracts behind proxies, but since an older version of OpenZeppelin libraries is used here (likely to maintain storage slot consistency between implementation contract versions), initializing the implementation contracts is a fine alternative.

Impact

Informational.

Recommendation

Initialize the ArbRdpxTokenIntermediate implementation immediately after it's deployed and initialize the ArbRdpxTokenV2 implementation immediately after it's deployed.

4. Informational - `initialize()` in RDPX token reverts

The `initialize()` function in RDPX v2 reverts. This is more of a development concern than a security concern, since the main vulnerability of an uninitialized contract is that someone else can call `initialize()`, but in this case no one can call `initialize()`.

Technical Details

There is a problem with the inheritance of the `initializer()` modifier that causes calls to the `initialize()` function to revert when called. When the `__ERC20PresetMinterPauser_init()` call is removed from the initializer, the revert is no longer an issue.

Impact

Informational.

Recommendation

Fix the revert issue so `initialize()` can be called.

5. Informational - Flashloans could allow fee bypass

The RDPX v2 token has a `feeDiscount.discount` value that reduces the buy fee and sell fee. This discount is available for any holder of a sufficient quantity of `feeDiscount.token` tokens. If this token can be borrowed in a flashloan for a negligible cost, any user can receive the fee discount.

Technical Details

The fee discount is applied when if the sender or recipient have a certain number of `feeDiscount.token` tokens. If flashloans are possible for this token, anyone planning to buy or sell with an AMM that would cause fees to be charged could flashloan enough tokens before the AMM trade and always receive the discounted rate.

Impact

Informational.

Recommendation

The easiest solution is for the contract admin to only choose a `feeDiscount.token` that does not have flashloans, but a comment should be added to `updateFeeDiscount()` to specify such a choice if that is the chosen solution. The current plan is to choose veDPX, which is documented as a non-transferable token. If a flashloan bypass of the standard fee payment is not acceptable, consider an alternative approach to applying the discount or store the most recent block that a RDPX v2 token transfer happened so that transfers that may involve flashloans can be detected.

Developer Response

The intended token to be used for fee discount is veDPX which is a vote escrow token following the exact specification of the veCRV model which makes the ERC20 non-transferrable and decaying over time and hence does not feature flashloans.

6. Informational - Timelock key admin operations for user protection

Certain admin-controlled actions, such as setting `sellFees` or `buyFees`, can negatively impact users who transfer RDPX v2 tokens without knowledge of the new fee. Using a timelock contract as an admin can provide extra assurance to users that no sudden changes will happen without warning.

Technical Details

The default admin role of RDPX v1, currently deployed on-chain, is an EOA. If the same admin is used for RDPX, actions such as `updateBuyFees()` and `updateSellFees()` can be performed at any time without a notice to users. Well-known protocols use timelock contracts (Compound Finance v2, Uniswap v2) around fee change actions to ensure no sudden actions happen.

Impact

Informational.

Recommendation

Consider protecting users with a Timelock contract around key actions that can impact holders of the token.

7. Informational - Add safety checks in Cup.sip() for smooth upgrade process

A number of extra safety checks can be added to `Cup.sip()` to ensure the WETH rescue process happens as planned and reverts if any unexpected scenario is encountered.

Technical Details

- `Cup.sip()` does not burn the RDPX tokens transferred by Cup.sol to the pool during the WETH rescue process. The same `blaze()` call, burning all pool RDPX tokens except for 1 wei, can be performed before the code is upgraded from the Intermediate token logic to the RDPX V2 logic contract. After the `blaze()` call, `pair.sync()` must also be called so that the pool is aware of this change. This way, there is no supply of RDPX that might be retrieved at some future point that could alter circulating supply of the token. If the tokens in the pool are not burned, there is negligible impact as long as the number of tokens left in the pool is very small relative to the total RDPX supply. If the number of left tokens is a considerable percentage of the total RDPX supply, anyone able to access these tokens profitably by swapping WETH into the pool could cause inflation of RDPX supply and loss of value to other RDPX holders.
- To further ensure no RDPX tokens are left behind in the pool, after the new `blaze()` call and before the upgrade to RDPX v2, add the line `require(rdpx.balanceOf(address(pair)) == 1);`
- To make sure that there is only a small dust amount of WETH left behind in the pair contract, add a check to verify the remaining WETH in the pair is below a certain amount, for example `require(weth.balanceOf(address(pair)) < 1e14);`

Impact

Informational.

Recommendation

Make the changes described above to ensure a safe migration to RDPX v2 without any loss of value from trapped tokens in the pool.

8. Informational - Replace magic numbers with constants

Constant variables should be used in place of magic numbers to prevent typos and better explain such values. For one example, the magic number 1e10 is found twice in `ArbRdpxTokenV2._transfer()` and the value 10000 appears three times in Cup.sol. These values could be constants. Using a constant also adds a description to the value to explain the purpose of the value.

Technical Details

ArbRdpxTokenV2.sol uses magic number 1e10 in `_transfer()`. Another unexplained number is 10000 in Cup.sol. Consider replacing these magic numbers with a constant private variable. This will not change gas consumption.

Impact

Informational.

Recommendation

Use constant variables instead of magic numbers.

9. Informational - Rescue more WETH from the pool

More WETH can be rescued from the pool.

Technical Details

In function `sip()`, more WETH value can be withdrawn if the value `10000` is changed to `1e12` or a similar larger value. By increasing the value, the WETH left in the pool is lowered from `0.1176 ether` to only `0.00000000138162 ether`.

Impact

Informational.

Recommendation

Change the value `10000` to `1e12` in all places in `sip()`.

10. Informational - Missing events for blacklist functions

The blacklist functions change the value of a state variable, but it's not tracked with events. Adding events can enable better on-chain analysis.

Technical Details

Functions `blacklist(address _addr)` and `unblacklist(address _addr)` change the state of mapping `blacklisted` but don't emit events about changed state.

Impact

Informational.

Recommendation

Add events to specified functions for better on-chain analysis.

Final remarks

Overall the $rDPX token upgrade should deliver its expected result of rescuing WETH from a pool. However, the lack of documentation explaining why the token is being upgraded and exactly what liquidity is being rescued will draw some questions, especially because the pool is being actively used at the time of this review. The team explained that the burned tokens from the liquidity pool will be minted back to the users by taking a snapshot of the existing state and moving liquidity to another protocol. The token upgrade introduces additional power to the team by introducing a blacklisting option. Also, transfer fees can be set by the team for specific cases like transfers on buy and sell. These changes increase the centralization of token management, which could be partially mitigated by introducing a multisig and more decentralized governance. The team was advised to inform the users before turning on the fees and blacklisting the users.