Dyad Audit Report

Version 1.0

Shikhar Agarwal

June 20, 2024

Dyad Report

Shikhar Agarwal

June 20, 2024

Prepared by: Shikhar Agarwal

Lead Auditors: - Shikhar Agarwal

Table of Contents

About the Project

DYAD is the first truly capital efficient decentralized stablecoin. Traditionally, two costs make stablecoins inefficient: surplus collateral and DEX liquidity. DYAD minimizes both of these costs through Kerosene, a token that lowers the individual cost to mint DYAD.

See more contest details here

Disclaimer

As the sole auditor all efforts have been made to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

QA (Quality Assurance) Includes Low risk (e.g. assets are not at risk: state handling, function incorrect as to spec, issues with comments) and Governance/Centralization risk (including admin privileges). Excludes Gas optimizations, which are submitted and judged separately. Non-critical issues code style, clarity, syntax, versioning, off-chain monitoring (events, etc) are discouraged.

2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.

3 — High: Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).

Audit Details

```
Commit Hash: cd48c684a58158de444b24854ffd8f07d046c31b
```

Scope

- src/core/VaultManagerV2.sol
- src/core/Vault.kerosine.bounded.sol
- src/core/Vault.kerosine.sol
- src/core/Vault.kerosine.unbounded.sol
- src/core/KerosineManager.sol
- script/deploy/Deploy.V2.s.sol
- src/staking/KerosineDenominator.sol

Roles

DYAD Multisig: 0xDeD796De6a14E255487191963dEe436c45995813

DYAD Multisig - Ability to: License new Vault Manager, License new Vaults, Change the kerosene denominator contract, Add new vaults to the Kerosene Manager

Issues found

Severity	Count of Findings
High	5
Medium	0
QA	1

Findings

-

High Risk Findings

-

H-01. Missing enough exogeneous collateral check in `VaultManagerV2::liquidate` makes the liquidation revert even if (DYAD Minted > Non Kerosene Value)

-

H-02. Incorrect ExoCollateral check inside `VaultManagerV2::withdraw` doesn't allow the user to withdraw their kerosene vault collateral, even if they are fully collateralized.

-

H-03. `VaultManagerV2::liquidate` using only `vaults` for repayment to liquidator and not `vaultsKerosene` leads to liquidator getting disincentivized

-

H-04. `VaultManagerV2::addKerosene` checks the kerosene vault being added from `keroseneManager` leads to addition of non kerosene token vaults to `vaultsKerosene`

-

H-05. `DeployV2` script adds the unboundedKerosineVault in `vaultLicenser`, allowing users to add kerosene token based vault in their `vaults` mapping.

•

QA

-

QA-01. Incorrect mint and withdrawal check in `VaultManagerV2` contract as mentioned in the protocol docs

High

High Risk Findings

H-01. Missing enough exogeneous collateral check in `VaultManagerV2::liquidate` makes the liquidation revert even if (DYAD Minted > Non Kerosene Value)

Relevant GitHub Links

https://github.com/code-423n4/2024-04-dyad/blob/main/src/core/VaultManagerV2.sol#L205

Vulnerability details

Vulnerability Details

The vulnerability is present in the `VaultManagerV2::liquidate` function where it doesn't have any check for whether the vault has enough exogeneous collateral leading to no liquidation even if the non-kerosene value is less than DYAD minted.

It is intended that the position of user's DNFT has enough exogeneous collateral and should be 150% overcollateralized. But there will arise a case when a position is no doubt 150% collateralized by the support of kerosene value, but the non-kerosene value is reduced below the DYAD token minted. As a result of which the vault doesn't have enough required exogeneous collateral. But due to the missing check for enough non-kerosene value collateral in liquidate function, the liquidation will never happen and reverts.

Impact

The position will never be liquidated, for the above discussed case. As a result of which the (DYAD Minted > Non Kerosene Value), making the value of DYAD to fall.

Proof of Concept

Add the below test file in `test/fork/v2.t.sol`

Add the imports:

```
import {DNft} from "../../src/core/DNft.sol";
import {ERC20} from "@solmate/src/tokens/ERC20.sol";
import {OracleMock} from "../OracleMock.sol";
import {IVaultManager} from "../../src/core/VaultManagerV2.sol";
```

Run the test (Replace the $ETH_MAINNET_RPC_URL by your eth mainnet rpc url):

```
forge test --mt
    test_LiquidationReverts_EvenIfVaultHasNotEnoughExoCollateral --fork-
    url $ETH_MAINNET_RPC_URL
```

```
function test_LiquidationReverts_EvenIfVaultHasNotEnoughExoCollateral()
     public {
  address user = makeAddr("user");
  vm.deal(user, 100 ether);

  Licenser licenser = Licenser(MAINNET_VAULT_MANAGER_LICENSER);
  vm.prank(MAINNET_OWNER);
  licenser.add(address(contracts.vaultManager));

  DNft dnft = DNft(MAINNET_DNFT);

  vm.startPrank(user);

  uint256 id = dnft.mintNft{value: 10 ether}(user);

  // user adds vault
```

```
16    contracts.vaultManager.add(id, address(contracts.ethVault));
17    contracts.vaultManager.addKerosene(id, address(contracts.wstEth));
18
19    // user adds weth to the vault
20    deal(MAINNET_WETH, user, 10 ether);       // get 10 weth to the user
21    ERC20(MAINNET_WETH).approve(address(contracts.vaultManager), 10 ether
          );
22    contracts.vaultManager.deposit(id, address(contracts.ethVault), 10
          ether);
23
24    // user adds wsteth to the vault
25    deal(MAINNET_WSTETH, user, 10 ether);       // get 10 wsteth to the
          user
26    ERC20(MAINNET_WSTETH).approve(address(contracts.vaultManager), 10
          ether);
27    contracts.vaultManager.deposit(id, address(contracts.wstEth), 10
          ether);
28
29    // user mints DYAD token
30    uint256 nonKeroseneValue = contracts.vaultManager.getNonKeroseneValue
          (id);
31
32    // user mints DYAD equal to the usd value of their non-kerosene
          collateral
33    uint256 dyadToMint = nonKeroseneValue;
34    contracts.vaultManager.mintDyad(id, dyadToMint, user);
35
36    // now the value of weth falls to 90%
37    _manipulateWethPrice();
38
39    // now get the nonKeroseneValue
40    uint256 newNonKeroseneValue = contracts.vaultManager.
          getNonKeroseneValue(id);
41
42    // non kerosene usd value reduced, as weth price reduced
43    assert(newNonKeroseneValue < dyadToMint);
44
45    // now it is intened that the user should be liquidated by a
          liquidator
46    // but due to missing exogeneous collateral check, the liquidation
          will not happen
47
48    vm.stopPrank();
49
50
51    address liquidator = makeAddr("liquidator");
52    vm.deal(liquidator, 100 ether);
53
54    vm.startPrank(liquidator);
55
56    uint256 liquidatorId = dnft.mintNft{value: 10 ether}(liquidator);
```

```
57
58    // the liquidation reverts as mentioned, even if (dyadMinted >
          nonKeroseneValue)
59    // thus shows unintended behaviour
60    vm.expectRevert(IVaultManager.CrTooHigh.selector);
61    contracts.vaultManager.liquidate(id, liquidatorId);
62
63    vm.stopPrank();
64  }
65
66  function _manipulateWethPrice() internal {
67    (, int256 ans, , , ) = contracts.ethVault.oracle().latestRoundData();
68    OracleMock oracleMock = new OracleMock(uint256(ans));
69    vm.etch(address(contracts.ethVault.oracle()), address(oracleMock).
          code);
70    vm.warp(1);
71    OracleMock(address(contracts.ethVault.oracle())).setPrice(uint256(ans
          ) * 90e18 / 100e18);
72  }
```

Tools Used

Manual Review, Unit Test in Foundry

Recommended Mitigation Steps

Update the line 214 in `VaultManagerV2.sol`

```
1  - if (cr >= MIN_COLLATERIZATION_RATIO) revert CrTooHigh();
2  + if (cr >= MIN_COLLATERIZATION_RATIO && getNonKeroseneValue(id) >=
       dyad.mintedDyad(address(this), id)) revert CrTooHigh();
```

Assessed type

Context

H-02. Incorrect ExoCollateral check inside `VaultManagerV2::withdraw` doesn't allow the user to withdraw their kerosene vault collateral, even if they are fully collateralized.

Relevant GitHub Links

https://github.com/code-423n4/2024-04-dyad/blob/main/src/core/VaultManagerV2.sol#L150

Vulnerability details

Vulnerability Details

The vulnerability is present in the `VaultManagerV2::withdraw`, where it compares the DYAD minted against the remaining non-kerosene value remaining in the vault. But, when a user withdraws a kerosene token from their vault, the DYAD Minted is compared by checking it against non-kerosene value subtracted by the value of kerosene amount the user wants to burn.

```
1 if (getNonKeroseneValue(id) - value < dyadMinted) revert
    NotEnoughExoCollat();
```

As, this check is dedicated for ensuring that user has more non-kerosene value than the DYAD minted.

But, even during withdrawal of kerosene token, it checks the net non-kerosene value by subtracting from it the withdrawal value of kerosene token and that is irrelevant, as this check is purely for non-kerosene value.

Impact

The user will not be able to withdraw their kerosene token even if they are fully collateralized as it considers the user's non-kerosene value by subtracting their kerosene withdrawal value and as a result of which it incorrectly reverts with `NotEnoughExoCollat` even there exo collateral is sufficient.

Proof of Concept

Consider the case where user has deposited $100 worth of weth in weth vault, and has $100 worth of kerosene token in unbounded kerosene vault and has minted 100 DYAD token.

Therefore, non-kerosene value - $100 (from weth vault) and, kerosene value - $100 (from unbounded kerosene vault)

Now, the user has sufficient exogeneous collateral as (DYAD <= non-kerosene value) and along with fully collateralized(200 / 100 = 2 = 200% collateralization).

But while withdrawing $50 worth of kerosene token from unbounded kerosene vault, there net non-kerosene value is considered by subtracting this $50 value of withdrawal even for the kerosene withdrawals, and thus it incorrectly reverts with `NotEnoughExoCollat` even there exo collateral is sufficient.

Tools Used

Manual Review

Recommended Mitigation Steps

For kerosene token withdrawals, don't perform the below check as it is only required for withdrawing non-kerosene token collateral.

```
if (getNonKeroseneValue(id) - value < dyadMinted) revert
    NotEnoughExoCollat();
```

Assessed type

Other

H-03. `VaultManagerV2::liquidate` using only `vaults` for repayment to liquidator and not `vaultsKerosene` leads to liquidator getting disincentivized

Relevant GitHub Links

https://github.com/code-423n4/2024-04-dyad/blob/main/src/core/VaultManagerV2.sol#L221-L226

Vulnerability details

Vulnerability Details

- The vulnerability is present in the `VaultManagerV2::liquidate` where it only repays the exogeneous collateral (in `vaults`) to the liquidator and doesn't pay the kerosene collateral (in `vaultsKerosene`) which makes the liquidator to pay more DYAD than what they have received in collateral, thus disincentivizing them.
- As a user's position comprises both of kerosene and non-kerosene collateral, but when a user's non-kerosene collateral falls below such that it causes the overall collateral ratio to fall below, then the liquidator's liquidating them is only paid in the non-kerosene value which is now actually less than the DYAD to be paid after fall of non-kerosene collateral's price fall.

Impact

- Liquidators are disincentivized while liquidating a position, as the collateral they receive is only non-kerosene and doesn't receive the user's kerosene collateral due to usage of only `vaults` and not `vaultsKerosene` for collateral payment in `liquidate` function.
- This will not attract any liquidator to liquidate a position, and will create several undercollateralized position, and which will eventually affect the TVL value, thus have a negative impact on the invariant (TVL > DYAD Supply).

Proof of Concept

Consider the scenario as below: - A user has added `weth` vault to their `vaults[id]` and `UnboundedKerosineVault` to their `vaultsKerosene`. - User deposits $100 value of weth collateral in the `weth` vault and $50 value of kerosene collateral in `UnboundedKerosineVault`. - Then user mints the maximum possible DYAD, i.e. equal to their non-kerosene value, which will be $100, as `weth` is the only non-kerosene collateral they deposited. Therefore DYAD minted = 100 DYAD - Now, the value of their non-kerosene collateral falls to $90, therefore their non-kerosene value = $90. - Collateral Ratio = (Non-Kerosene Value + Kerosene Value) / DYAD Minted = (90 + 50) / 100 = 1.4 = 140% - Therefore, they are required to be liquidated as collateral ratio < 150% - `cappedCr` = 1.4 - `liquidationEquityShare` = (cappedCr - 1) * 0.2 = (1.4 - 1) * 0.2 = 0.08 - `liquidationAssetShare` = (liquidationEquityShare + 1) / cappedCr = (0.08 + 1) / 1.4 = 0.7714 = 77.14% - Now, only vaults from `vaults` mapping are used for repayment of collateral to liquidator, that means only `weth` vault is used. - Therefore, collateral received by liquidator = vault.id2asset(id).mulWadUp(liquidationAssetShare) = 77.14% of ($90 worth of weth) = $69.426 worth of weth - Therefore, only receiving $69.426 of weth as repayment for paying the debt of 100 DYAD. - Hence, disincentivizing the liquidator.

Tools Used

Manual Review

Recommended Mitigation Steps

Along with the usage of `vaults` for payment of collateral, also use `vaultsKerosene`, so that the kerosene collateral can also be paid to the liquidator.

Assessed type

Context

H-04. `VaultManagerV2::addKerosene` checks the kerosene vault being added from `keroseneManager` leads to addition of non kerosene token vaults to `vaultsKerosene`

Relevant GitHub Links

https://github.com/code-423n4/2024-04-dyad/blob/main/src/core/VaultManagerV2.sol#L88
https://github.com/code-423n4/2024-04-dyad/blob/main/script/deploy/Deploy.V2.s.sol#L64-L65

Vulnerability details

Vulnerability Details

The vulnerability is present in the `VaultManagerV2::addKerosene`, as it verifies the vault being added from the `keroseneManager` contract.

The kerosene manager contract is intended to contain all the non-kerosene token related vaults that are to be used for TVL calculation while calculating the price of kerosene token inside `UnboundedKerosineVault::assetPrice`.

From the deploy scripts it can be noticed that to the `keroseneManager`, weth and wsteth vaults are added, as these are the vaults from where TVL will be calculated.

But, the thing is that `addKerosene` function is intended to add kerosene token related vaults to the `vaultsKerosene` mapping corresponding to the user's DNFT id, and as a result of which non-kerosene vaults can also be added by user to `vaultsKerosene`.

Impact

- Non-kerosene token vaults can be added to the `vaultsKerosene` mapping.
- As `weth` vaults and `wsteth` vaults are non-kerosene vaults and contribute to non-kerosene value which is the exogeneous collateral, but a user adding these vaults to their `vaultsKerosene` will make them also contribute to the kerosene value. (Given a user added weth and wsteth to their vaults.) Therefore, while calculating the `VaultManagerV2::collatRatio` of a user's DNFT, it will include their weth and wseth vaults in both non-kerosene

value (intended) as well as in kerosene value (unintended). Thus, having a collateral ratio pumped by 2x.

Proof of Concept

Below is a Unit Test written to justify the finding. Add the test in the file: `test/fork/v2.t.sol`

Run the test (Replace $ETH_MAINNET_RPC_URL with your eth mainnet rpc url):

```
forge test --mt test_NonKeroseneVaultsContributeToKeroseneVaults --fork-url $ETH_MAINNET_RPC_URL -vv
```

Add the imports:

```
import {DNft} from "../../src/core/DNft.sol";
import {ERC20} from "@solmate/src/tokens/ERC20.sol";
```

```
function test_NonKeroseneVaultsContributeToKeroseneVaults() public {
  address user = makeAddr("user");
  vm.deal(user, 100 ether);

  Licenser licenser = Licenser(MAINNET_VAULT_MANAGER_LICENSER);
  vm.prank(MAINNET_OWNER);
  licenser.add(address(contracts.vaultManager));

  DNft dnft = DNft(MAINNET_DNFT);

  vm.startPrank(user);

  uint256 id = dnft.mintNft{value: 10 ether}(user);

  // user adds eth vault to their vaults
  contracts.vaultManager.add(id, address(contracts.ethVault));

  // now due to incorrect vault address check in addKerosene, user is able to add non-kerosene vault
  // to their vaultsKerosene
  contracts.vaultManager.addKerosene(id, address(contracts.ethVault));

  // user adds weth to the vault
  deal(MAINNET_WETH, user, 10 ether);       // get 10 weth to the user
  ERC20(MAINNET_WETH).approve(address(contracts.vaultManager), 10 ether);
  contracts.vaultManager.deposit(id, address(contracts.ethVault), 10 ether);

  // notice that the user has added only 10 eth as collateral
  uint256 userNonKeroseneValue = contracts.vaultManager.getNonKeroseneValue(id);
```

```
29
30    // user mints `userNonKeroseneValue` amount of DYAD token
31    contracts.vaultManager.mintDyad(id, userNonKeroseneValue, user);
32
33    // it is intended that there mint should fail as they are only 100%
          collateralized
34    // but the same amount of 10 weth value being considered in their
          kerosene value makes them 200% coll (i.e., 2e18)  which is more
          than 150% req overcollaterlization
35    // as a result of which dyas is minted with only 100%
          collateralization
36    vm.stopPrank();
37
38    // the collateralization comes out to be 200% due to considering non-
          kerosene vaults in kerosene vaults
39    console.log("Collateralization Ratio - %e", contracts.vaultManager.
          collatRatio(id));
40  }
```

Tools Used

Manual Review, Unit Test in Foundry

Recommended Mitigation Steps

Instead of validating the vaults being added to `vaultsKerosene` via `keroseneManager` contract, use another licenser contract for the kerosene vaults. As `keroseneManager` is for vaults to be used for TVL calculation. The usage of dedicated licenser for kerosene vaults is recommended which allows the owner of the DYAD protocol to add the kerosene vaults, and make `keroseneManagerV2` to validate the kerosene vaults being added via that contract.

Assessed type

Invalid Validation

H-05. `DeployV2` script adds the unbounded`KerosineVault` in `vaultLicenser`, allowing users to add kerosene token based vault in their `vaults` mapping.

Relevant GitHub Links

https://github.com/code-423n4/2024-04-dyad/blob/main/script/deploy/Deploy.V2.s.sol#L95
https://github.com/code-423n4/2024-04-dyad/blob/main/src/core/VaultManagerV2.sol#L75

Vulnerability details

Vulnerability Details

The vulnerability is present in the `DeployV2` script where it adds the `unboundedKerosineVault` to `vaultLicenser` contract.

The `vaultLicenser` contract is used in `VaultManagerV2::add` function which validates adding only licensed vaults in `vaults` mapping. The `vaults` contains all the vaults for a DNFT id issued to a user, and those vaults contributes to the user's non-kerosene collateral value.

But, licensing the `unboundedKerosineVault` allows the user to make a kerosene token vault contribute to the non-kerosene value by adding it to the `vaults` via `VaultManagerV2::add`.

Impact

Allowing the user to add `unboundedKerosineVault` in the `vaults` mapping will make it contribute to the non-kerosene value. As `unboundedKerosineVault` is intended for only contributing to a user's kerosene collateral value, but due to this identified vulnerability it will also contribute to the non-kerosene value.

Therefore, a user who has added `unboundedKerosineVault` in their `vaultsKerosene` as intended as well as in `vaults` which was unintended, will make their collateral deposited to be counted in the same vault more than one time and making their collateral value to be 2x their actual deposit.

Thus, allowing users to mint more amount of DYAD token, even if they are undercollateralized.

Proof of Concept

The deploy scripts perform the following action:

```
1  vaultLicenser.add(address(unboundedKerosineVault));
```

As a result of which they are allowed to add `unboundedKerosineVault` in their `vaults` mapping corresponding to their DNFT id and this is the unintended behavior.

The user also adds `unboundedKerosineVault` in their `vaultsKerosene` which is intended functionality.

But the `vaults` mapping was intended to store only non-kerosene token related vaults, thus showing unintended behavior.

Now same vault being in both `vaults` and `vaultsKerosene` will make the `getTotalUsdValue` to output 2x the actual collateral amount, as `getKeroseneValue` will return their kerosene value as intended from the amount deposited in `unboundedKerosineVault`, but due to the unintended addition of the same vault in `vaults` mapping will also make `getNonKeroseneValue` to also return the same collateral amount, but from a kerosene token based vault.

Thus, making the same vault to return collateral amount both for kerosene value as well as non-kerosene value.

Tools Used

Manual Review

Recommended Mitigation Steps

```
vaultLicenser.add(address(unboundedKerosineVault));
```

The above action was responsible for this vulnerability. Therefore, consider not adding `unboundedKerosineVault` to `vaultLicenser` as it is only for non-kerosene token vaults. Make a seperate licenser contract for the `vaultsKerosene` and add `unboundedKerosineVault` in that.

Assessed type

Context

QA

QA-01. Incorrect mint and withdrawal check in `VaultManagerV2` contract as mentioned in the protocol docs

Relevant GitHub Links

https://github.com/code-423n4/2024-04-dyad/blob/main/src/core/VaultManagerV2.sol#L165
https://github.com/code-423n4/2024-04-dyad/blob/main/src/core/VaultManagerV2.sol#L150

Vulnerability details

Vulnerability Details

- The vulnerability occurs due to the incorrect minting and withdrawing check in `VaultManagerV2` contract.
- The docs mentions that the non-kerosene value should always be strictly greater than the DYAD Minted, but in actual implementation it allows the non-kerosene value to be >= to DYAD Minted.

Impact

- User will be able to mint DYAD equal to the non-kerosene value, but DYAD Minted should be strictly less than non-kerosene value as mentioned in docs.
- User will be able to leave non-kerosene value equal to DYAD Minted after withdrawal of some amount, but the net non-kerosene value after withdrawal should be strictly greater than DYAD Minted.

Proof of Concept

The above mentioned docs mentions about mint check and withdrawal check: https://dyadstable.notion.site/DYAD-design-outline-v6-3fa96f99425e458abbe574f67b795145

Tools Used

Manual Review

Recommended Mitigation Steps

Modify the code as mentioned in the docs for the required check on non-kerosene value and DYAD Minted during minting and withdrawal.

Assessed type

Invalid Validation