DYAD Security Review

Auditor: Al-Qa'qa’

30 June 2024

Table of Contents

1 Introduction

1.1 About Al-Qa'qa'

Al-Qa'qa' is an independent Web3 security researcher who specializes in smart contract audits. Success in placing top 5 in multiple contests on code4rena and sherlock. In addition to smart contract audits, he has moderate experience in core EVM architecture, geth.

For security consulting, reach out to him on Twitter - @Al_Qa_qa

1.2 About DYAD

DYAD is a stablecoin protocol (Dollar pegged), it allows *ERC721* positions, where positions can be traded on third markets. In addition to this, they introduce the kerosine token, which its value depends on the volume of minted DYAD, and the locked collaterals, so it is as valuable as the degree of DYAD's overcollateralization.

1.3 Disclaimer

Security review cannot guarantee *100%* the safeness of the protocol, In the Auditing process, we try to get all possible issues, and we can not be sure if we missed something or not.

Al-Qa'qa' is not responsible for any misbehavior, bugs, or exploits affecting the audited code or any part of the deployment phase.

And change to the code after the mitigation process, puts the protocol at risk, and should be audited again.

1.4 Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.4.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality
- Low - Funds are **not** at risk

1.4.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align or little-to-no incentive

2 Executive Summary

2.1 Overview

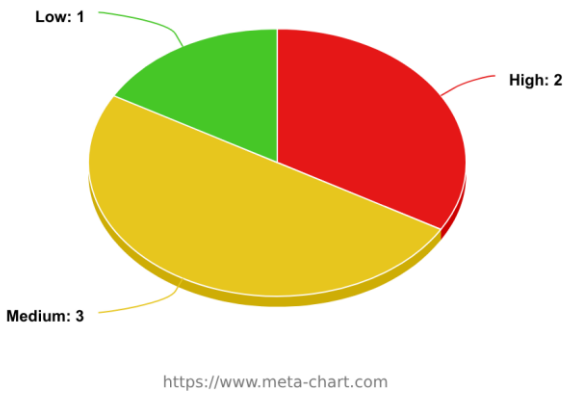
Project	DYAD Stablecoin
Repository	DyadStableCoin::deploy/vaultManagerV3
Commit Hash	3ddcbcc7616ba6cacef4f381c90bda6b8f2245d4
Mitigation Hash	059b8fcde7c17777bed914d2fb2ed18ed1779694
Audit Timeline	13 June 2024 to 17 June 2024

2.2 Scope

- src/core/VaultManagerV3.sol
- script/deploy/DeployVaultManagerV3.s.sol

2.3 Issues Found

Severity	Count
High Risk	2
Medium Risk	3
Low Risk	1
Total Issues	**6**



3 Finding Summary

ID	title	status
H-01	Free DYAD can be existed without collateral because of partial liquidation edge case	Resolved
H-02	Upgrading process uses initialize instead of reinitialize modifier	Resolved
M-01	Liquidation process will get *Dos'ed* if *tvl* falls below *DYAD::totalSupply*	Resolved
M-02	Partial Liquidation Bonus is set wrongly	Acknowledge
M-03	*DeployVaultManagerV3* compilation will fail as *encodeCall()* expects two args	Resolved
L-01	Single-step ownership transfer mechanism by *OwnableUpgradeable*	Acknowledge
L-02	The new liquidation implementation does not round in the favour of the liquidator	Resolved
I-01	mappings key/value support names, no need to comment them	Acknowledge

4 Findings

4.1 High Risk

4.1.1 Free DYAD can be existed without collateral because of partial liquidation edge case

Severity: HIGH

Context: VaultManagerV3.sol#L198-L199

Description: There is support now to make partial liquidation, but the problem is that the liquidator can take all undercollateralized position collateral without burning all DYAD minted by that position.
Since there is a *20%* bonus if the position goes below *120%* the value is capped to send all collaterals in the vault to the liquidator. However, since there is partial liquidation, the liquidator can escape burning all position DYAD and will still get all the collaterals.

VaultManagerV3.sol#L199

```
        uint cappedValue = valueToMove > value ? value : valueToMove;
```

This will cause a critical situation, where it will leave a DYAD with actually zero collaterals (*CR* = 0). This will affect the coin stability as the coin should have enough collateral in order to retain stability, and in this situation, it will lead to inflation.
The problem will not stop there, as the team may do the liquidation himself and sacrifice the DYAD he will burn in order to retain coin stability. But the problem is that this position will be *unliquidatable,* and the tx will always revert as the *totalValue = 0* in this case (all collaterals have been moved to the previous liquidator), and we divide by *totalValue* to get the *share* value.

VaultManagerV3.sol#L196

```
        uint totalValue = getTotalValue(id);
        ...
        uint share = value.divWadDown(totalValue);
```

So this will end up having free DYAD in the market without any collaterals, and no one can burn it except the owner itself. which will affect the Stability of the coin.

Recommendations: The idea will to give the liquidator less bonus in order to preserve, some collaterals in the position, for the DYAD that will be existed. Although it will be not efficient for the liquidator, as its bonus will decrease, it will prevent any existing of DYAD without collaterals.

The idea is simple, We will use the old liquidation logic which was used before, if this edge case occurs.

> VaultManagerV3::liquidate()

```
  function liquidate( ... )  ... {
+     uint cr = collatRatio(id);
      if (collatRatio(id) >= MIN_COLLAT_RATIO) revert CrTooHigh();
      ...

+         if (cr < 1.2e18 && debt != amount) {
+             uint cappedCr               = cr < 1e18 ? 1e18 : cr;
+             uint liquidationEquityShare = (cappedCr - 1e18).mulWadDown(0.2e18);
+             uint liquidationAssetShare  = (liquidationEquityShare +
1e18).divWadDown(cappedCr);
+             uint256 allAsset = vault.id2asset(id).mulWadUp(liquidationAssetShare);
+             asset = allAsset.mulWadDown(amount).divWadDown(debt);
+         }
           vault.move(id, to, asset);
      ...
  }
```

Sponsor: Fixed in PR-57

Al-Qa'qa': Verified. Fixing the issue done by decreasing liquidation ration to only 2% in case of partial liquidation when cr goes below *120%*.

4.1.2 Upgrading process uses initialize instead of reinitialize modifier

Severity: HIGH

Context: VaultManagerV3.sol#L198-L199

Description: The VaultManager is an upgradable contract and it is live on mainnet now. when first deploying the contract, we call *initialize(),* which is like *constructor()* for upgradable contracts. The issue lies is that the deployment script recalls initialize once again, which is like deploying the contract again, instead of upgrading it.

VaultManagerV3.sol#L51

```
function initialize( ... )
        public
@>              initializer
        { ... }
```

The *initializer* modifier, implements a mechanism like *constructor()*, where it can only be called once, ref.

Initializable.sol#L117-L122

```
        bool initialSetup = initialized == 0 && isTopLevelCall; // @audit        checks that the function is executed for the first time
        bool construction = initialized == 1 && address(this).code.length == 0;
        // @audit prevent reentrancy in construction

        if (!initialSetup && !construction) {
                revert InvalidInitialization();
        }
```

The deployment script is done by calling *initialize()* function again which contains *initialize* modifier. which will make the call reverts, because the value of *initialized != zero.*

DeployVaultManagerV3.s.sol#L22-L27

```
        Upgrades.upgradeProxy(
                MAINNET_V2_VAULT_MANAGER,
                "VaultManagerV3.sol",
                abi.encodeCall(
@>                      VaultManagerV3.initialize,
                        ( ... )
                )
        );
```

Recommendation: Change *initialize* modifier to *reinitialize* modifier, and pass the new version, which is *2* in that case, as the old version is *1*.

> VaultManagerV3.sol

```
function initialize( ... )
    public
-     initializer
+     reinitializer(2)
  { ... }
```

Sponsor: Fixed in PR-52
Al-Qa'qa': Verified. the recommended fix was implemented successfully.

4.2 Medium Risk

4.2.1 Liquidation process will get *Dos'ed* if *tvl* falls below *DYAD::totalSupply*

Severity: MEDIUM
Context: VaultManagerV3.sol#L202

Description: As stated in issue H-08 in Code4rena contest, if the *tvl* in the Vaults is less than the DYAD *totalSupply*. *KerosineVault::assetPrice()* will revert because of underflow, which will lead to reverting when withdrawing Kerosine.
This issue introduced two impacts. First, is that the kerosine will get locked until the *tvl* becomes greater than DYAD *totalSupply*. and the liquidation process will get Dos'ed, as the function will revert when trying to fire *KerosineVault::assetPrice()* because of underflow.
As stated in issue 244 in code4rena contest, the liquidation process will get *Dos'ed*, and the mitigation was to let the function return *0* if *tvl < totalSupply*.
The issue is that this mitigates the issue only for the old implementation of *liquidate()*, but for the new implementation of the *liquidate()* it will not mitigate it. and the liquidation will still be *Dos'ed*.

In the new implementation of *liquidate* function we are dividing by *vault.assetPrice()*, so the liquidate will still not be possible but instead of underflow error, it will revert because of division by zero error.

VaultManagerV3.sol#L200-L203

```
        uint asset = cappedValue
                * (10**(vault.oracle().decimals() + vault.asset().decimals())
@>              / vault.assetPrice()
                / 1e18;
```

As I stated, the main issue, which was in code4rena, introduces two impacts. temporal locked of kerosine, and liquidation *Dos'ed*. but in this case, only the liquidate function will get Dos'ed.

Recommendations: if *vault::assetPrice()* returned *0* skip the current for-loop iteration.

> VaultManagerV3::liquidate()

```
function liquidate( ... ) ... {
      ...
      for (uint i = 0; i < numberOfVaults; i++) {
        Vault vault = Vault(vaults[id].at(i));
        if (vaultLicenser.isLicensed(address(vault))) {
          uint value       = vault.getUsdValue(id);
+         if (value == 0) continue;
          ...
        }
      }

      emit Liquidate(id, msg.sender, to);
  }
```

Sponser: Fixed in PR-57, commit: 0acd8f1e748ee1c5a768cbaec556b6802fcd9d05
***Al-Qa'qa'*:** Verified. the recommended fix was implemented successfully.

4.2.2 Partial Liquidation Bonus is set wrongly

Severity: MEDIUM

Context: VaultManagerV3.sol#L198

Description: The Liquidation Bonus is *20%* of the bad position collaterals. So if the bad position minted DYAD is *1000* DYAD (USD), the liquidator will earn *1200* USD.

$$= 1000 / 2 + (1000 * 0.2) / 2 = 500 + 100 = 600\$$$
$$= 500 + 500 * 0.2 = 600\$$$

So We can calculate the liquidator value by taking the full *20%* from the amount he will pay or taking a percentage from the liquidation bonus, relative to the percentage of liquidation to the full position.
The problem is that the implementation performs both of them. where *liquidate()* function multiplies the partial amount to get liquidate with a partial liquidation rate, which makes the process incorrect.

VaultManagerV3.sol#L198

```
@>      uint reward_rate =          amount.divWadDown(debt).mulWadDown(LIQUIDATION_REWARD);
        ...
        uint amountShare = share.mulWadDown(amount);
@>      uint valueToMove = amountShare + amountShare.mulWadDown(reward_rate
                                                    ^^^^^^^^^^^
```

Recommendations: Either doing the partial to the bonus or the total amount. The following mitigation is done by reducing the amount.

```
-         uint valueToMove = amountShare + amountShare.mulWadDown(reward_rate);
+         uint valueToMove = amountShare +
amountShare.mulWadDown(LIQUIDATION_REWARD);
```

And this is if we will do it to the liquidation bonus.

```
-         uint valueToMove = amountShare + amountShare.mulWadDown(reward_rate);
+         uint valueToMove = amountShare + debt.mulAcknowledgedWadDown(reward_rate);
```

But I prefer the first one as it will not encounter rounding issues.

Sponser: Acknowledged, decreasing the bonus ratio for partial liquidation can be used as feature to incentivize full liquidation.

Al-Qa'qa': Although the docs said that liqudiation bonus is *20%*, decreasing this ratio with respect to the amount to be liquidated to the total position is a good feature, and it will encourage users to full liquidate bad positions.

4.2.3 *DeployVaultManagerV3* compilation will fail as *encodeCall()* expects two args

Severity: MEDIUM

context: This issue introduced when mitigating *H-02* issue.

Description: When mitigating issue *H-02*, the team chooses to not reinitialize the variables again. But the problem is that they let *encodeCall()* function takes only one argument, which is the function signature, and not pass the arguments in the second parameter.

```
        Upgrades.upgradeProxy(
                MAINNET_V2_VAULT_MANAGER,
                "VaultManagerV3.sol",
@>              abi.encodeCall(VaultManagerV3.initialize)
        );
```

The problem here is that *encodeCall()* takes only one argument, and it should takes two. The first for *functionPointer*, and the second is a tuble with args.

https://docs.soliditylang.org/en/latest/cheatsheet.html#abi-encoding-and-decoding-functions

> *abi.encodeCall(function functionPointer, (...)) returns (bytes memory):* ABI-encodes a call to *functionPointer* with the arguments found in the tuple. Performs a full type-check, ensuring the types match the function signature. Result equals *abi.encodeWithSelector(functionPointer.selector, ...)*

This will result in a compilation process fail, as the function should take two args, and the passed is just one.

Recommendations: Pass an empty tuble in the second arg.

```
   Upgrades.upgradeProxy(
      MAINNET_V2_VAULT_MANAGER,
      "VaultManagerV3.sol",
```

```
-     abi.encodeCall(VaultManagerV3.initialize)
+     abi.encodeCall(VaultManagerV3.initialize, ())
    );
```

Sponser: Fixed in PR-52, commit: 6124b8b03a8121eeb2aba7a88e967c0f2434b448

Al-Qa'qa': Verified. the recommended fix was implemented successfully.

4.3 Low Risk

4.3.1 Single-step ownership transfer mechanism by *OwnableUpgradeable*

Severity: LOW

Context: VaultManagerV3.sol#L19

Description: Single Step ownership transfer is dangerous as if the transfer is made to an incorrect address. the contract will be with no owner, and the role will be lost forever.
This will make the contract *non-upgradable*, where the owner is the only one who can upgrade the implementation of the *VaultManger*.

Recommendations: Use OZ::Ownable2StepUpgradeable. where in order to change the owner of the contract, the new owner should accept the new role in order to become the owner.

Sponser: Acknowledged

4.3.2 The new liquidation implementation does not round in the favour of the liquidator

Severity: LOW

Context: VaultManagerV3.sol#L197-L198

Description: *liquidate()* old implementation roundsUp (in the favour of liquidator), this is to secure the largest collateral transfer to the liquidator.

```
        uint collateral = vault.id2asset(id).mulWadUp(liquidationAssetShare
```

But in the new implementation, the rounding is Down when calculating shares and when calculating Assets to transfer.

VaultManagerV3.sol#L197-L198

```
@>      uint amountShare = share.mulWadDown(amount);
                uint valueToMove = amountShare + amountShare.mulWadDown(reward_rate);
                uint cappedValue = valueToMove > value ? value : valueToMove;
                uint asset = cappedValue
                        * (10**(vault.oracle().decimals() + vault.asset().decimals()))
                        / vault.assetPrice()
                        / 1e18;
```

Recommendations: Rounding *Up* instead of rounding *Down* when calculating *amountShare*.

> VaultManagerV3::liquidate()

```
-           uint amountShare = share.mulWadDown(amount);
-           uint valueToMove = amountShare + amountShare.mulWadDown(reward_rate);
+           uint amountShare = share.mulWadUp(amount);
+           uint valueToMove = amountShare + amountShare.mulWadUp(reward_rate);
```

Sponser: Fixed in PR-57, commit: 3f83712ae81bf296aa80ee374f256b06b03e4b36
Al-Qa'qa': Verified. the recommended fix was implemented successfully.

4.4 Informational

4.4.1 mappings key/value support names, no need to comment them

Severity: INFO
Context: VaultManagerV3.sol#L33

Description: There is an embedded comment in *lastDeposit* mappings that expresses the label for the mappings key and value. but solidity supports naming mapping itself without comments, so there is no need to comment them.

```
        mapping (uint/* id */ => uint/* block */) public lastDeposit;
```

Recommendations: remove comments.

```
-   mapping (uint/* id */ => uint/* block */)  public   lastDeposit;
+   mapping (uint id => uint block) public lastDeposit;
```

Sponser: Acknowledged.