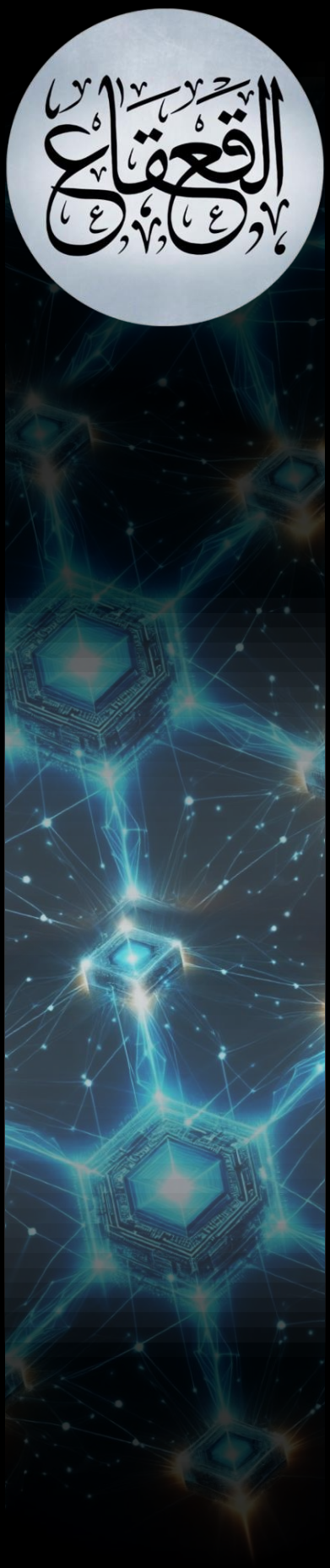
Al-Qa'qa’



DYAD

VaultManagerV5 Auditing Report

Auditor: Al-Qa'qa’

11 October 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 over-collateralization.

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

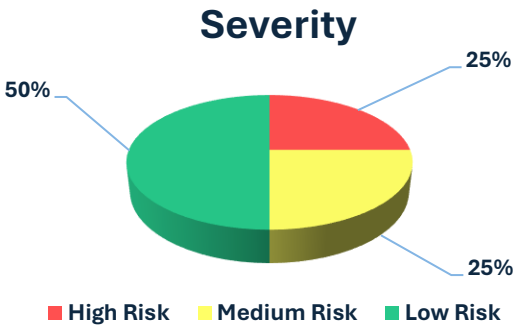
Project	**DYAD Stablecoin**
Repository	DyadStableCoin
Commit Hash	7a7229a83f6e8ffddf2a303a41aa80c70fe44642
Mitigation Hash	f4b1809ee60f932c1e4a13a81d3fc287e95f2ad8
Audit Timeline	05 Sept 2024 to 10 Sept 2024

2.2 Scope

- src/core/VaultManagerV5.sol
- src/periphery/WETHGateway.sol
- src/core/DyadHooks.sol

2.3 Issues Found

Severity	Count
High Risk	1
Medium Risk	1
Low Risk	2



3 Findings Summary

ID	Title	state
H-01	VaultManager minting, burning, and liquidating will get Dos'ed permanently till DyadXP upgraded	Fixed
M-01	An Attacker can grief Noteholders and prevent them from taking their staking rewards	Acknowledged
L-01	Authorized Extension can't remove Vaults	Fixed
L-02	Attackers can prevent users from removing these vaults	Fixed
I-01	`authorizeSystemExtension()` should use Hooks library when enabiling Hooks	Fixed
I-02	No check for the returned value from `OZ::EnumerableSet`	Fixed
I-03	unchecking for `address(0)` value in the construction	Acknowledged
I-04	`_authorizeCall` don't check the validity of the NFT	Acknowledged
I-05	There is no event emiting in `WETHGateway::redeemNative()`	Acknowledged
I-06	Anyone can deposit to any NoteId authorizing `WETHGateway`	Fixed
I-07	Liquidate Event lacks `amount` value	Fixed

4 Findings

4.1 High Risk

4.1.1 VaultManager minting, burning, and liquidating will get Dos'ed permanently till DyadXP upgraded

context:

- VaultManagerV5.sol#L154
- VaultManagerV5.sol#L194
- VaultManagerV5.sol#L262

Description:
Since the upgrading process will occur first, then `DyadXP` will get upgraded (it is still in development). The new features intended to be added to the `DyadXP` contract will not occur until it gets upgraded.

The problem is that the new Implementation of `VaultManagerV5` relies on the new implementation of the `DyadXP`, so this will result in reverting of all transactions that are calling the new `DyadXP` functions as they will not be added yet.

These functions are `mintDyad()`, `burnDyad()`, and `liquidate()`

VaultManagerV5.sol#L154 | VaultManagerV5.sol#L194 | VaultManagerV5.sol#L262

```
  function mintDyad( ... ) ... {
      uint256 extensionFlags = _authorizeCall(id);
      dyad.mint(id, to, amount); // changes `mintedDyad` and `cr`
>>    dyadXP.afterDyadMinted(id);
      ...
  }
  ...
  function _burnDyad(uint256 id, uint256 amount) internal {
      dyad.burn(id, msg.sender, amount);
>>    dyadXP.afterDyadBurned(id);
      emit BurnDyad(id, amount, msg.sender);
  }
  ...
  function liquidate( ... )  ... {
      ...

>>    dyadXP.afterDyadBurned(id);
      emit Liquidate(id, msg.sender, to);

      return (vaultAddresses, vaultAmounts);
  }
```

- Users will not be able to mint DYAD if they need to.
- Users will not be able to burn their DYAD from Their noteID to rescue liquidating for example.
- the liquidating process will get DoS'ed which will make the bad position unliquidatable till `DyadXP` gets upgraded.

This will affect the System and the stablecoin, especially the DoS of the liquidate function, which can make the undercollateralized position remain without liquidating affecting coin stability.

Recommendations:
Since `DyadXP` and `VaultManager` rely on each other, making the upgrade atomic is the right choice, this can be achieved easily by making MultiCall from Safe Smart contract wallet. Or In the case or the inability to make Multi call, `DyadXP` should get upgraded first.

Sponser: Fixed at: [9517383d2aeed8b24ab610b50a9ea75f450ae14e]
Al-Qa'qa': The issue has been fixed as recommended.

4.2 Medium Risk

4.2.1 An Attacker can grief Noteholders and prevent them from taking their staking rewards

context: [VaultManagerV5.sol#L187]

Description:
When burning DYAD from a given Note, anyone can burn his dyad tokens from any NoteID.

[VaultManagerV5.sol#L187] | [VaultManagerV5.sol#L194]

```
  function burnDyad( ... )  ... {
    _burnDyad(id, amount);
  }
  ...
  function _burnDyad(uint256 id, uint256 amount) internal {
    dyad.burn(id, msg.sender, amount);
>>  dyadXP.afterDyadBurned(id);
    emit BurnDyad(id, amount, msg.sender);
  }
```

This will make anyone decrease the amount of minted Dyad at any position.

Although the attacker will gain nothing when doing this in previous VaultManager versions, as this will increase the CR ratio of that NoteId, and will benefit the noteId hold. But in the new version, staking occurs, where the more note hold DYAD, the more it gains rewards.

After burning we call `dyadXP.afterDyadBurned()` to express the new amount of DYAD it holds. since the amount decreased this will decrease or prevent that ID from taking rewards. this will make anyone decrease the amount of rewards a given NoteId owner should take.

Proof of Concept:

- The note holder has some NoteId minted for his position.
- The amount of minted DYAD is `1000`
- The Attacker burned that `1000` DYAD from his position
- The note Holder will not get his reward.

Recommendations:
Prevent anyone from burning Dyad from a given position unless he is the owner or an authorized extension for that NFT.

```
diff --git a/src/core/VaultManagerV5.sol b/src/core/VaultManagerV5.sol
index eb7ae18..91ca305 100644
--- a/src/core/VaultManagerV5.sol
+++ b/src/core/VaultManagerV5.sol
@@ -99,9 +99,8 @@ contract VaultManagerV5 is IVaultManager, UUPSUpgradeable,
OwnableUpgradeable {
      uint256 id,
      address vault,
      uint256 amount
-   )
-     external isValidDNft(id)
-   {
+   ) {
+     _authorizeCall(id);
      Vault _vault = Vault(vault);
      _vault.asset().safeTransferFrom(msg.sender, vault, amount);
      _vault.deposit(id, amount);
```

Sponser: Acknowledged.

4.3 Low Risk

4.3.1 Authorized Extension can't remove Vaults

context: VaultManagerV5.sol#L88

Description:
When doing critical operations like adding Vaults, Minting Dyad, or withdrawing, we should check that the owner of that position is the caller or accepts this action.

The owner shouldn't be the caller at this case, as if the Owner accepts an extension it can do that task for him and we validate this with `_authorizeCall()` function.

All the Critical function, has that function, but in `remove()` function there is `isDNftOwner` modifier which will force the caller to be the owner himself. and we are also doing the

VaultManagerV5.sol#L88

```
  function remove(
      uint256 id,
      address vault
  )
    external
>>    isDNftOwner(id)
  {
>>  _authorizeCall(id);
    if (Vault(vault).id2asset(id) > 0) revert VaultHasAssets();
    if (vaults[id].remove(vault)) {
      emit Removed(id, vault);
    }
  }
```

This will make extensions the Position owner adds unable to remove vaults, which is not a desired thing.

Recommendations:
Remove `isDNftOwner` modifier, to allow extensions to remove vaults too.

```
diff --git a/src/core/VaultManagerV5.sol b/src/core/VaultManagerV5.sol
index eb7ae18..79a63e5 100644
--- a/src/core/VaultManagerV5.sol
+++ b/src/core/VaultManagerV5.sol
@@ -85,7 +85,6 @@ contract VaultManagerV5 is IVaultManager, UUPSUpgradeable,
OwnableUpgradeable {
        address vault
    )
      external
-       isDNftOwner(id)
    {
      _authorizeCall(id);
      if (Vault(vault).id2asset(id) > 0) revert VaultHasAssets();
```

Sponser: Fixed at: [9517383d2aeed8b24ab610b50a9ea75f450ae14e]

Al-Qa'qa': The issue has been fixed as recommended.

4.3.2 Attackers can prevent users from removing these vaults

context: [VaultManagerV5.sol#L103]

Description:
When removing a vault we are checking that amount of assets should be `0` before removing that Vault.

[VaultManagerV5.sol#L91)]

```
  function remove( ... ) ... {
    _authorizeCall(id);
>>  if (Vault(vault).id2asset(id) > 0) revert VaultHasAssets();
    if (vaults[id].remove(vault)) {
      emit Removed(id, vault);
    }
  }
```

The problem is that `deposit()` function is accessible to anyone. Anyone can deposit any amount to any position.

[VaultManagerV5.sol#L103]

```
  function deposit( ... )
>>  external isValidDNft(id)
  {
    Vault _vault = Vault(vault);
    _vault.asset().safeTransferFrom(msg.sender, vault, amount);
    _vault.deposit(id, amount);

    if (vault == KEROSENE_VAULT) {
      dyadXP.afterKeroseneDeposited(id, amount);
    }
  }
```

If the position id valid then anyone can deposit collaterals to it, which can be used by attackers to prevent users removing there Vaults.

- UserA wants to remove wETH vault.
- UserA withdrawed all his collaterals.
- UserA fired `VaultManager::remove()`
- Attacker FrontRunned him and deposit `1 wei`
- UserA transaction failed

This issue was found in Dyad Auditing contest in Code4rena before, link

Recommendations:
Allow only the owner or his authorized extensions to deposit to a specific NoteID.

```
diff --git a/src/core/VaultManagerV5.sol b/src/core/VaultManagerV5.sol
index eb7ae18..91ca305 100644
--- a/src/core/VaultManagerV5.sol
+++ b/src/core/VaultManagerV5.sol
@@ -99,9 +99,8 @@ contract VaultManagerV5 is IVaultManager, UUPSUpgradeable,
OwnableUpgradeable {
     uint256 id,
     address vault,
     uint256 amount
-  )
-    external isValidDNft(id)
-  {
+  ) {
+    _authorizeCall(id);
     Vault _vault = Vault(vault);
     _vault.asset().safeTransferFrom(msg.sender, vault, amount);
     _vault.deposit(id, amount);
```

Sponser: Fixed
at: 9517383d2aeed8b24ab610b50a9ea75f450ae14e, 7a55724ddb73b2ed7bb5a94fe16952c8ad0b4c01

Al-Qa'qa': Mitigation is not `100%` correct, more info at I-06

4.4 Informational Findings

4.4.1 `authorizeSystemExtension()` should use Hooks library when enabiling Hooks

context: VaultManagerV5.sol#L381

Description:
When activating/deactivating a given extension, admins call `authorizeSystemExtension()`. There is already a library that handles enabling and disability extentions, where for enability extentions we use `DyadHooks::enableExtension()` and for disability we use `DyadHooks::disableExtension()`.

DyadHooks.sol#L13-L19

```
    function enableExtension(uint256 flags) internal pure returns (uint256) {
        return flags | EXTENSION_ENABLED;
    }

    function disableExtension(uint256 flags) internal pure returns (uint256) {
        return flags & ~EXTENSION_ENABLED;
    }
```

We are using the library only when disability a given extension, and when enability we rewrite the enability logic ourselves.

VaultManagerV5.sol#L381)

```
  function authorizeSystemExtension(address extension, bool isAuthorized) external
onlyOwner {
    uint256 hooks;
    if (isAuthorized) {
>>    hooks = IExtension(extension).getHookFlags() | DyadHooks.EXTENSION_ENABLED;
    } else {
      hooks = DyadHooks.disableExtension(_systemExtensions[extension]);
    }
    _systemExtensions[extension] = hooks;
    emit SystemExtensionAuthorized(extension, hooks);
  }
```

We are not using our library when enabling Hooks, which makes the enable function in the Library useless.

This is not a good design, as in case of changing enability implementation in the library we should go and change it in the VaultManager too. which affects quality.

Recommendations:
Use the library enabling extension function instead of writing the logic of enabling again.

```
diff --git a/src/core/VaultManagerV5.sol b/src/core/VaultManagerV5.sol
index eb7ae18..1e0a768 100644
--- a/src/core/VaultManagerV5.sol
+++ b/src/core/VaultManagerV5.sol
@@ -378,7 +378,7 @@ contract VaultManagerV5 is IVaultManager, UUPSUpgradeable,
OwnableUpgradeable {
   function authorizeSystemExtension(address extension, bool isAuthorized)
external onlyOwner {
     uint256 hooks;
     if (isAuthorized) {
-      hooks = IExtension(extension).getHookFlags() | DyadHooks.EXTENSION_ENABLED;
+      hooks = DyadHooks.enableExtension(IExtension(extension).getHookFlags());
     } else {
       hooks = DyadHooks.disableExtension(_systemExtensions[extension]);
     }
```

Sponser: Fixed at: [3ecd6cdaccf633b149cf254e4e839fc7b6a17a73]

Al-Qa'qa': The issue has been fixed as recommended.

4.4.2 No check for the returned value from `OZ::EnumerableSet`

context:

- VaultManagerV5.sol#L368
- VaultManagerV5.sol#L370

Description:
Extentions used by users are stored in `EnumerableSet`. Users can either add or remove a given extension . The implementation of `OpenZeppelin` EnumerableSet is designed for returning a boolean value to express the state of the operation.

- When successful adding it return true, and if the item was already existed it returns false.
- When removing an item it returns true in case if was existed and removed, and false if it was not existed.

When a user call `authorizeExtension()` to either add or remove an extension, we are not checking for the returned value.

VaultManagerV5.sol#L368-L370

```
  function authorizeExtension(address extension, bool isAuthorized) external {
    if (isAuthorized) {
      if (!DyadHooks.hookEnabled(_systemExtensions[extension],
DyadHooks.EXTENSION_ENABLED)) {
        revert Unauthorized();
      }
>>    _authorizedExtensions[msg.sender].add(extension);
    } else {
>>    _authorizedExtensions[msg.sender].remove(extension);
    }
    emit UserExtensionAuthorized(msg.sender, extension, isAuthorized);
  }
```

So if the extension was existed when adding, or a user tried to remove an extension that is not exist, we will not be able to know the result of the operation.

Recommendations:
Take the returned value from the operation, and add it into the event `UserExtensionAuthorized`, so that the event emitting when users add/remove will have an info of weather add/remove took place or not.

Sponser: Fixed at: 50cb159c4de13e53652852d900f9f9a7405dd4d7

Al-Qa'qa': The issue has been fixed as recommended.

4.4.3 unchecking for `address(0)` value in the construction

context: WETHGateway.sol#L22-L29

Description:
When deploying `WETHGateway` extension, we are providing some addresses to initialize our contract with them. However, we are not checking if the parameters are set or not (not checking `address_zero`.

WETHGateway.sol#L22-L29

```
    constructor(address _dyad, address _dNft, address _weth, address
_vaultManager, address _wethVault) {
        dyad = IERC20(_dyad);
        dNft = IERC721(_dNft);
        weth = IWETH(_weth);
        vaultManager = IVaultManager(_vaultManager);
        wethVault = _wethVault;
        weth.approve(_vaultManager, type(uint256).max);
    }
```

This will result in a wrong deployment in case one of the parameters is not set.

Recommendations:
Check that parameters are not `address(0)` in the construction.

Sponser: Acknowledged.

4.4.4 `_authorizeCall` don't check the validity of the NFT

context: VaultManagerV5.sol#L423-L424

Description:
When doing actions like depositing/minting the caller should be either the owner of the NFT or an authorized extension. When doing this check we are not checking if the NFT token is existed or not (Owner is `address(0)`).

VaultManagerV5.sol#L423-L424

```
  function _authorizeCall(uint256 id) internal view returns (uint256) {
    address dnftOwner = dNft.ownerOf(id);
    if (dnftOwner != msg.sender) { ... }
    return 0;
  }
```

If the owner was not `msg.sender` we go directly for checking authorization, without checking the Validity of NFT first. This will make the function revert with a wrong error message `Unauthorized()` instead of `InvalidNFT()` which is not the right error message, as the NFT token doesn't even exist, and not the caller is Unauthorized.

Recommendations:
Check that the NFT is valid before checking authorization.

```
diff --git a/src/core/VaultManagerV5.sol b/src/core/VaultManagerV5.sol
index eb7ae18..46533ac 100644
--- a/src/core/VaultManagerV5.sol
+++ b/src/core/VaultManagerV5.sol
@@ -421,6 +421,7 @@ contract VaultManagerV5 is IVaultManager, UUPSUpgradeable,
OwnableUpgradeable {
   /// @param id The note id
   function _authorizeCall(uint256 id) internal view returns (uint256) {
     address dnftOwner = dNft.ownerOf(id);
+    if (dnftOwner == address(0)) revert InvalidDNft();
     if (dnftOwner != msg.sender) {
       uint256 extensionFlags = _systemExtensions[msg.sender];
       if (!DyadHooks.hookEnabled(extensionFlags, DyadHooks.EXTENSION_ENABLED)) {
```

Sponser: Acknowledged.

4.4.5 There is no event emiting in `WETHGateway::redeemNative()`

context: WETHGateway.sol#L74

Description:
When depositing or withdrawing, it was designed in `VaultManagerV4` to not emit events and depend on the events emitting from the vault we are withdrawing from.

in `VaultManagerV4::redeemDyad()` which is used to burn Dyad and receive collaterals, there is an event emitting occuar.

VaultManagerV4.sol#L171

```
  function redeemDyad( ... ) ...  {
      ...
      withdraw(id, vault, asset, to);
>>    emit RedeemDyad(id, vault, amount, to);
      return asset;
  }
```

In the `VaultManagerV5` this function is removed, but it is added as en extension and also in `WETHGateway`. But `WETHGateway` redeem function implementation don't emit `RedeemDyad` event.

WETHGateway.sol#L74

```
    function redeemNative(uint256 id, uint256 amount, address to) external {
        ...
        vaultManager.withdraw(id, wethVault, redeemedAmount, address(this));
        weth.withdraw(redeemedAmount);
        (bool success,) = to.call{value: redeemedAmount}("");
        if (!success) {
            revert WithdrawFailed();
        }
    }
```

This will make the new system work in a different way than the old system. Besides, Not emiting events is not good as it handles analyzing transactions history.

Recommendation:

emit Even when redeeming Dyad using `WETHGateway`

Sponser: Acknowledged.

4.4.6 Anyone can deposit to any NoteId authorizing `WETHGateway`

context: this issue introduced when mitigating issue `L-02`

Description:
When mitigating issue `L-02`, the `deposit()` function implementation restricts the deposit to only the Owner or the authorized extension.

WETHGateway.sol#L47-L50

```
    function depositNative(uint256 id) external payable {
        weth.deposit{value: msg.value}();
        vaultManager.deposit(id, wethVault, msg.value);
    }
```

So this will allow anyone who to deposit to any NoteId that authorized this extension.

Recommendation:

Since the design choice is to restrict the deposits to only Note Owners, we should restrict only the owner to be able to `depositNative()` ETH using `WETHGateway`.

Sponser: Fixed at PR-86, commit: 4dc6b5490f1124ea6e8c335b7cbf4dcf086784af

Al-Qa'qa': The issue has been fixed as recommended.

4.4.7 Liquidate Event lacks `amount` value

context: VaultManagerV5.sol#L250

Description:
When liquidating positions, Partial liquidation is supported now. So the liquidator can specify a specific amount to either liquidate All positions, half of it, quarter, etc...

But when finishing this process, and emitting an event, we are not specifying the exact amount of `Dyad` we burned (liquidated) from that user.

VaultManagerV5.sol#L250

```
  function liquidate(
    uint256 id,
    uint256 to,
>>  uint256 amount
  )  ... {
      ...
>>    dyad.burn(id, msg.sender, amount); // changes `debt` and `cr`
      ...
```

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

        return (vaultAddresses, vaultAmounts);
    }
```

Since in the old implementation, only full liquidation was supported, the amount was only the full position minted `DYAD`, so there was no need to put the amount of `DYAD` we burned. But now we should log/emit the amount of `DYAD` we burned, as different liquidations will result in the same event in that case.

Recommendations:
Add the amount of burned `DYAD`, the Liquidator paid to liquidate the targeted undercollateralized position.

Sponser: Fixed at PR-87, commit: 4adadaccda641895bc29fa08d3da2dbfc87c2bfe

Al-Qa'qa': The issue has been fixed as recommended.