DYAD Security Review

Auditor: Al-Qa'qa’

22 July 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

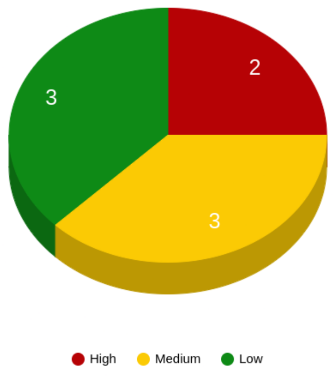
Project	DYAD Stablecoin
Repository	DyadStableCoin::feat/momentum
Commit Hash	a8245ea3671dfada7bd3845f2862f384a9294066
Mitigation Hash	85a3a6d91275ecbbc634978423f2f8677510ca80
Audit Timeline	8 July 2024 to 10 July 2024

2.2 Scope

- src/core/VaultManagerV4.sol
- Src/staking/DyadXP.sol
- script/deploy/Deploy.Momentum.s.sol

2.3 Issues Found

Severity	Count
High Risk	2
Medium Risk	3
Low Risk	3
Total Issues	**8**



3 Finding Summary

ID	title	status
H-01	users' XP initialization state can get manipulated as initializing users' XP and upgrading VaultManager is not atomic.	Resolved
H-02	Tracking kerosene internally logic introduced a lot of issues	Resolved
M-01	*totalKeroseneInVault is calculated wrongly in DyadXP::afterKeroseneDeposited()*	Resolved
M-02	*beforeKeroseneWithdrawn()* should be called before withdrawing in *VaultManagerV4*	Resolved
M-03	A malicious user can fund *KEROSENE_VAULT* making users rewards decrease	Resolved
L-01	The compilation process will fail as *Momentum* Contract no longer exists	Resolved
L-02	Incorrect Error Message respond in *DyadXP::approve()*	Resolved
L-03	initialize function should not take *DyadXP* as a parameter as deploying will occuar inside it	Resolved
I-01	using *totalMomentum* expression in *balanceOf()* is not ideal after changing the term to *XP*	Resolved
I-02	Upgrading comment is written V2 to V3 but this is V3 to V4 upgrade	Resolved
I-03	*VaultManager* on-chain version is not the same as the current version	Resolved
I-04	*__UUPSUpgradeable_init* is not called when initializing *DyadXP*	Acknowledged

4 Findings

4.1 High Risk

4.1.1 users' XP initialization state can get manipulated as initializing users' XP and upgrading VaultManager is not atomic.

Severity: HIGH

Context: DyadXP.sol#L50-L60

Description: The current *DyadXP*, changes users' State (XP), in the construction, where when the contract is deployed users' XP state is updated.

DyadXP.sol#L50-L60

```
constructor(address vaultManager, address keroseneVault, address dnft){
 ...

 for (uint256 i = 0; i < dnftSupply; ++i) {
  uint256 depositedKero = KEROSENE_VAULT.id2asset(i);
  if (depositedKero == 0) {
   continue;
  }
  noteData[i] = NoteXPData({
   lastAction: uint40(block.timestamp),
   keroseneDeposited: uint96(depositedKero),
   lastXP: 0
  });
 }
}
```

The problem is that if we check the deployment script, we will find that we deploy the new version *V4* (implementation, we do not upgrade it), and then we deploy Momentum (DyadXP) contract.

Deploy.Momentum.s.sol#L12-L23

```
/// @dev we do the upgrade manually through the multi-sig UI
VaultManagerV4 vm4 = new VaultManagerV4();
```

```
Momentum momentum = new Momentum(
  MAINNET_V2_VAULT_MANAGER,
  MAINNET_V2_KEROSENE_V2_VAULT,
  MAINNET_DNFT
);
```

As stated in the comment, the upgrading is done through a Multi-Sig wallet, which will be done after we deploy Momentum (DyadXP) contract, and when we just deploy DyadXP. we initialize users' XP.

Deploying through Multi-sig will take time (it took 2 days for testing and simulation when upgrading from V2 to V3 last time), so all deposit/withdraw/liquidate processes at this time will not affect users' state, which will make XP points calculations goes wrong.

Recommendations: All we need to do is to make initializing users' XP and upgrading VaultManager an atomic tx, each take place in the same execution.

Sponsor: We will do the deployment in the init function of the vault manager. Fixed in PR-68, commit: 6668ddc55ad8197f96c8c85873868d5e2dddfce5.
Al-Qa'qa': The fix is not *100%* correct, more info *L-03*.

4.1.2 Tracking kerosene internally logic introduced a lot of issues

Severity: HIGH
Context: This issue was introduced when mitigating issue *m-03*.

Description: There are are of issues in the internal tracking of Kerosene amount in kerosene vault. we will illustrate each of them separately.

1. initializing totalKeroseneInVault is not implemented

The mitigation of issue *m-03* has two parts, the first part was to set the variable *totalVaultKerosene* in the construction to be equal to the actual total kerosene tokens held by Liquidity Providers.

```
   constructor(address vaultManager, address keroseneVault, address dnft) {
        ...

        for (uint256 i = 0; i < dnftSupply; ++i) {
            uint256 depositedKero = KEROSENE_VAULT.id2asset(i);
            if (depositedKero == 0) { ... }
+           uint256 totalVaultkerosene += depositedKero;
            noteData[i] = NoteXPData({
                lastAction: uint40(block.timestamp),
                keroseneDeposited: uint96(depositedKero),
                lastXP: 0
            });
        }
    }
```

This check is not implemented in the mitigation, and it should be. It also exists on the main issue page *m-03*.

Recommendations: add this check to initialize the *totalVaultkerosene* with the right value when initializing users' XPs.

2. afterKeroseneDeposited() is not subtracting the newly deposited amount when calculating globalLastXP

As we illustrated in issue *m-01* when a user deposits new kerosine we calculate the totalXP gained in that interval using the totalkerosineValue before depositing, as the newly deposited amount sits in the Vault for *0* seconds, so we should now consider it when calculating totalXP.

```
@>  totalVaultKerosene += amountDeposited;


  …


  globalLastXP += uint192(
@>    (block.timestamp - globalLastUpdate) * totalVaultKerosene
  );
```

Recommendations: Subtract the *amountDeposited* from *totalVaultKerosene* when calculating *globalLastXP*

```
        globalLastXP += uint192(
-           (block.timestamp - globalLastUpdate) * totalVaultKerosene
+           (block.timestamp - globalLastUpdate) * (totalVaultKerosene -
amountDeposited)

        );
```

3. beforeKeroseneWithdrawn() is subtracting the amount before calculating globalLastXP

The amount of totalXP gained by the vault should be the amount of assets it has for the time interval. since the function is getting called before the actual withdrawing process, we was using *balanceOf()* state.

The problem is that we are subtracting the amount before calculating *globalLastXP*, which will make the assets to be withdrawn accumulate *0 XPs* although it has existed in the vault for a certain interval.

```
@> totalVaultKerosene -= amountWithdrawn;
  …
  globalLastXP = uint192(
@>   globalLastXP + (block.timestamp - globalLastUpdate) *     totalVaultKerosene - slashedXP
  );
```

Recommendations: Since the function will get fired before withdrawing, we should subtract the amount in the last of the function execution.

```
-       totalVaultKerosene -= amountWithdrawn;

       ...

       globalLastXP = uint192(
         globalLastXP + (block.timestamp - globalLastUpdate) * totalVaultKerosene -
slashedXP
        );
       globalLastUpdate = uint40(block.timestamp);
+       totalVaultKerosene -= amountWithdrawn;
```

Sponsor: Fixed in PR-72, commit: e43b0e4b58847cf7f56862802dd92ec58995647a.
Al-Qa'qa': Verified. The issue has been fixed as recommended.

4.2 Medium Risk

4.2.1 *totalKeroseneInVault* is calculated wrongly in *DyadXP::afterKeroseneDeposited()*

Severity: MEDIUM

Context: DyadXP.sol#L121-L124

Description: *DyadXP* is calculated by time-weighted algorism for kerosine tokens, and any process of deposit/withdraw/l from a vault changes the state of the target position as well as the global variables.

The problem is that *afterKeroseneDeposited()* function calculates the totalXP deserved (gained) wrongly as it should take the balance of Kerosine of the vault before the increase of the deposit amount, and multiply it with the interval. But the logic currently is not doing this.

DyadXP.sol#L121-L124

```
NoteXPData memory lastUpdate = noteData[noteId];
uint256 totalKeroseneInVault = KEROSENE.balanceOf(
   address(KEROSENE_VAULT)
) - lastUpdate.keroseneDeposited;
...
globalLastXP += uint192(
   (block.timestamp - globalLastUpdate) * totalKeroseneInVault
);
```

As we can see *totalKeroseneInVault* is calculated by subtracting the old user balance from the new totalSupply, and this is wrong, as we should subtract the newly deposited amount instead (New Balance - Old Balance).

Recommendations: Since the changing occurs after depositing (increasing vault supply), we should subtract the exact amount added by the user from that total supply and use it to determine the new *globalLastXP*.

```
diff --git a/src/staking/DyadXP.sol b/src/staking/DyadXP.sol
index 46d43ae..6356fc6 100644
--- a/src/staking/DyadXP.sol
+++ b/src/staking/DyadXP.sol
@@ -119,15 +119,17 @@ contract DyadXP is IERC20 {
```

```diff
        }

        NoteXPData memory lastUpdate = noteData[noteId];
+        uint96 noteIdNewBalance = uint96(KEROSENE_VAULT.id2asset(noteId));
+
        uint256 totalKeroseneInVault = KEROSENE.balanceOf(
            address(KEROSENE_VAULT)
-        ) - lastUpdate.keroseneDeposited;
+        ) - (noteIdNewBalance - lastUpdate.keroseneDeposited);

        uint256 newXP = _computeXP(lastUpdate);

        noteData[noteId] = NoteXPData({
            lastAction: uint40(block.timestamp),
-            keroseneDeposited: uint96(KEROSENE_VAULT.id2asset(noteId)),
+            keroseneDeposited: noteIdNewBalance,
            lastXP: uint120(newXP)
        });
```

NOTE: *noteIdNewBalance* should exceeds *lastUpdate.keroseneDeposited* as we are depositing (increasing) tokens, there is no normal scenario that allows firing this function without increasing funds.

Sponser: Fixed in PR-67, commit: 460a541fb6f42f495827b5c6a91064825fd2f818.
Al-Qa'qa': Verified. The issue has been fixed as recommended.

4.2.2 *beforeKeroseneWithdrawn()* should be called before withdrawing in *VaultManagerV4*

Severity: MEDIUM

Context: VaultManagerV3.sol#L198

Description: *DyadXP::beforeKeroseneWithdrawn()* is designed to get called before the actual withdrawing process, this should occur as it calculates the totalXP we will add to the *globalLastXP*. but in *VaultManagerV4::withdraw()*, it is getting fired after withdrawing process.

VaultManagerV4.sol#L115

```
  function withdraw( ... ) ... {
    if (lastDeposit[id] == block.number) revert CanNotWithdrawInSameBlock();
@>    Vault(vault).withdraw(id, to, amount); // changes `exo` or `kero` value and `cr`
    _checkExoValueAndCollatRatio(id);

@>    if (vault == KEROSENE_VAULT) momentum.beforeKeroseneWithdrawn(id, amount);
  }
```

As we can see, the function *momentum.beforeKeroseneWithdrawn()* is getting fired after withdrawing, this will make the calculations for *globalLastXP* give wrong results as the totalkerosineBalance will decrease results in fewer *globalLastXP* than expected.

DyadXP.sol#L171-L173

```
  function beforeKeroseneWithdrawn( ... ) external {
    ...
    NoteXPData memory lastUpdate = noteData[noteId];
@>    uint256 totalKeroseneInVault =      KEROSENE.balanceOf(address(KEROSENE_VAULT));
    ...
    globalLastXP = uint192(
@>        globalLastXP + (block.timestamp - globalLastUpdate) * totalKeroseneInVault - slashedXP
    );
    ...
  }
```

Recommendations: Fire *DyadXP::beforeKeroseneWithdrawn()* before calling *Vault::withdraw()*

```diff
diff --git a/src/core/VaultManagerV4.sol b/src/core/VaultManagerV4.sol
index b20f9f6..b208186 100644
--- a/src/core/VaultManagerV4.sol
+++ b/src/core/VaultManagerV4.sol
@@ -109,10 +109,10 @@ contract VaultManagerV4 is IVaultManager, UUPSUpgradeable,
OwnableUpgradeable {
       isDNftOwner(id)
   {
     if (lastDeposit[id] == block.number) revert CanNotWithdrawInSameBlock();
+    if (vault == KEROSENE_VAULT) momentum.beforeKeroseneWithdrawn(id, amount);
+
     Vault(vault).withdraw(id, to, amount); // changes `exo` or `kero` value and `cr`
     _checkExoValueAndCollatRatio(id);
-
-    if (vault == KEROSENE_VAULT) momentum.beforeKeroseneWithdrawn(id, amount);
```

```
  }

  function mintDyad(
```

Sponser: Fixed in PR-66, commit: dc34e41a42b2dd820bc065f4b7cc0778af22c9a4.
Al-Qa'qa': Verified. The issue has been fixed as recommended.

4.2.3 A malicious user can fund *KEROSENE_VAULT* making users rewards decrease

Severity: MEDIUM
context: DyadXP.sol#L65

Description: *DyadXP* directly reads the balance of *KEROSENE_VAULT* to know the amount of kerosene tokens in the vault, and this value is used to calculate *globalLastXP*.

```
function totalSupply() public view returns (uint256) {
  uint256 totalKerosene = KEROSENE.balanceOf(address(KEROSENE_VAULT));
  uint256 timeElapsed = block.timestamp - globalLastUpdate;
  return uint256(globalLastXP + timeElapsed * totalKerosene);
}
```

The problem is that anyone can fund *KEROSENE_VAULT* which will make *globalLastXP* increases significantly to the total users Balances. This will make *DyadXP::totalSupply* > All users DyadXPs.

This breaks invariants for ERC20 tokens which should make totalSupply == all balances of the users.

The problem is not just breaking invariants, increasing *globalLastXP* will make the ratio of user balance to the totalSupply decrease. So when distributing staking rewards, All users' rewards will get reduced as the divisor increases.
Recommendations: We should track the totalSupply of Kerosine in the *KEROSINE_VAULT* internally. this can be done by having a variable and this variable will be set when initializing DNFT notes. and when firing *afterKeroseneDeposited()* or *beforeKeroseneWithdrawn()* we will update the values.

Sponser: Fixed in PR-69, commit: 2005f110b5ba9c56965acd9a240eb14b3fc4c3a2.
Al-Qa'qa': The fix broke a lot of functionality, more info in *H-02*.

4.3 Low Risk

4.3.1 The compilation process will fail as *Momentum* Contract no longer exists

Severity: LOW
Context:

- VaultManagerV4.sol#L8
- Deploy.Momentum.s.sol#L8

Description: DYAD team has chosen to change the Name from *Momentum* to *DyadXP*, but *VaultManagerV4* and *DeployMomentum* still import it as *Momentum*. This will make compilation process fail.

VaultManagerV4.sol#L8
```
  import {Momentum} from "../staking/Momentum.sol";
```

Deploy.Momentum.s.sol#L8
```
  import {Momentum} from "../../src/staking/Momentum.sol";
```
This will make compilation process fail, as there is no file named *Momentum.sol* nor contract named *Momentum* now.

Recommendations: Change Importing to import *DyadXP*, instead of *Momentum*.

Sponser: Fixed in PR-65, commit: bd480f4aad7d073f1a399117163636fb393ea3e1.
Al-Qa'qa': Verified. The issue has been fixed as recommended.

4.3.2 Incorrect Error Message respond in *DyadXP::approve()*

Severity: LOW

Context: DyadXP.sol#L102-L104

Description: DyadXP is an ERC20 token, but it is not transferable, so the transferring process is disabled. there is also no support for approval, which is obvious as transferring is not supported. The problem is that the error message when trying to approve is the same as that when trying to transfer, which is *TransferNotAllowed()*.

DyadXP.sol#L102-L104

```
function approve(address, uint256) external pure returns (bool) {
    revert TransferNotAllowed();
}
```

Recommendations: Make another error message when calling approve(), like ApproveNotAllowed().

```
diff --git a/src/staking/DyadXP.sol b/src/staking/DyadXP.sol
index 46d43ae..0dcaeed 100644
--- a/src/staking/DyadXP.sol
+++ b/src/staking/DyadXP.sol
@@ -22,6 +22,7 @@ contract DyadXP is IERC20 {
     using FixedPointMathLib for uint256;

     error TransferNotAllowed();
+    error ApproveNotAllowed();
     error NotVaultManager();

     IVaultManager public immutable VAULT_MANAGER;
@@ -100,7 +101,7 @@ contract DyadXP is IERC20 {
     /// @notice Sets `amount` as the allowance of `spender` over the caller's 
tokens.
     /// @dev Be aware of front-running risks: 
https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     function approve(address, uint256) external pure returns (bool) {
-        revert TransferNotAllowed();
+        revert ApproveNotAllowed();
     }
```

Sponser: Fixed in PR-65, commit: 4cc6d78139f5023c9f8c871a84cc6222ec95b9bc.

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

4.3.3 initialize function should not take *DyadXP* as a parameter as deploying will occuar inside it

Severity: LOW

Context: This issue was introduced when mitigating *H-01*.

Description: The Sponsor chooses to make the deploying of DyadXP in the *initialize()* function to achieve the atomic process of deploying *DyadXP* and upgrading VaultManager. The problem is that this new function should not take the *DyadXP* contract as a parameter, as it is not deployed yet.

PR-68

```
            vvvvvvvvvvvvvvv
function initialize(DyadXP _dyadXP) ... {
  __UUPSUpgradeable_init();
  __Ownable_init(msg.sender);

  dyadXP = new DyadXP(
    address(this),
    0x4808e4CC6a2Ba764778A0351E1Be198494aF0b43,
    0xDc400bBe0B8B79C07A962EA99a642F5819e3b712 // MAINNET_DNFT
  );
}
```

Besides this, there is no change in the deployment script that prevents deploying *DyadXP* contract. So when deploying *VaultManagerV4* implementation contract we will deploy *DyadXP* too, which will be a useless contract as we will redeploy the version to use when upgrading VaultManager.

This will be a waste of money for the sponsor to deploy a useless contract on Mainnet.

Recommendations:

1. In the deployment script, remove the deploying code of *DyadXP* contract.
2. Remove that parameter (DyadXp) from *VaultManagerV4::initialize()* function.

Sponser: Fixed in PR-68, commit: 2890f0e6d54003a22606d4c75de429abd952120b.
Al-Qa'qa': Verified. the recommended fix was implemented successfully.

4.4 Informational

4.4.1 using *totalMomentum* expression in *balanceOf()* is not ideal after changing the term to *XP*

Severity: INFO
Context: DyadXP.sol#L71-L82

Description: In *DyadXP::balanceOf()*, we are using the term *totalMomentum* when calculating the total *XP* held by the user. This is not the right naming right now, as the contract name is DyadXP, and the function that computes the balance of the user called *_computeXP()*, so naming the variable *totalMomentum* is not ideal, and give no meaning in the current context.

DyadXP.sol#L71-L82

```
  function balanceOf(address account) external view returns (uint256) {
@>     uint256 totalMomentum;
     uint256 noteBalance = DNFT.balanceOf(account);

     for (uint256 i = 0; i < noteBalance; i++) {
       uint256 noteId = DNFT.tokenOfOwnerByIndex(account, i);
       NoteXPData memory lastUpdate = noteData[noteId];
@>       totalMomentum += _computeXP(lastUpdate);
     }

@>     return totalMomentum;
  }
```

Recommendations: Change the name from *totalMomentum* to *totalXP*.

```
diff --git a/src/staking/DyadXP.sol b/src/staking/DyadXP.sol
index 46d43ae..99529ae 100644
--- a/src/staking/DyadXP.sol
```

```diff
+++ b/src/staking/DyadXP.sol
@@ -69,16 +69,16 @@ contract DyadXP is IERC20 {

     /// @notice Returns the amount of tokens owned by `account`.
     function balanceOf(address account) external view returns (uint256) {
-        uint256 totalMomentum;
+        uint256 totalXP;
         uint256 noteBalance = DNFT.balanceOf(account);

         for (uint256 i = 0; i < noteBalance; i++) {
             uint256 noteId = DNFT.tokenOfOwnerByIndex(account, i);
             NoteXPData memory lastUpdate = noteData[noteId];
-            totalMomentum += _computeXP(lastUpdate);
+            totalXP += _computeXP(lastUpdate);
         }

-        return totalMomentum;
+        return totalXP;
     }

     function balanceOfNote(uint256 noteId) external view returns (uint256) {
```

Sponser: Fixed in PR-65, commit: 13c66e06a6c25a5b7c6156226d80634306249e49.
Al-Qa'qa': Verified. the recommended fix was implemented successfully.

4.4.2 Upgrading comment is written V2 to V3 but this is V3 to V4 upgrade

Severity: INFO

Context: VaultManagerV4.sol#L20

Description: In *VaultManagerV4*, it is writen that this is an upgrade for *VaultManagerV2*, but this is not true. as it is an upgrade for *VaultManagerV3*

```
/// @custom:oz-upgrades-from src/core/VaultManagerV2.sol:VaultManagerV2
```

Recommendations: Change it from VaultManagerV2 to VaultManagerV3.

```diff
diff --git a/src/core/VaultManagerV4.sol b/src/core/VaultManagerV4.sol
index b20f9f6..1d5b029 100644
```

```diff
--- a/src/core/VaultManagerV4.sol
+++ b/src/core/VaultManagerV4.sol
@@ -17,7 +17,7 @@ import {EnumerableSet} from "@openzeppelin/contracts/utils/struct
import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/Ownabl
import {UUPSUpgradeable}    from "@openzeppelin/contracts-upgradeable/proxy/utils/U

-/// @custom:oz-upgrades-from src/core/VaultManagerV2.sol:VaultManagerV2
+/// @custom:oz-upgrades-from src/core/VaultManagerV3.sol:VaultManagerV3
 contract VaultManagerV4 is IVaultManager, UUPSUpgradeable, OwnableUpgradeable {
   using EnumerableSet     for EnumerableSet.AddressSet;
   using FixedPointMathLib for uint;
```

Sponser: Fixed in PR-70, commit: 83e32546abd084fc7e42a1193c5a5f3333ff0e88.
Al-Qa'qa': Verified. the recommended fix was implemented successfully.

4.4.3 *VaultManager* on-chain version is not the same as the current version

Severity: INFO

Context: VaultManagerV4.sol#L53

Description: In upgradable contracts, we are providing a variable, which is the version, to the reinitializer modifier. The current version of VaultManager is V4. but in reinitializer, we provide 3.

VaultManagerV4.sol#L53

```
  function initialize(Momentum _momentum) public
@>     reinitializer(3)
  {
    __UUPSUpgradeable_init();
    __Ownable_init(msg.sender);


    momentum = _momentum;
  }
```

This will not affect deploying process, as the current version is 2. so any number greater than 2 will pass the check. But it is better to keep them (on-chain and off-chain versions match).

Recommendations: Pass 4 instead of 3 in the reinitializer modifier.

```
diff --git a/src/core/VaultManagerV4.sol b/src/core/VaultManagerV4.sol
index b20f9f6..9520459 100644
--- a/src/core/VaultManagerV4.sol
+++ b/src/core/VaultManagerV4.sol
@@ -50,7 +50,7 @@ contract VaultManagerV4 is IVaultManager, UUPSUpgradeable,
OwnableUpgradeable {

   function initialize(Momentum _momentum)
     public
-      reinitializer(3)
+      reinitializer(4)
   {
     __UUPSUpgradeable_init();
     __Ownable_init(msg.sender);
```

Sponser: Fixed in PR-68, commit: 6668ddc55ad8197f96c8c85873868d5e2dddfce5.
Al-Qa'qa': Verified. the recommended fix was implemented successfully.

4.4.4 __*UUPSUpgradeable_init* is not called when initializing *DyadXP*

Severity: INFO
Context: DyadXP.sol#L53-L54

Description: When initializing an upgradable contract, we should initialize all its children's (the contract he inherit from them) and do not leave them uninitialized.

DyadXP inherits from both *UUPSUpgradeable* and *OwnableUpgradeable*, but the only initialized one is *OwnableUpgradeable*.

DyadXP.sol#L53-L54

```
  function initialize(address owner) public initializer {
    __Ownable_init(owner);
    ...
  }
```

__*UUPSUpgradeable_init* is actually an empty function, and non-calling it has no security risk, but it is preferable to implement standards, as the contract may get upgraded in the future, and implement a logic in __*UUPSUpgradeable_init* function, so it is better to call it even if it is empty.

Recommendations: call *__UUPSUpgradeable_init* when initializing DyadXP.

```
diff --git a/src/staking/DyadXP.sol b/src/staking/DyadXP.sol
index 594da3c..c74c465 100644
--- a/src/staking/DyadXP.sol
+++ b/src/staking/DyadXP.sol
@@ -51,6 +51,7 @@ contract DyadXP is IERC20, UUPSUpgradeable, OwnableUpgradeable {
     }

     function initialize(address owner) public initializer {
+        __UUPSUpgradeable_init();
         __Ownable_init(owner);

         globalLastUpdate = uint40(block.timestamp);
```

Sponser: Acknowledged.