shieldify

DYAD

SECURITY REVIEW

Date: 2 June 2024

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About DYAD

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.

Each DYAD stablecoin is backed by at least $1.50 of exogenous collateral. This surplus absorbs the collateral's volatility, keeping DYAD fully backed in all conditions. Kerosene is a token that lets you mint DYAD against this collateral surplus. Kerosene can be deposited in Notes just like other collateral to increase the Note's DYAD minting capacity.

Learn more about DYAD v6's concept and the technicalities behind it here.

4. Risk Classification

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

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 3 days with a total of 24 hours dedicated to the audit by the Shieldify team.

Overall, the code is well-written. The audit report contributed by identifying one Critical and two High-severity issues, mostly related to collateral calculations due to missing asset decimal adjustments.

5.1 Protocol Summary

Project Name	**DYAD**
Repository	DyadStablecoin
Type of Project	DEX, First Capital Efficient Overcollateralized Stablecoin
Audit Timeline	3 days
Review Commit Hash	37b4d8bbbb59de52b25056fa8b9759203fe2bc1d
Fixes Review Commit Hash	68b4784ae1a62b9847738e23306ef0d4f40a9e46

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
src/core/VaultKerosene.sol	75
src/core/VaultManagerV2.sol	147
Total	**222**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical** and **High** issues: **3**
- **Medium** issues: **0**
- **Low** issues: **0**

ID	Title	Severity	Status
[C-01]	Non-compliance with EIP-7579	Critical	Fixed
[H-01]	The `redeemDyad()` Function Does Not Adjust Decimals Properlyd	High	Fixed
[H-02]	Insufficient Exogenous Collateral Check in `VaultManagerV2::liquidate()` function	High	Acknowledged

7. Findings

[C-01] Missing Asset Decimal Adjustment When Calculating TVL

Severity

Critical Risk

Description

When the TVL is being calculated in the `VaultKerosene.sol` contract the balance is multiplied with the oracle price, and adjusted with the oracle decimals.

```
tvl += vault.asset().balanceOf(address(vault))
    * vault.assetPrice()
    / (10**vault.oracle().decimals());
```

However, it does not adjust this based on the asset decimals. Furthermore, the `assetPrice` here is just a Chainlink oracle, as seen in the vault implementations.

```
function assetPrice() public view returns (uint256) {
    (, int256 answer,, uint256 updatedAt,) = oracle.latestRoundData();
    if (block.timestamp > updatedAt + STALE_DATA_TIMEOUT) revert
       StaleData();
    return answer.toUint256();
}
```

So if two vaults have `WBTC` and `WETH`, they will have different decimals and different amounts. But their price feeds will return the same decimals. So they will return a different scale of prices.

Say both `WBTC` and `WETH` are valued at 100 USD. So price feed returns 1e10 for both.

For 1e8 `WBTC tvl = 1e8 * 1e10 / 1e8 = 1e10`

For 1e18 `WETH: tvl = 1e18 * 1e10 / 1e8 = 1e20`

So `WBTC` is massively undervalued.

Location of Affected Code

File: src/core/VaultKerosene.sol#L113-L115

Recommendation

Adjust by asset decimals.

```
function assetPrice() public view override returns (uint) {
    uint tvl;
    address[] memory vaults = kerosineManager.getVaults();
    uint numberOfVaults = vaults.length;
    for (uint i = 0; i < numberOfVaults; i++) {
    Vault vault = Vault(vaults[i]);
    tvl += vault.asset().balanceOf(address(vault))
-       * vault.assetPrice()
-       / (10**vault.oracle().decimals());
+       * vault.assetPrice() * 1e18
+       / (10**vault.asset().decimals())
+       / (10**vault.oracle().decimals());
    }

    if (tvl < dyad.totalSupply()) return 0;
    uint numerator   = tvl - dyad.totalSupply();
    uint denominator = kerosineDenominator.denominator();
    return numerator * 1e8 / denominator;
}
```

Team Response

Fixed as proposed.

[H-01] The `redeemDyad()` Function Does Not Adjust Decimals Properly

Severity

High Risk

Description

The `redeemDyad()` function can be called to burn up DYAD tokens to free up collateral, and then pay out that collateral to the owner.

```
burnDyad(id, amount);
Vault _vault = Vault(vault);
uint asset = amount
    * 10**_vault.oracle().decimals()
    / _vault.assetPrice();
withdraw(id, vault, asset, to);
```

The issue is that this only works for assets which are in `18 decimals`. For assets like `USDC` `(6 decimals)` or `WBTC` `(8 decimals)`, the math is incorrect since the decimals are not adjusted.

For example, let's say a Chainlink oracle is used for a vault with `USDC` `(6 decimals)` and the price feed has `8 decimals`. So the price feed returns `1e8`.

Say the amount of DYAD repaid = `100 dollars` = `100e18`, since the DYAD is in `18 decimals`.

Then `asset` is calculated as = `100e18 * 1e8 / 1e8 = 100e18`

So `100e18 USDC` is going to be removed instead of `100e6`.

This can lead to reverts, or unintentional collateral withdrawals which can lead to unintentional liquidations.

Location of Affected Code

File: src/core/VaultManagerV2.sol#L163-L164

Recommendation

Adjust with the asset decimals.

```
function redeemDyad(
    uint id,
    address vault,
    uint amount,
    address to
) external isDNftOwner(id) returns (uint) {
    burnDyad(id, amount);
    Vault _vault = Vault(vault);
    uint asset = amount
-       * 10**_vault.oracle().decimals()
-       / _vault.assetPrice();
+       * (10**(_vault.oracle().decimals() + _vault.asset().decimals()))
+       / _vault.assetPrice()
+       / 1e18;
    withdraw(id, vault, asset, to);
    emit RedeemDyad(id, vault, amount, to);
    return asset;
}
```

Team Response

Fixed as proposed.

[H-02] Insufficient Exogenous Collateral Check in `VaultManagerV2::liquidate()` function

Severity

High Risk

Description

Code4rena issue #338 linked here.

The issue shows that liquidations do not go through if the exogenous collateral does not sufficiently back the `DYAD` minted.

The protocol lays down 2 ground rules:

1. Exo collateral backs `DYAD` at least 1:1 (100%)
2. Kerosene can be used to keep the minimum backing to 150%.

So for this to function properly, a user must have exo-backing of 100% and exo+kerosene backing of 150%. But if a user's exo-backing falls below 100% but their exo+kerosene backing is still above 150%, they won't get liquidated:

```
if (collatRatio(id) >= MIN_COLLAT_RATIO) revert CrTooHigh();
```

This can lead to systematic problems with collateral backing.For instance, if there is 1 million USD worth of exo collateral, and 1 million DYAD minted. Let's also say 600k USD worth of kerosene is in the vaults as well.

Now, Exo collateral backing = 1 million / 1 million = 100% Total backing = 1.6 / 1 = 160%.

Now, say the price of the exo collateral drops so there is only 950k USD worth of exo collateral left. Exo collateral backing = 950k / 1 million = 95% Total backing = 1.55 / 1 = 155 %.

If this was a single vault, this wouldn't be liquidatable since the CR is still above 150%.

The overall backing of the system is not 100% with exo collateral anymore. This can lead to people closing and withdrawing funds from their vaults, which further reduces the TVL of the system.

The main idea is that the system and individual vaults can reach a state where some of the `DYAD` is backed by kerosene, and not by other exo collateral. This would make it a fractionally collateralized stablecoin, like `FRAX` or `DEI`, both of which had stability issues and `FRAX` later voted to fully collateralize itself.

Location of Affected Code

File src/core/VaultManagerV2.sol#L179

Recommendation

Consider allowing liquidations when exo collateral backing goes below 100% as well. This will prevent the total exo collateral backing from going below 100% unless it is a large bad debt event.

Team Response

Acknowledged.

shieldify

Thank you!