yAudit Flower Review

Review Resources:

None beyond the code repositories.

Auditors:

- fedebianu
- adriro

Table of Contents

Review Summary

Flower

The Flower protocol comprises a series of smart contracts that implement a customized synthetic asset for use in Euler vaults.

The contracts of the Flower repository were reviewed over three days. Two auditors performed the code review between October 28th and October 30th, 2024. The repository was under active development during the review, but the review was limited to commit 19df79a3f3508be7255f6854c8f88f4c01ed14ad.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src
├── Allocator.sol
├── AutoScooper.sol
├── FlowerSynth.sol
├── FlowerSynthBackStopperWithdrawNFT.sol
├── FlowerSynthBackstopper.sol
├── interface
│   └── IPermit2.sol
├── lib
│   └── SafePermit2Lib.sol
└── token
    ├── FlowerToken.sol
    ├── PeriodicProportionalMinter.sol
    └── TimelockedMigrator.sol
```

After the findings were presented to the Flower team, fixes were made and included in PR#7.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Flower and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Authorization is implemented through ownable contracts and role-based access control.

Category	Mark	Description
Mathematics	Good	The code doesn't contain complex mathematical formulas. Some minor unsafe type casts were detected.
Complexity	Average	Even though logic is spread into multiple simple contracts, the double responsibility of FlowerSynthBackstopper.sol to act as a yield distributor and bad debt manager feels a bit too complex.
Libraries	Good	The codebase uses an updated version of the OpenZeppelin library.
Decentralization	Average	While on-chain activity is limited, the protocol relies on several privileged roles.
Code stability	Good	The codebase remained stable during the audit.
Documentation	Average	We recommended including high-level documentation about the intention and purpose of the protocol, what stability mechanism will be deployed to maintain the peg, and which entities will be assigned each role.
Monitoring	Good	Some events were missing, but overall, the contracts emit logs for key actions.
Testing and verification	Good	The repository contains a unit testing suite and an invariant test suite in development.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
  - Findings that can improve the gas efficiency of the contracts.

- Informational
  - Findings including recommendations and best practices.

Critical Findings

1. Critical - NFT sequence breaks after burning a token

The FlowerSynthBackstopperWithdrawNFT.sol contract uses the total supply of tokens to determine the next token ID to mint, breaking the sequence when a token with a different ID than the last one is burned.

Technical Details

When minting an NFT, the id of the new token is given by the `totalSupply()` function:

```
44:     function mint(address to) public onlyBackstopper returns (uint256) {
45:         uint256 id = totalSupply();
46:         _mint(to, id);
47:         return id;
48:     }
```

The contract also supports burning NFTs. When a token with an ID different from the last minted is burned, the total supply will be reduced by one, causing the next mint to fail due to a clash of IDs.

Proof of concept

```
function test_AUDIT_mint_burn() public asBackstopper {
    nft.mint(address(this)); // 0
    nft.mint(address(this)); // 1

    nft.burn(0);

    // fails
    nft.mint(address(this));
}
```

Impact

Critical. The issue could easily cause a denial of service during vault withdrawals in FlowerSynthBackstopper.sol.

Recommendation

Use an incremental sequence to determine the token ID.

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/91ce47aa7bf58669a3b64f058ba6b69b2d0569f9.

High Findings

None.

Medium Findings

1. Medium - `tokenURI()` does not comply with EIP-721

While separating the `tokenURI()` implementation into a dedicated contract enhances flexibility, the function must validate token existence as required by EIP-721.

Technical Details

`tokenURI()` relies on an external contract to provide URI data, but lacks validation to ensure the queried token exists before attempting to fetch its URI.

Impact

Medium. EIP standard violation. Incorrect data could be returned, leading to integration problems.

Recommendation

Add token existence validation before retrieving the URI:

```
     function tokenURI(uint256 id) public view override returns (string memory) {
+        _requireOwned(id);
         return ITokenURIImplementation(tokenURIImplementation).tokenURI(id);
     }
```

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/9cc497e4e2f840ae2f3714beaec54ca8276cf4b0.

Low Findings

1. Low - Backstopper can distribute yield when total supply is zero

Yield can still be distributed even if the total supply is zero.

Technical Details

The implementation of `interestAccruedFromCache()` can still return a positive yield when the smear period has elapsed, even if the total supply is zero.

```
354:     function interestAccruedFromCache(ESRSlot memory esrSlotCache) internal view
returns (uint256) {
355:         // If distribution ended, full amount is accrued
356:         if (block.timestamp >= esrSlotCache.interestSmearEnd) {
357:             return esrSlotCache.interestLeft;
358:         }
359:
360:         if(totalSupply() == 0) return 0;
```

Impact

Low. Interest can be accrued to dead shares.

Recommendation

Move the total supply check to the top of the function.

```
    function interestAccruedFromCache(ESRSlot memory esrSlotCache) internal view returns
(uint256) {
+       if(totalSupply() == 0) return 0;

        // If distribution ended, full amount is accrued
        if (block.timestamp >= esrSlotCache.interestSmearEnd) {
            return esrSlotCache.interestLeft;
        }

-       if(totalSupply() == 0) return 0;
```

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/c7a612a7f7b41a216064dc1f707b2b44ba63e859.

2. Low - Socializing small debts could be risky

`FlowerSynthBackstopper` contract does not enforce a minimum threshold for debt socialization, contrary to Euler's recommended implementation, which could lead to share price distortions in edge cases.

Technical Details

`executeBackstop()` checks if `liabilityValue` is zero to determine if socialization should occur. However, as suggested in the `EulerSavingsRate` implementation here, it is better to avoid small debt socialization.

Impact

Low. Socializing small debts when the vault is nearly empty could lead to share price distortions due to Oracle price rounding effects, potentially affecting all vault users. However, the specific conditions required reduce the likelihood of this scenario.

Recommendation

Add a check for `MIN_SOCIALIZATION_LIABILITY_VALUE` in `executeBackstop()` to prevent socialization of small debts.

```
+        if (collateralValue != 0 || liabilityValue < MIN_SOCIALIZATION_LIABILITY_VALUE)
{
-        if (collateralValue != 0 || liabilityValue == 0) {
             emit BackstopNoOp(vault_, account_, collateralValue, liabilityValue);
             return 0; // no bad debt
         }
```

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/15da990c370ac8cd68a002b9c3e39354e7771b8a.

3. Low - Unsafe type casts in Allocator.sol

The implementation utilizes unsafe type casts when processing the allocation delta.

Technical Details

The following check in `scoopDelta()` can be easily bypassed by overflowing `amount`:

```
117:         if (delta < int256(amount)) {
118:             revert E_InsufficientDelta();
119:         }
```

The calculation of the delta in `getAllocationDelta()` also does an unsafe type cast on `assetsBalance` and `currentAllocation`.

```
138:         return int256(assetsBalance) - int256(currentAllocation);
```

Impact

Low.

Recommendation

The `scoopDelta()` function only needs to handle a positive allocation delta, as a negative balance is already a no-op. Given this, the implementation can be greatly simplified by early exiting when `assetsBalance <= currentAllocation`, avoiding the need for `int256` handling.

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/4a38bbc54b691ab87dd1e6d505fb65968afd00c0.

4. Low - `PeriodicProportionalMinter` can't start minting from zero supply

Technical Details

The `PeriodicProportionalMinter` can't function as the sole minter for `FlowerToken` due to a mathematical edge case when the total supply is zero.

In `mint()`, when calculating `mintAmount`:

- First call will have totalSupply = 0
- This results in mintAmount = 0 * mintProportion / MINT_SCALE = 0
- Subsequent calls will also mint 0 tokens
- This creates a deadlock where no tokens can ever be minted

Impact

Low. The contract can't fulfill its intended purpose if it's the only minter.

Recommendation

Mint an initial supply of `FlowerToken` in the token constructor or change `mint()` function to handle the first mint:

```
     function mint() external {
         if (block.timestamp < startFrom) revert MINT_NOT_STARTED();
         if (block.timestamp < lastMint + mintFrequency) revert MINT_TOO_SOON();

         uint256 totalSupply = token.totalSupply();
+        uint256 mintAmount = totalSupply > 0 ? totalSupply * mintProportion /
MINT_SCALE : INITIAL_SUPPLY;
-        uint256 mintAmount = totalSupply * mintProportion / MINT_SCALE;
         lastMint = block.timestamp;
         token.mint(receiver, mintAmount);
     }
```

Developer Response

Acknowledged. On deployment, the initial token supply will be minted separately to start with a non-zero supply.

Gas Saving Findings

1. Gas - Optimize for loops

Technical Details

For loops can be optimized to save gas.

Impact

Gas savings.

Recommendation

Optimize for loops by applying these changes:

- don't initialize the loop counter variable, as there is no need to initialize it to zero because it is the default value

- cache the array length outside a loop as it saves reading it on each iteration
- increment `i` variable with `++i` statement

Developer Response

Acknowledged. Will keep as is.

2. Gas - Cache storage variable locally to prevent multiple reads from storage

Cache variables read from storage to prevent multiple SLOAD operations.

Technical Details

- `FlowerSynthBackstopper::_deposit()`: `_totalAssets` is read again to emit the `TotalAssetsUpdated` event.
- `FlowerSynthBackstopper::_witdraw_()`: `_totalAssets` is read again to emit the `TotalAssetsUpdated` event. The same happens with `_totalAssetsPendingWithdraw` to emit the `TotalAssetsPendingWithdrawUpdated` event.
- `FlowerSynthBackstopper::redeemWithdrawNFT()`: `withdrawNFTData[id].sharesAmount` is read twice in lines 251 and 252. `_totalAssetsPendingWithdraw` is read again to emit the `TotalAssetsPendingWithdrawUpdated` event.
- `FlowerSynthBackstopper::executeBackstop()`: both `_totalAssets` and `_totalAssetsPendingWithdraw` are read from storage multiple times across the implementation.

Impact

Gas savings.

Recommendation

Cache state in local variables instead of reading again from storage.

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/136900a5bb8a5508dcbf22af608fbbb3b0546208.

3. Gas - `onlyTransferAllowedRole()` can be optimized using De Morgan's Laws

Technical Details

The `onlyTransferAllowedRole()` modifier uses two NOT operators with AND. Applying De Morgan's Laws can save gas by converting this to a single OR operation with one NOT.

```
Gas Report for FlowerToken Test Suite

FlowerTokenTest:test_transferAllowedRole() (gas: 344229) -> (gas: 344216) [-13 gas]
FlowerTokenTest:test_transferFromAllowedRole() (gas: 351465) -> (gas: 351452) [-13 gas]
FlowerTokenTest:test_transferFromRevertNotAllowed() (gas: 246017) -> (gas: 246004) [-13 gas]
FlowerTokenTest:test_transferRevertNotAllowed() (gas: 219679) -> (gas: 219666) [-13 gas]
FlowerTokenTest:test_wildCardTransferAllowed() (gas: 342689) -> (gas: 342680) [-9 gas]
```

Impact

Gas savings.

Recommendation

Apply De Morgan's Laws to save gas.

```
     modifier onlyTransferAllowedRole() {
+        if (!(hasRole(TRANSFER_ALLOWED_ROLE, WILD_CARD_ADDRESS) ||
hasRole(TRANSFER_ALLOWED_ROLE, _msgSender()))) {
-        if (!hasRole(TRANSFER_ALLOWED_ROLE, WILD_CARD_ADDRESS) &&
!hasRole(TRANSFER_ALLOWED_ROLE, _msgSender())) {
             revert TRANSFER_NOT_ALLOWED();
         }
         _;
     }
```

Developer Response

Acknowledged. Will keep as is. Personal preference with regards to readability.

Informational Findings

1. Informational - Fix typos

Technical Details

src/lib/SafePermit2Lib.sol.sol#17

```
+    /// @dev Transfer from using Permit2. If `permit2 == address(0)` this will use the normal `ERC20.transferFrom()` call.
-    /// @dev Transfer from using Permit2. If `permit2 == address(0)` this will use the normale `ERC20.transferFrom()` call.
```

Impact

Informational.

Recommendation

Fix typos.

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/b375d7aef857ebdb47b523273561af5add650a67.

2. Informational - Remove unused imports

Technical Details

- `ERC20` import is unused.
- `IEVault` import is unused.

Impact

Informational.

Recommendation

Remove unused imports.

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/fe891a33a87b51e59c561a063d644218705a6d59.

3. Informational - Remove unused parameters

Technical Details

In FlowerToken.sol constructor `permit2_` address is not used.

Impact

Informational.

Recommendation

Remove unused parameters.

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/984f707d65aaf29129b22c46e311d7800ad62693.

4. Informational - Allocations in AutoScooper.sol will fail before reaching the maximum

The maximum check on the allocation length is performed inclusively, causing the validation to fail when the count equals the limit.

Technical Details

The validation to check if the maximum number of allocations has been reached is done using a `>=` operator, which will also revert when the number of elements is at the limit.

```
63:         // If now zero remove from allocations, otherwise set
64:         if (points == 0) {
65:             allocations.remove(receiver);
66:         } else {
67:             allocations.set(receiver, points);
68:         }
69:
70:         if (allocations.length() >= MAX_ALLOCATION_LENGTH) revert
E_MaxAllocationReached();
```

Impact

Informational.

Recommendation

```
-   if (allocations.length() >= MAX_ALLOCATION_LENGTH) revert E_MaxAllocationReached();
+   if (allocations.length() > MAX_ALLOCATION_LENGTH) revert E_MaxAllocationReached();
```

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/81cf31a5c920f47a53da93f37ddfb6fb7d2a35a0.

5. Informational - `FlowerSynth` tokens are incompatible with Euler synthetic vaults

Euler docs states:

> Euler synthetic vaults are a special configuration of vaults which use hooks to disable deposit-related operations for all addresses except the synth address itself.

Given this, the current implementation is incompatible with Euler synthetic vaults due to architectural differences in deposit handling. According to Euler's documentation, synthetic vaults only accept deposits from the synth contract itself, but Flower's allocation logic has been moved to a separate Allocator contract.

Technical Details

Euler's synthetic vaults implement strict deposit control in HookTargetSynth.sol's `deposit()` allowing deposits only from the asset contract itself. As `allocate()` calls vault deposit from an external contract, the call will fail.

The `FlowerSynth` contract cannot interact with Euler synthetic vaults, limiting its intended functionality and potential integrations.

Impact

Informational.

Recommendation

Consider implementing one of these approaches:

1. Move allocation logic back to the `FlowerSynth` contract, allowing it to be called only by the `Allocator` contract
2. Implement a proxy mechanism where calls to Euler vaults are forwarded through the `FlowerSynth` contract

Developer Response

Acknowledged. Intended to be used with the following hook: HookTargetAccessControl.sol only to allow the Allocator to deposit.

6. Informational - Remove `amount` from `Scooped` event

Technical Details

The `amount` parameter in the `Scooped` event is misleading as the `scooped` parameter already represents the actual scooped value.

Impact

Informational.

Recommendation

Delete `amount` parameter from the `Scooped` event.

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/365d6ebcd8098cf5c1c48bf7ea52b954cccf69d3.

7. Informational - Missing event for state change

Parameter or configuration changes should trigger an event to allow being tracked off-chain.

Technical Details

- `setTokenURIImplementation()`
- `setReceiver()`
- `addIgnoredForTotalSupply()`
- `removeIgnoredForTotalSupply()`

Impact

Informational.

Recommendation

Add events to the functions listed above.

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/b0e20e3da431b7b5b5e89512c059374fadbeb2b7.

8. Informational - Occurrences of `msg.sender` in EVCUtil contracts

There are some usages of the direct caller in contracts that are intended to be compatible with the EVC.

Technical Details

- FlowerSynth.sol#L45
- FlowerSynthBackStopperWithdrawNFT.sol#L23

Impact

Informational.

Recommendation

Although the EVC is unlikely to wrap these instances, for consistency, consider using `_msgSender()` instead.

Developer Response

Fixed: https://github.com/MickdeGraaf/flower-contracts/commit/ec14dd29490d95397e124efd8b66404b7547e6e5.

Final Remarks

The Flower protocol builds on top of the synthetic core contracts of the Euler EVK to provide a custom allocation mechanism, which is further extended by an automatic bad debt backstopping system.

No significant issues were found other than the accidental assumption detailed in C-1. While it is reasonable to have multiple dependencies on privileged roles to potentially control the peg of the synthetic asset, the auditors recommend providing greater clarity about the overall

solution, explicitly detailing the components to be deployed and the accounts or contracts with privileged access.