// HALBORN

Dex Finance - DexFiVaults V3

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **November 28th, 2023 - January 5th, 2024**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	01/05/2024
0.2	Document Updates	01/05/2024
0.3	Final Draft	01/08/2024
0.4	Draft Review	01/08/2024
0.5	Draft Review	01/08/2024
1.0	Remediation Plan	01/12/2024
1.1	Remediation Plan Review	01/13/2024

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Dex Finance engaged Halborn to conduct a security assessment on their smart contracts beginning on November 28th, 2023 and ending on January 5th, 2024. The security assessment was scoped to the smart contracts provided in the DeFi-Gang/DexFi-Vaults GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided 5 weeks for the engagement and assigned a team of 1 full-time security engineer to review the security of the smart contracts in scope. The security team consists of a blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment was to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks, that were successfully addressed by Dex Finance. The main ones were the following:

- Missing storage gaps.
- The use of arrays to store farms can lead to DOS.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (Slither).
- Testnet deployment (Foundry, Brownie).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Project Name

- Repository: DeFi-Gang/DexFi-Vaults
- Commit ID: ea9512ab041fc30811314e84536b89f63f8dd0a5
- Smart contracts in scope:
    1. DexFiVault.sol (core/DexFiVault.sol)
    2. DexFiVaultFactory.sol (core/DexFiVaultFactory.sol)
    3. DexFiVaultHarvester.sol (core/DexFiVaultHarvester.sol)
    4. DexFiVaultMigrator.sol (core/DexFiVaultMigrator.sol)
    5. DexFiVaultProfitClaimer.sol (core/DexFiVaultProfitClaimer.sol)
    6. DexfiVaultProfitStorage.sol (core/DexFiVaultProfitStorage.sol)
    7. DexFiVaultZapper.sol (core/DexFiVaultZapper.sol)
    8. DexFiInitializable.sol (core/abstract/utils/DexFiInitializable.sol)
    9. DexFiFarm.sol (core/abstract/DexFiFarm.sol)
    10. DexFiProfit.sol (core/abstract/DexFiProfit.sol)
    11. DexFiZapperToken.sol (core/abstract/DexFiZapperToken.sol)

Out-of-scope

- Third-party libraries and dependencies.
- Economic attacks.

REMEDIATION COMMIT IDs :

- 402dad1b9cfa3dfc90f55cdfd027643a9ec8d073

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	0	0

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) TOO MUCH FARMS CAN LEAD TO DOS	Medium (4.5)	SOLVED - 01/12/2024
(HAL-02) MISSING STORAGE GAPS	Medium (5.0)	SOLVED - 01/12/2024

FINDINGS & TECH DETAILS

4.1 (HAL-01) TOO MUCH FARMS CAN LEAD TO DOS - MEDIUM (4.5)

Description:

The farms that a vault uses in the DexFiVault contract are stored in array and are frequently looped for multiple operations such as depositing or harvesting. However, if there are too many farms, it can reach a point where the gas cost of harvesting or depositing into those farms may become bigger than the block gas limit.

This would lead to a denial of service as users wouldn't be able to properly use the vault functions.

Code Location:

Listing 1: DexFi-Vaults/contracts/core/DexFiVault.sol (Line 158)

```
158 for (uint256 i = 0; i < _farms.length; i++) {
159     Farm storage farm_ = _farms[i];
160     IDexFiFarm connector = farmConnector[farm_.beacon];
161     uint256 preDepositFarmAmount = connector.liquidity(address(
 ↳ connector));
162     uint256 currentFarmNativeAmount = i == _farms.length - 1
163         ? amount - amountSum
164         : (amount * farm_.percent) / divider;
165     amountSum += currentFarmNativeAmount;
166     if (currentFarmNativeAmount > 0) {
167         native.forceApprove(address(connector),
 ↳ currentFarmNativeAmount);
168         connector.deposit(currentFarmNativeAmount);
169     }
170     uint256 postDepositFarmAmount = connector.liquidity(address(
 ↳ connector));
171     preDepositTotalFarmsNativeInvestments +=
172         (currentFarmNativeAmount * preDepositFarmAmount) /
173         (postDepositFarmAmount - preDepositFarmAmount);
174 }
```

BVSS:

AO:A/AC:M/AX:M/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (4.5)

Recommendation:

Limit the number of farms that can be used in a vault to a value low enough that there is no risk of running out of gas.

Remediation Plan:

SOLVED: The Dex Finance team solved the issue by limiting farms with maxFarmsCount.

Commit ID: 402dad1b9cfa3dfc90f55cdfd027643a9ec8d073

4.2 (HAL-02) MISSING STORAGE GAPS - MEDIUM (5.0)

Description:

The DexFiFarm, DexFiProfit and DexFiZapperToken contracts are abstract contracts that are meant to be inherited from. Moreover, they implement a custom initialization mechanism, it is important to note that if these contracts are meant to be used by upgradeable contracts storage gaps should be added to the contract itself, to prevent storage collisions in the case of upgrading the implementation.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (5.0)

Recommendation:

Add storage gaps to the abstract contracts.

Remediation Plan:

SOLVED: The Dex Finance team solved the issue by adding storage gaps on the contracts.

Commit ID: 402dad1b9cfa3dfc90f55cdfd027643a9ec8d073

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.
The security team assessed all findings identified by the Slither software, however, findings with severity Information and Optimization are not included in the below results for the sake of report readability.

Results:

Slither results for contracts	
Finding	**Impact**

AUTOMATED TESTING

Finding	Impact
Reentrancy in DexFiVault.publish() (src/DexFiVault.sol#381-390): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#382) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) - _depositAll() (src/DexFiVault.sol#384) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- native.forceApprove(address(connector),currentFarmNativeAmount) (src/DexFiVault.sol#536) - connector.deposit(currentFarmNativeAmount) (src/DexFiVault.sol#537) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- _depositAll() (src/DexFiVault.sol#384) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - renounceOwnership() (src/DexFiVault.sol#387) - _owner = newOwner (lib/openzeppelin-contracts-upgradeable/contrac	High

AUTOMATED TESTING

Finding	Impact
Reentrancy in DexFiVault.updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#273-281): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#274) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#275) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) - _withdrawAndSwapToNative(totalSupply()) (src/DexFiVault.sol#276) - nativeAmount += connector.withdraw(amount) (src/DexFiVault.sol#554) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#275) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - delete _farms (src/DexFiVault.sol#277) DexFiVault._farms (src/DexFiVault.sol#30) can be used in cross function reentrancies: - DexFiVault._actualizeFarmsStatus() (src/DexFiVault.sol#414-427) - DexFiVault._depositAll() (src/DexFiVault.sol#524-541) - DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618) - DexFiVault._updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#485-507) - DexFiVault.convertlessDepositRatio() (src/DexFiVault.sol#61-67) - DexFiVault.farms(uint256) (src/DexFiVault.sol#45-47) - DexFiVault.farmsCount() (src/DexFiVault.sol#53-55)	High

AUTOMATED TESTING

Finding	Impact
Reentrancy in DexFiVault.withdrawConvertless(uint256) (src/DexFiVault.sol#342-375): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#346) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#347) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) - feeAmounts = _withdrawConvertless(feeSyntheticAmount,treasury) (src/DexFiVault.sol#354) - connector.withdrawConvertless(outputAmounts[i],recipient) (src/DexFiVault.sol#574) - native.safeTransfer(treasury,nativeFeeAmount) (src/DexFiVault.sol#357) - _burn(msg.sender,feeSyntheticAmount) (src/DexFiVault.sol#360) - IDexFiVaultProfitStorage(profitStorage).updateUsersSharesBalance(updateInfo) (src/DexFiVault.sol#476) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#347) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - _burn(msg.sender,feeSyntheticAmount) (src/DexFiVault.sol#360) - _totalSupply -= amount (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#292)ERC20Upgradeable._totalSupply (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#44) can be used in cross function reentrancies: - ERC20Upgradeable.totalSupply() (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#99-101)	High

AUTOMATED TESTING

Finding	Impact
Reentrancy in DexFiVault.updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#273-281): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#274) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#275) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) - _withdrawAndSwapToNative(totalSupply()) (src/DexFiVault.sol#276) - nativeAmount += connector.withdraw(amount) (src/DexFiVault.sol#554) - _updateFarms(farms_) (src/DexFiVault.sol#278) - connector = IDexFiFarm(address(new BeaconProxy(farm_.beacon,))) (src/DexFiVault.sol#493) - ! connector.initialize(farm_.data) (src/DexFiVault.sol#494) - ! farmConnector_.reinitialize(farm_.data) (src/DexFiVault.sol#498) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#275) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - _updateFarms(farms_) (src/DexFiVault.sol#278) - _farms.push(farm_) (src/DexFiVault.sol#503) DexFiVault._farms (src/DexFiVault.sol#30) can be used in cross function reentrancies: - DexFiVault._actualizeFarmsStatus() (src/DexFiVault.sol#414-427) - DexFiVault._depositAll() (src/DexFiVault.sol#524-541) - DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618) - DexFiVault._updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#485-507)	High

AUTOMATED TESTING

Finding	Impact
Reentrancy in DexFiVault.depositConvertless(uint256[],uint256) (src/DexFiVault.sol#189-235): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#194) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#195) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) - _depositAll() (src/DexFiVault.sol#196) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- native.forceApprove(address(connector),currentFarmNativeAmount) (src/DexFiVault.sol#536) - connector.deposit(currentFarmNativeAmount) (src/DexFiVault.sol#537) - Address.functionCall(connector_scope_1.stakingToken(),connector_scope_1.stakingTokenTransferFromData(msg.sender,address(this),stakingAmounts[i_scope_0])) (src/DexFiVault.sol#215-218) - Address.functionCall(connector_scope_1.stakingToken(),connector_scope_1.stakingTokenApproveData(address(connector_scope_1),stakingAmounts[i_scope_0])) (src/DexFiVault.sol#219-222) - connector_scope_1.depositConvertless(stakingAmounts[i_scope_0],depositLiquidityAmounts[i_scope_0],feeLiquidityAmounts[i_scope_0],factory,integrationConfig(),treasury,msg.sender)	High

AUTOMATED TESTING

Finding	Impact
Reentrancy in DexFiVault.withdraw(uint256,address,uint256) (src/DexFiVault.sol#290-315): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#296) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#297) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) - nativeAmount += _withdrawAndSwapToNative(amount) (src/DexFiVault.sol#302) - nativeAmount += connector.withdraw(amount) (src/DexFiVault.sol#554) - native.safeTransfer(integrationConfig.treasury,withdrawFeeAmount) (src/DexFiVault.sol#307) - native.safeTransfer(msg.sender,nativeAmount) (src/DexFiVault.sol#311) - _burn(from,amount) (src/DexFiVault.sol#312) - IDexFiVaultProfitStorage(profitStorage).updateUsersSharesBalance(updateInfo) (src/DexFiVault.sol#476) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#297) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - _burn(from,amount) (src/DexFiVault.sol#312) - _totalSupply -= amount (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#292)ERC20Upgradeable._totalSupply (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#44) can be used in cross function reentrancies:	High

AUTOMATED TESTING

Finding	Impact
Reentrancy in DexFiVault.withdrawConvertless(uint256) (src/DexFiVault.sol#342-375): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#346) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#347) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) - feeAmounts = _withdrawConvertless(feeSyntheticAmount,treasury) (src/DexFiVault.sol#354) - connector.withdrawConvertless(outputAmounts[i],recipient) (src/DexFiVault.sol#574) - native.safeTransfer(treasury,nativeFeeAmount) (src/DexFiVault.sol#357) - _burn(msg.sender,feeSyntheticAmount) (src/DexFiVault.sol#360) - IDexFiVaultProfitStorage(profitStorage).updateUsersSharesBalance(updateInfo) (src/DexFiVault.sol#476) - native.safeTransfer(msg.sender,userNativeAmount) (src/DexFiVault.sol#363) - outputAmounts = _withdrawConvertless(amount,msg.sender) (src/DexFiVault.sol#364) - connector.withdrawConvertless(outputAmounts[i],recipient) (src/DexFiVault.sol#574) - _burn(msg.sender,amount) (src/DexFiVault.sol#365) - IDexFiVaultProfitStorage(profitStorage).updateUsersSharesBalance(updateInfo) (src/DexFiVault.sol#476) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#347) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.so	High

Finding	Impact
Reentrancy in DexFiVaultProfitStorage.updateProfitToken(address) (src/DexFiVaultProfitStorage.sol#191-197): External calls: - _claim(vaultOwner,availableToClaim[vaultOwner]) (src/DexFiVaultProfitStorage.sol#194) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#122)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#135)- IERC20(profitTokenConnector.underlying()).safeTransfer(user,amount) (src/DexFiVaultProfitStorage.sol#207) External calls sending eth: - _claim(vaultOwner,availableToClaim[vaultOwner]) (src/DexFiVaultProfitStorage.sol#194) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#135)State variables written after the call(s): - _updateProfitToken(profitToken_) (src/DexFiVaultProfitStorage.sol#195) - profitToken = profitToken_ (src/DexFiVaultProfitStorage.sol#217)DexFiVaultProfitStorage.profitToken (src/DexFiVaultProfitStorage.sol#25) can be used in cross function reentrancies: - DexFiVaultProfitStorage._claim(address,uint256) (src/DexFiVaultProfitStorage.sol#204-209) - DexFiVaultProfitStorage._updateProfitToken(address) (src/DexFiVaultProfitStorage.sol#215-220) - DexFiVaultProfitStorage.profitToken (src/DexFiVaultProfitStorage.sol#25) - _updateProfitToken(profitToken_) (src/DexFiVaultProfitStorage.sol#195) - profitTokenConnector = factory.profitTokenConnector(profitToken_) (src/DexFiVaultProfitStorage.sol#218)DexFiVaultProfitStorage.profitTokenConnector (src/DexFiVaultProfitStorage.sol#26) can be used in cross function reentrancies: - DexFiVaultProfitStorage._claim(address,uint256) (src/DexFiVaultProfitStorage.sol#204-209) - DexFiVaultProfitStorage._updateProfitToken(address) (src/DexFiVaultProfitStorage.sol#215-220) - DexFiVaultProfitStorage.allocate() (src/DexFiVaultProfitStorage.sol#123-159) - DexFiVaultProfitStorage.profitTokenConnector (src/DexFiVaultProfitStorage.sol#26)	High

Finding	Impact
DexFiVault.deposit(uint256,address,uint256) (src/DexFiVault.sol#129-181) performs a multiplication on the result of a division: - preDepositTotalFarmsNativeInvestments += (currentFarmNativeAmount * preDepositFarmAmount) / (postDepositFarmAmount - preDepositFarmAmount) (src/DexFiVault.sol#171-173) - currentFarmNativeAmount = (amount * farm_.percent) / divider (src/DexFiVault.sol#162-164)	Medium
DexFiVault.withdrawConvertless(uint256) (src/DexFiVault.sol#342-375) performs a multiplication on the result of a division: - feeSyntheticAmount = (amount * factory.feeConfig().withdrawFee) / divider (src/DexFiVault.sol#350) - userNativeAmount = (native.balanceOf(address(this)) * amount) / totalSupply() (src/DexFiVault.sol#351) - nativeFeeAmount = (userNativeAmount * feeSyntheticAmount) / amount (src/DexFiVault.sol#355)	Medium
DexFiVault.withdraw(uint256,address,uint256) (src/DexFiVault.sol#290-315) performs a multiplication on the result of a division: - nativeAmount = (native.balanceOf(address(this)) * amount) / totalSupply() (src/DexFiVault.sol#301) - withdrawFeeAmount = (nativeAmount * factory.feeConfig().withdrawFee) / divider (src/DexFiVault.sol#304)	Medium

Finding	Impact
Reentrancy in DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618): External calls: - reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) State variables written after the call(s): - harvesterDebt -= paidDebtAmount (src/DexFiVault.sol#608) DexFiVault.harvesterDebt (src/DexFiVault.sol#25) can be used in cross function reentrancies: - DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618) - DexFiVault.harvesterDebt (src/DexFiVault.sol#25) - DexFiVault.increaseHarvesterDebt(uint256) (src/DexFiVault.sol#82-88)	Medium

Finding	Impact
Reentrancy in DexFiVault.emergencyWithdraw() (src/DexFiVault.sol#321-335): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#322) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - connector.emergencyWithdraw(outputAmounts[i],msg.sender) (src/DexFiVault.sol#331) - _burn(msg.sender,syntheticAmount) (src/DexFiVault.sol#333) - IDexFiVaultProfitStorage(profitStorage).updateUsersSharesBalance(updateInfo) (src/DexFiVault.sol#476) State variables written after the call(s): - _burn(msg.sender,syntheticAmount) (src/DexFiVault.sol#333) - _balances[account] = accountBalance - amount (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#290) ERC20Upgradeable._balances (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#40) can be used in cross function reentrancies: - ERC20Upgradeable._transfer(address,address,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#227-245)- ERC20Upgradeable.balanceOf(address) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#106-108)- _burn(msg.sender,syntheticAmount) (src/DexFiVault.sol#333) - _totalSupply -= amount (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#292)ERC20Upgradeable._totalSupply (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#44) can be used in cross function reentrancies: - ERC20Upgradeable.totalSupply() (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#99-101)	Medium

Finding	Impact
Reentrancy in DexFiVault._actualizeFarmsStatus() (src/DexFiVault.sol#414-427): External calls: - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) State variables written after the call(s): - farm_.data = factory.defaultFarmsInitializeData(farm_.beacon) (src/DexFiVault.sol#421) DexFiVault._farms (src/DexFiVault.sol#30) can be used in cross function reentrancies: - DexFiVault._actualizeFarmsStatus() (src/DexFiVault.sol#414-427) - DexFiVault._depositAll() (src/DexFiVault.sol#524-541) - DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618) - DexFiVault._updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#485-507) - DexFiVault.convertlessDepositRatio() (src/DexFiVault.sol#61-67) - DexFiVault.farms(uint256) (src/DexFiVault.sol#45-47) - DexFiVault.farmsCount() (src/DexFiVault.sol#53-55) - farmConnectorLatestReinitializationTimestamp[farm_.beacon] = block.timestamp (src/DexFiVault.sol#424)DexFiVault.farmConnectorLatestReinitializationTimestamp (src/DexFiVault.sol#27) can be used in cross function reentrancies: - DexFiVault._actualizeFarmsStatus() (src/DexFiVault.sol#414-427) - DexFiVault._updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#485-507) - DexFiVault.farmConnectorLatestReinitializationTimestamp (src/DexFiVault.sol#27)	Medium

Finding	Impact
Reentrancy in DexFiVault._updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#485-507): External calls: - connector = IDexFiFarm(address(new BeaconProxy(farm_.beacon,))) (src/DexFiVault.sol#493) - ! connector.initialize(farm_.data) (src/DexFiVault.sol#494) - ! farmConnector_.reinitialize(farm_.data) (src/DexFiVault.sol#498) State variables written after the call(s): - _farms.push(farm_) (src/DexFiVault.sol#503) DexFiVault._farms (src/DexFiVault.sol#30) can be used in cross function reentrancies: - DexFiVault._actualizeFarmsStatus() (src/DexFiVault.sol#414-427) - DexFiVault._depositAll() (src/DexFiVault.sol#524-541) - DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618) - DexFiVault._updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#485-507) - DexFiVault.convertlessDepositRatio() (src/DexFiVault.sol#61-67) - DexFiVault.farms(uint256) (src/DexFiVault.sol#45-47) - DexFiVault.farmsCount() (src/DexFiVault.sol#53-55) - farmConnector[farm_.beacon] = connector (src/DexFiVault.sol#495) DexFiVault.farmConnector (src/DexFiVault.sol#26) can be used in cross function reentrancies: - DexFiVault._actualizeFarmsStatus() (src/DexFiVault.sol#414-427) - DexFiVault._depositAll() (src/DexFiVault.sol#524-541) - DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618) - DexFiVault._updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#485-507) - DexFiVault.convertlessDepositRatio() (src/DexFiVault.sol#61-67) - DexFiVault.farmConnector (src/DexFiVault.sol#26)	Medium
DexFiVault.withdrawConvertless(uint256).feeAmounts (src/DexFiVault.sol#349) is a local variable never initialized	Medium
DexFiVault._depositAll() (src/DexFiVault.sol#524-541) ignores return value by connector.deposit(currentFarmNativeAmount) (src/DexFiVault.sol#537)	Medium
DexFiVault.depositConvertless(uint256[],uint256) (src/DexFiVault.sol#189-235) ignores return value by Address.functionCall(connector_scope_1.stakingToken(),connector_scope_1.stakingTokenApproveData(address(connector_scope_1),stakingAmounts[i_scope_0])) (src/DexFiVault.sol#219-222)	Medium

Finding	Impact
DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618) ignores return value by profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612)	Medium
DexFiVault.depositConvertless(uint256[],uint256) (src/DexFiVault.sol#189-235) ignores return value by Address.functionCall(connector_scope_1.stakingToken(),connector_scope_1.stakingTokenTransferFromData(msg.sender,address(this),stakingAmounts[i_scope_0])) (src/DexFiVault.sol#215-218)	Medium
DexFiVault._afterTokenTransfer(address,address,uint256) (src/DexFiVault.sol#435-479) ignores return value by IDexFiVaultProfitStorage(profitStorage).updateUsersSharesBalance(updateInfo) (src/DexFiVault.sol#476)	Medium
DexFiVault.initialize(string,string,address,uint256,address,IDexFiVault.Farm[]) (src/DexFiVault.sol#100-120) ignores return value by IDexFiVaultProfitStorage(profitStorage).initialize(factory,profitToken_) (src/DexFiVault.sol#116)	Medium
DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618) ignores return value by profit_.allocate() (src/DexFiVault.sol#614)	Medium
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#150-156) ignores return value by Address.functionDelegateCall(IBeacon(newBeacon).implementation(),data) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#154)	Medium
ERC1967Upgrade._upgradeToAndCall(address,bytes,bool) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#59-64) ignores return value by Address.functionDelegateCall(newImplementation,data) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#62)	Medium
DexFiVault.deposit(uint256,address,uint256) (src/DexFiVault.sol#129-181) ignores return value by connector.deposit(currentFarmNativeAmount) (src/DexFiVault.sol#168)	Medium

AUTOMATED TESTING

Finding	Impact
Reentrancy in DexFiVaultProfitStorage.allocate() (src/DexFiVaultProfitStorage.sol#123-159): External calls: - native.forceApprove(address(profitTokenConnector),fund) (src/DexFiVaultProfitStorage.sol#124) - (amount,fundSubtrahend) = profitTokenConnector.swapNativeToProfit(fund) (src/DexFiVaultProfitStorage.sol#125) State variables written after the call(s): - fund -= fundSubtrahend (src/DexFiVaultProfitStorage.sol#126)DexFiVaultProfitStorage.fund (src/DexFiVaultProfitStorage.sol#24) can be used in cross function reentrancies: - DexFiVaultProfitStorage.addProfitFunds(uint256) (src/DexFiVaultProfitStorage.sol#166-172) - DexFiVaultProfitStorage.allocate() (src/DexFiVaultProfitStorage.sol#123-159) - DexFiVaultProfitStorage.fund (src/DexFiVaultProfitStorage.sol#24)	Medium
DexFiVaultProfitStorage.allocate() (src/DexFiVaultProfitStorage.sol#123-159) contains a tautology or contradiction: - i_scope_0 >= 0 (src/DexFiVaultProfitStorage.sol#144)	Medium
DexFiVaultProfitStorage.allocate() (src/DexFiVaultProfitStorage.sol#123-159) ignores return value by _users.remove(user__scope_1) (src/DexFiVaultProfitStorage.sol#152)	Medium
DexFiVaultProfitStorage.updateUsersSharesBalance(IDexFiVaultProfitStorage.UserSyntheticAmount[]) (src/DexFiVaultProfitStorage.sol#95-117) ignores return value by _users.add(user_) (src/DexFiVaultProfitStorage.sol#102)	Medium
DexFiVaultZapper.removeTokensWhitelist(address[]) (src/DexFiVaultZapper.sol#108-116) ignores return value by _tokensWhitelist.remove(underlying) (src/DexFiVaultZapper.sol#111)	Medium
DexFiVaultZapper.updateTokensWhitelist(IDexFiZapperToken[]) (src/DexFiVaultZapper.sol#88-101) ignores return value by _tokensWhitelist.add(underlying) (src/DexFiVaultZapper.sol#96)	Medium

Finding	Impact
Reentrancy in DexFiVaultFactory._updateProfitConfig(IDexFiVaultFactory.ProfitConfig) (src/DexFiVaultFactory.sol#450-465): External calls: - UpgradeableBeacon(previousProfitStorageImplementation).upgradeTo(config.profitStorageImplementation) (src/DexFiVaultFactory.sol#460) State variables written after the call(s): - _profitConfig = config (src/DexFiVaultFactory.sol#463)DexFiVaultFactory._profitConfig (src/DexFiVaultFactory.sol#36) can be used in cross function reentrancies: - DexFiVaultFactory._updateProfitConfig(IDexFiVaultFactory.ProfitConfig) (src/DexFiVaultFactory.sol#450-465) - DexFiVaultFactory.profitConfig() (src/DexFiVaultFactory.sol#56-58)	Medium
Reentrancy in DexFiVaultFactory.addProfitTokensWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#317-336): External calls: - profitTokenConnector[beacon] = IDexFiProfit(address(new BeaconProxy(beacon,))) (src/DexFiVaultFactory.sol#325) - ! profitTokenConnector[beacon].initialize(profitToken_.defaultInitializeData) (src/DexFiVaultFactory.sol#331) State variables written after the call(s): - profitTokenConnector[beacon] = IDexFiProfit(address(new BeaconProxy(beacon,))) (src/DexFiVaultFactory.sol#325)DexFiVaultFactory.profitTokenConnector (src/DexFiVaultFactory.sol#32) can be used in cross function reentrancies: - DexFiVaultFactory.addProfitTokensWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#317-336) - DexFiVaultFactory.profitTokenConnector (src/DexFiVaultFactory.sol#32) - DexFiVaultFactory.updateProfitTokensWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#343-358)	Medium

Finding	Impact
Reentrancy in DexFiVaultFactory.addFarmsWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#250-271): External calls: - farmCalculationConnector[beacon] = IDexFiFarm(address(new BeaconProxy(beacon,))) (src/DexFiVaultFactory.sol#258) - ! farmCalculationConnector[beacon].initialize(farm_.defaultInitializeData) (src/DexFiVaultFactory.sol#264) State variables written after the call(s): - farmCalculationConnector[beacon] = IDexFiFarm(address(new BeaconProxy(beacon,))) (src/DexFiVaultFactory.sol#258)DexFiVaultFactory.farmCalculationConnector (src/DexFiVaultFactory.sol#33) can be used in cross function reentrancies: - DexFiVaultFactory.addFarmsWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#250-271) - DexFiVaultFactory.farmCalculationConnector (src/DexFiVaultFactory.sol#33) - DexFiVaultFactory.updateFarmsWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#278-294)	Medium
Reentrancy in DexFiVaultFactory._updateVaultConfig(IDexFiVaultFactory.VaultConfig) (src/DexFiVaultFactory.sol#486-514): External calls: - UpgradeableBeacon(previousVaultImplementation).upgradeTo(config.vaultImplementation) (src/DexFiVaultFactory.sol#509) State variables written after the call(s): - _vaultConfig = config (src/DexFiVaultFactory.sol#512)DexFiVaultFactory._vaultConfig (src/DexFiVaultFactory.sol#37) can be used in cross function reentrancies: - DexFiVaultFactory._updateVaultConfig(IDexFiVaultFactory.VaultConfig) (src/DexFiVaultFactory.sol#486-514) - DexFiVaultFactory.createVault(string,string,uint256,address,IDexFiVault.Farm[]) (src/DexFiVaultFactory.sol#211-225) - DexFiVaultFactory.vaultConfig() (src/DexFiVaultFactory.sol#64-66)	Medium
DexFiVaultFactory.toggleVaultAutoHarvest(address) (src/DexFiVaultFactory.sol#416-427) ignores return value by _harvesterBlacklist.remove(vault) (src/DexFiVaultFactory.sol#420)	Medium

Finding	Impact
DexFiVaultFactory.addFarmsWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#250-271) ignores return value by _farmsWhitelist.add(beacon) (src/DexFiVaultFactory.sol#266)	Medium
DexFiVaultFactory.addProfitTokensWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#317-336) ignores return value by _profitTokensWhitelist.add(beacon) (src/DexFiVaultFactory.sol#333)	Medium
DexFiVaultFactory.toggleVaultAutoHarvest(address) (src/DexFiVaultFactory.sol#416-427) ignores return value by _harvesterBlacklist.add(vault) (src/DexFiVaultFactory.sol#423)	Medium
DexFiVaultFactory.removeProfitTokensWhitelist(address[]) (src/DexFiVaultFactory.sol#365-369) ignores return value by _profitTokensWhitelist.remove(beacons[i]) (src/DexFiVaultFactory.sol#366)	Medium
DexFiVaultFactory.createVault(string,string,uint256,address,IDexFiVault.Farm[]) (src/DexFiVaultFactory.sol#211-225) ignores return value by _vaults.add(vault) (src/DexFiVaultFactory.sol#223)	Medium
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#150-156) ignores return value by Address.functionDelegateCall(IBeacon(newBeacon).implementation(),data) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#154)	Medium
ERC1967Upgrade._upgradeToAndCall(address,bytes,bool) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#59-64) ignores return value by Address.functionDelegateCall(newImplementation,data) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#62)	Medium
DexFiVaultFactory.removeFarmsWhitelist(address[]) (src/DexFiVaultFactory.sol#301-310) ignores return value by _farmsWhitelist.remove(beacon) (src/DexFiVaultFactory.sol#304)	Medium
DexFiVaultProfitClaimer.claimByProfitStorages(IDexFiVaultProfitStorage[]) (src/DexFiVaultProfitClaimer.sol#17-24) ignores return value by profitStorage.claim(userAvailableToClaim,msg.sender) (src/DexFiVaultProfitClaimer.sol#21)	Medium
DexFiVaultProfitClaimer.claimByVaults(IDexFiVault[]) (src/DexFiVaultProfitClaimer.sol#31-39) ignores return value by profitStorage.claim(userAvailableToClaim,msg.sender) (src/DexFiVaultProfitClaimer.sol#36)	Medium

Finding	Impact
DexFiVaultHarvester.harvest(address[]) (src/DexFiVaultHarvester.sol#23-35) ignores return value by vault.harvest() (src/DexFiVaultHarvester.sol#31)	Medium
DexFiVaultHarvester.harvest(address[]) (src/DexFiVaultHarvester.sol#23-35) ignores return value by vault.increaseHarvesterDebt(gasUsed[i] * tx.gasprice) (src/DexFiVaultHarvester.sol#33)	Medium
DexFiVault.depositConvertless(uint256[],uint256) (src/DexFiVault.sol#189-235) has external calls inside a loop: ratio = (divider * connector.stakingTokenLiquidity(stakingAmounts[i])) / connector.liquidity(address(connector)) (src/DexFiVault.sol#204-205)	Low
DexFiVault.depositConvertless(uint256[],uint256) (src/DexFiVault.sol#189-235) has external calls inside a loop: Address.functionCall(connector_scope_1.stakingToken(),connector_scope_1.stakingTokenTransferFromData(msg.sender,address(this),stakingAmounts[i_scope_0])) (src/DexFiVault.sol#215-218)	Low
DexFiVault.deposit(uint256,address,uint256) (src/DexFiVault.sol#129-181) has external calls inside a loop: postDepositFarmAmount = connector.liquidity(address(connector)) (src/DexFiVault.sol#170)	Low
DexFiVault.emergencyWithdraw() (src/DexFiVault.sol#321-335) has external calls inside a loop: connector.emergencyWithdraw(outputAmounts[i],msg.sender) (src/DexFiVault.sol#331)	Low
DexFiVault.deposit(uint256,address,uint256) (src/DexFiVault.sol#129-181) has external calls inside a loop: connector.deposit(currentFarmNativeAmount) (src/DexFiVault.sol#168)	Low
DexFiVault.convertlessDepositRatio() (src/DexFiVault.sol#61-67) has external calls inside a loop: output[i] = connector.liquidity(address(connector)) (src/DexFiVault.sol#65)	Low
DexFiVault.depositConvertless(uint256[],uint256) (src/DexFiVault.sol#189-235) has external calls inside a loop: connector_scope_1.depositConvertless(stakingAmounts[i_scope_0],depositLiquidityAmounts[i_scope_0],feeLiquidityAmounts[i_scope_0],factory.integrationConfig().treasury,msg.sender) (src/DexFiVault.sol#223-229)	Low

Finding	Impact
DexFiVault.deposit(uint256,address,uint256) (src/DexFiVault.sol#129-181) has external calls inside a loop: preDepositFarmAmount = connector.liquidity(address(connector)) (src/DexFiVault.sol#161)	Low
DexFiVault.depositConvertless(uint256[],uint256) (src/DexFiVault.sol#189-235) has external calls inside a loop: depositLiquidityAmounts[i_scope_0] = (connector_scope_1.liquidity(address(connector_scope_1)) * minRatio) / divider (src/DexFiVault.sol#212)	Low
DexFiVault.depositConvertless(uint256[],uint256) (src/DexFiVault.sol#189-235) has external calls inside a loop: Address.functionCall(connector_scope_1.stakingToken(),connector_scope_1.stakingTokenApproveData(address(connector_scope_1),stakingAmounts[i_scope_0])) (src/DexFiVault.sol#219-222)	Low
DexFiVault.emergencyWithdraw() (src/DexFiVault.sol#321-335) has external calls inside a loop: outputAmounts[i] = (connector.liquidity(address(connector)) * syntheticAmount) / syntheticTotalSupply (src/DexFiVault.sol#330)	Low

Finding	Impact
Reentrancy in DexFiVault.withdraw(uint256,address,uint256) (src/DexFiVault.sol#290-315): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#296) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#297) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#297) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - _harvestAndSwapToNative() (src/DexFiVault.sol#297) - lastHarvestTimestamp = block.timestamp (src/DexFiVault.sol#615)	Low

Finding	Impact
Reentrancy in DexFiVault.initialize(string,string,address,uint256,address,IDexFiVault.Farm[]) (src/DexFiVault.sol#100-120): External calls: - profitStorage = address(new BeaconProxy(factory.profitConfig().profitStorageImplementation,)) (src/DexFiVault.sol#115) - IDexFiVaultProfitStorage(profitStorage).initialize(factory,profitToken_) (src/DexFiVault.sol#116) State variables written after the call(s): - _updateProfit(profit_) (src/DexFiVault.sol#117) - profit = profit_ (src/DexFiVault.sol#517)	Low

Finding	Impact
Reentrancy in DexFiVault.withdrawConvertless(uint256) (src/DexFiVault.sol#342-375): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#346) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#347) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#347) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - _harvestAndSwapToNative() (src/DexFiVault.sol#347) - lastHarvestTimestamp = block.timestamp (src/DexFiVault.sol#615)	Low

Finding	Impact
Reentrancy in DexFiVault.deposit(uint256,address,uint256) (src/DexFiVault.sol#129-181): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#135) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#136) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#136) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - _harvestAndSwapToNative() (src/DexFiVault.sol#136) - lastHarvestTimestamp = block.timestamp (src/DexFiVault.sol#615)	Low

Finding	Impact
Reentrancy in DexFiVault.initialize(string,string,address,uint256,address,IDexFiVault.Farm[]) (src/DexFiVault.sol#100-120): External calls: - profitStorage = address(new BeaconProxy(factory.profitConfig().profitStorageImplementation,)) (src/DexFiVault.sol#115) - IDexFiVaultProfitStorage(profitStorage).initialize(factory,profitToken_) (src/DexFiVault.sol#116) - _updateFarms(farms_) (src/DexFiVault.sol#118) - connector = IDexFiFarm(address(new BeaconProxy(farm_.beacon,))) (src/DexFiVault.sol#493) - ! connector.initialize(farm_.data) (src/DexFiVault.sol#494) - ! farmConnector_.reinitialize(farm_.data) (src/DexFiVault.sol#498) State variables written after the call(s): - _updateFarms(farms_) (src/DexFiVault.sol#118) - farmConnectorLatestReinitializationTimestamp[farm_.beacon] = block.timestamp (src/DexFiVault.sol#501)	Low

Finding	Impact
Reentrancy in DexFiVault.publish() (src/DexFiVault.sol#381-390): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#382) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - lastHarvestTimestamp = block.timestamp (src/DexFiVault.sol#615)	Low

Finding	Impact
Reentrancy in DexFiVault.depositConvertless(uint256[],uint256) (src/DexFiVault.sol#189-235): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#194) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#195) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#195) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - _harvestAndSwapToNative() (src/DexFiVault.sol#195) - lastHarvestTimestamp = block.timestamp (src/DexFiVault.sol#615)	Low
Reentrancy in DexFiVault._updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#485-507): External calls: - connector = IDexFiFarm(address(new BeaconProxy(farm_.beacon,))) (src/DexFiVault.sol#493) - ! connector.initialize(farm_.data) (src/DexFiVault.sol#494) - ! farmConnector_.reinitialize(farm_.data) (src/DexFiVault.sol#498) State variables written after the call(s): - farmConnectorLatestReinitializationTimestamp[farm_.beacon] = block.timestamp (src/DexFiVault.sol#501)	Low

Finding	Impact
Reentrancy in DexFiVault.updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#273-281): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#274) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#275) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#275) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - _harvestAndSwapToNative() (src/DexFiVault.sol#275) - lastHarvestTimestamp = block.timestamp (src/DexFiVault.sol#615)	Low

Finding	Impact
Reentrancy in DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618): External calls: - reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) State variables written after the call(s): - lastHarvestTimestamp = block.timestamp (src/DexFiVault.sol#615)	Low

Finding	Impact
Reentrancy in DexFiVault.publish() (src/DexFiVault.sol#381-390): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#382) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) - _depositAll() (src/DexFiVault.sol#384) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- native.forceApprove(address(connector),currentFarmNativeAmount) (src/DexFiVault.sol#536) - connector.deposit(currentFarmNativeAmount) (src/DexFiVault.sol#537) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- _depositAll() (src/DexFiVault.sol#384) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - _publishData.published = true (src/DexFiVault.sol#385) - _publishData.publisher = msg.sender (src/DexFiVault.sol#386)	Low

Finding	Impact
Reentrancy in DexFiVault.harvest() (src/DexFiVault.sol#245-266): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#257) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - (reinvestAmount,harvestFeeAmount,profitAmount,paidDebtAmount,remainingDebtAmount) = _harvestAndSwapToNative() (src/DexFiVault.sol#258-264) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) External calls sending eth: - (reinvestAmount,harvestFeeAmount,profitAmount,paidDebtAmount,remainingDebtAmount) = _harvestAndSwapToNative() (src/DexFiVault.sol#258-264) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)State variables written after the call(s): - (reinvestAmount,harvestFeeAmount,profitAmount,paidDebtAmount,remainingDebtAmount) = _harvestAndSwapToNative() (src/DexFiVault.sol#258-264) - lastHarvestTimestamp = block.timestamp (src/DexFiVault.sol#615)	Low

Finding	Impact
Reentrancy in DexFiVault.initialize(string,string,address,uint256,address,IDexFiVault.Farm[]) (src/DexFiVault.sol#100-120): External calls: - profitStorage = address(new BeaconProxy(factory.profitConfig().profitStorageImplementation,)) (src/DexFiVault.sol#115) - IDexFiVaultProfitStorage(profitStorage).initialize(factory,profitToken_) (src/DexFiVault.sol#116) - _updateFarms(farms_) (src/DexFiVault.sol#118) - connector = IDexFiFarm(address(new BeaconProxy(farm_.beacon,))) (src/DexFiVault.sol#493) - ! connector.initialize(farm_.data) (src/DexFiVault.sol#494) - ! farmConnector_.reinitialize(farm_.data) (src/DexFiVault.sol#498) Event emitted after the call(s): - FarmsUpdated(farms_) (src/DexFiVault.sol#506) - _updateFarms(farms_) (src/DexFiVault.sol#118)	Low
Reentrancy in DexFiVault.initialize(string,string,address,uint256,address,IDexFiVault.Farm[]) (src/DexFiVault.sol#100-120): External calls: - profitStorage = address(new BeaconProxy(factory.profitConfig().profitStorageImplementation,)) (src/DexFiVault.sol#115) - IDexFiVaultProfitStorage(profitStorage).initialize(factory,profitToken_) (src/DexFiVault.sol#116) Event emitted after the call(s): - ProfitUpdated(profit_) (src/DexFiVault.sol#518) - _updateProfit(profit_) (src/DexFiVault.sol#117)	Low
Reentrancy in DexFiVault._updateFarms(IDexFiVault.Farm[]) (src/DexFiVault.sol#485-507): External calls: - connector = IDexFiFarm(address(new BeaconProxy(farm_.beacon,))) (src/DexFiVault.sol#493) - ! connector.initialize(farm_.data) (src/DexFiVault.sol#494) - ! farmConnector_.reinitialize(farm_.data) (src/DexFiVault.sol#498) Event emitted after the call(s): - FarmsUpdated(farms_) (src/DexFiVault.sol#506)	Low

Finding	Impact
Reentrancy in DexFiVault.publish() (src/DexFiVault.sol#381-390): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#382) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) - _depositAll() (src/DexFiVault.sol#384) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- native.forceApprove(address(connector),currentFarmNativeAmount) (src/DexFiVault.sol#536) - connector.deposit(currentFarmNativeAmount) (src/DexFiVault.sol#537) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- _depositAll() (src/DexFiVault.sol#384) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)Event emitted after the call(s): - OwnershipTransferred(oldOwner,newOwner) (lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#86)-	Low

Finding	Impact
Reentrancy in DexFiVault.publish() (src/DexFiVault.sol#381-390): External calls: - _actualizeFarmsStatus() (src/DexFiVault.sol#382) - ! farmConnector[farm_.beacon].reinitialize(farm_.data) (src/DexFiVault.sol#422) - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#122)- (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol#139)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)- reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) External calls sending eth: - _harvestAndSwapToNative() (src/DexFiVault.sol#383) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts-upgradeable/contracts/utils/AddressUpgradeable.sol#135)Event emitted after the call(s): - Harvested(msg.sender,reinvestAmount,harvestFeeAmount,profitAmount,paidDebtAmount,remainingDebtAmount) (src/DexFiVault.sol#617) - _harvestAndSwapToNative() (src/DexFiVault.sol#383)	Low

Finding	Impact
Reentrancy in DexFiVault._harvestAndSwapToNative() (src/DexFiVault.sol#586-618): External calls: - reinvestAmount += farmConnector[_farms[i].beacon].harvest() (src/DexFiVault.sol#597) - native.safeTransfer(integrationConfig.treasury,harvestFeeAmount) (src/DexFiVault.sol#605) - native.safeTransfer(integrationConfig.keeper,paidDebtAmount) (src/DexFiVault.sol#607) - native.forceApprove(profitStorage,profitAmount) (src/DexFiVault.sol#611) - profit_.addProfitFunds(profitAmount) (src/DexFiVault.sol#612) - profit_.allocate() (src/DexFiVault.sol#614) Event emitted after the call(s): - Harvested(msg.sender,reinvestAmount,harvestFeeAmount,profitAmount ,paidDebtAmount,remainingDebtAmount) (src/DexFiVault.sol#617)	Low
DexFiVault._actualizeFarmsStatus() (src/DexFiVault.sol#414-427) uses timestamp for comparisons Dangerous comparisons: - farmConnectorLatestReinitializationTimestamp[farm_.beacon] < factory.defaultFarmsInitializeDataTimestamp(farm_.beacon) (src/DexFiVault.sol#418-419)	Low
Reentrancy in DexFiVaultProfitStorage.addProfitFunds(uint256) (src/DexFiVaultProfitStorage.sol#166-172): External calls: - native.safeTransferFrom(msg.sender,address(this),amount) (src/DexFiVaultProfitStorage.sol#168) State variables written after the call(s): - fund += amount (src/DexFiVaultProfitStorage.sol#169)	Low

Finding	Impact
Reentrancy in DexFiVaultProfitStorage.allocate() (src/DexFiVaultProfitStorage.sol#123-159): External calls: - native.forceApprove(address(profitTokenConnector),fund) (src/DexFiVaultProfitStorage.sol#124) - (amount,fundSubtrahend) = profitTokenConnector.swapNativeToProfit(fund) (src/DexFiVaultProfitStorage.sol#125) State variables written after the call(s): - _usersSharesInfo[user_].accShares += addShares (src/DexFiVaultProfitStorage.sol#139) - _usersSharesInfo[user__scope_1].accShares = 0 (src/DexFiVaultProfitStorage.sol#148) - _usersSharesInfo[user__scope_1].lastRewardBlock = block.number (src/DexFiVaultProfitStorage.sol#150) - availableToClaim[publisher] += publisherPart (src/DexFiVaultProfitStorage.sol#130) - availableToClaim[user__scope_1] += (amount * _usersSharesInfo[user__scope_1].accShares) / totalAccShares (src/DexFiVaultProfitStorage.sol#147)	Low
Reentrancy in DexFiVaultProfitStorage.addProfitFunds(uint256) (src/DexFiVaultProfitStorage.sol#166-172): External calls: - native.safeTransferFrom(msg.sender,address(this),amount) (src/DexFiVaultProfitStorage.sol#168) Event emitted after the call(s): - ProfitFundsAdded(amount) (src/DexFiVaultProfitStorage.sol#170)	Low
Reentrancy in DexFiVaultProfitStorage.allocate() (src/DexFiVaultProfitStorage.sol#123-159): External calls: - native.forceApprove(address(profitTokenConnector),fund) (src/DexFiVaultProfitStorage.sol#124) - (amount,fundSubtrahend) = profitTokenConnector.swapNativeToProfit(fund) (src/DexFiVaultProfitStorage.sol#125) Event emitted after the call(s): - UserRemoved(user__scope_1) (src/DexFiVaultProfitStorage.sol#153)	Low

Finding	Impact
Reentrancy in DexFiVaultProfitStorage._claim(address,uint256) (src/DexFiVaultProfitStorage.sol#204-209): External calls: - IERC20(profitTokenConnector.underlying()).safeTransfer(user,amount) (src/DexFiVaultProfitStorage.sol#207) Event emitted after the call(s): - Claimed(user,profitToken,amount) (src/DexFiVaultProfitStorage.sol#208)	Low
Reentrancy in DexFiVaultProfitStorage.updateProfitToken(address) (src/DexFiVaultProfitStorage.sol#191-197): External calls: - _claim(vaultOwner,availableToClaim[vaultOwner]) (src/DexFiVaultProfitStorage.sol#194) - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#122)- (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#135)- IERC20(profitTokenConnector.underlying()).safeTransfer(user,amount) (src/DexFiVaultProfitStorage.sol#207) External calls sending eth: - _claim(vaultOwner,availableToClaim[vaultOwner]) (src/DexFiVaultProfitStorage.sol#194) - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#135)Event emitted after the call(s): - ProfitTokenUpdated(profitToken_) (src/DexFiVaultProfitStorage.sol#219) - _updateProfitToken(profitToken_) (src/DexFiVaultProfitStorage.sol#195)	Low
DexFiVaultZapper.deposit(IDexFiVault,DexFiVaultZapper.DepositSource[],uint256) (src/DexFiVaultZapper.sol#125-152) has external calls inside a loop: nativeAmount += sourceLogic.swapUnderlyingToNative(source_.amount,address(this)) (src/DexFiVaultZapper.sol#145)	Low
DexFiVaultZapper.updateTokensWhitelist(IDexFiZapperToken[]) (src/DexFiVaultZapper.sol#88-101) has external calls inside a loop: underlying = tokenLogic_.underlying() (src/DexFiVaultZapper.sol#95)	Low
DexFiVaultZapper.updateTokensWhitelist(IDexFiZapperToken[]) (src/DexFiVaultZapper.sol#88-101) has external calls inside a loop: revert UpdateTokensWhitelistTokenLogicNativeDiffers(address,address)(tokenLogic_.native(),native) (src/DexFiVaultZapper.sol#94)	Low

Finding	Impact
DexFiVaultZapper.updateTokensWhitelist(IDexFiZapperToken[]) (src/DexFiVaultZapper.sol#88-101) has external calls inside a loop: tokenLogic_.native() != native (src/DexFiVaultZapper.sol#93)	Low
DexFiVaultFactory.updateFarmsWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#278-294) has external calls inside a loop: beacon.implementation() != info.source (src/DexFiVaultFactory.sol#283)	Low
DexFiVaultFactory.updateProfitTokensWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#343-358) has external calls inside a loop: ! profitTokenConnector[info.beacon].reinitialize(info.defaultInitializeData) (src/DexFiVaultFactory.sol#353)	Low
DexFiVaultFactory.updateFarmsWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#278-294) has external calls inside a loop: ! farmCalculationConnector[info.beacon].reinitialize(info.defaultInitializeData) (src/DexFiVaultFactory.sol#287)	Low
DexFiVaultFactory.updateProfitTokensWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#343-358) has external calls inside a loop: beacon.upgradeTo(info.source) (src/DexFiVaultFactory.sol#351)	Low
DexFiVaultFactory.addProfitTokensWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#317-336) has external calls inside a loop: ! profitTokenConnector[beacon].initialize(profitToken_.defaultInitializeData) (src/DexFiVaultFactory.sol#331)	Low
DexFiVaultFactory.addFarmsWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#250-271) has external calls inside a loop: ! farmCalculationConnector[beacon].initialize(farm_.defaultInitializeData) (src/DexFiVaultFactory.sol#264)	Low
DexFiVaultFactory.updateFarmsWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#278-294) has external calls inside a loop: beacon.upgradeTo(info.source) (src/DexFiVaultFactory.sol#285)	Low
DexFiVaultFactory.updateProfitTokensWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#343-358) has external calls inside a loop: beacon.implementation() != info.source (src/DexFiVaultFactory.sol#349)	Low

Finding	Impact
Reentrancy in DexFiVaultFactory.addFarmsWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#250-271): External calls: - farmCalculationConnector[beacon] = IDexFiFarm(address(new BeaconProxy(beacon,))) (src/DexFiVaultFactory.sol#258) - ! farmCalculationConnector[beacon].initialize(farm_.defaultInitializeData) (src/DexFiVaultFactory.sol#264) State variables written after the call(s): - defaultFarmsInitializeData[beacon] = farm_.defaultInitializeData (src/DexFiVaultFactory.sol#267) - defaultFarmsInitializeDataTimestamp[beacon] = block.timestamp (src/DexFiVaultFactory.sol#268)	Low
Reentrancy in DexFiVaultFactory.updateFarmsWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#278-294): External calls: - beacon.upgradeTo(info.source) (src/DexFiVaultFactory.sol#285) - ! farmCalculationConnector[info.beacon].reinitialize(info.defaultInitializeData) (src/DexFiVaultFactory.sol#287) State variables written after the call(s): - defaultFarmsInitializeData[info.beacon] = info.defaultInitializeData (src/DexFiVaultFactory.sol#289) - defaultFarmsInitializeDataTimestamp[info.beacon] = block.timestamp (src/DexFiVaultFactory.sol#290)	Low
Reentrancy in DexFiVaultFactory.updateProfitTokensWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#343-358): External calls: - beacon.upgradeTo(info.source) (src/DexFiVaultFactory.sol#351) - ! profitTokenConnector[info.beacon].reinitialize(info.defaultInitializeData) (src/DexFiVaultFactory.sol#353) Event emitted after the call(s): - ProfitTokensWhitelistUpdated(profitTokens_) (src/DexFiVaultFactory.sol#356)	Low

Finding	Impact
Reentrancy in DexFiVaultFactory._updateProfitConfig(IDexFiVaultFactory.ProfitConfig) (src/DexFiVaultFactory.sol#450-465): External calls: - UpgradeableBeacon(previousProfitStorageImplementation).upgradeTo(config.profitStorageImplementation) (src/DexFiVaultFactory.sol#460) Event emitted after the call(s): - ProfitConfigUpdated(config) (src/DexFiVaultFactory.sol#464)	Low
Reentrancy in DexFiVaultFactory.createVault(string,string,uint256,address,IDexFiVault.Farm[]) (src/DexFiVaultFactory.sol#211-225): External calls: - vault = address(new BeaconProxy(_vaultConfig.vaultImplementation,)) (src/DexFiVaultFactory.sol#220) - ! IDexFiVault(vault).initialize(name_,symbol_,msg.sender,profit_,profitToken_,farms_) (src/DexFiVaultFactory.sol#221) Event emitted after the call(s): - VaultCreated(msg.sender,vault,profit_,profitToken_,farms_) (src/DexFiVaultFactory.sol#224)	Low
Reentrancy in DexFiVaultFactory._updateVaultConfig(IDexFiVaultFactory.VaultConfig) (src/DexFiVaultFactory.sol#486-514): External calls: - UpgradeableBeacon(previousVaultImplementation).upgradeTo(config.vaultImplementation) (src/DexFiVaultFactory.sol#509) Event emitted after the call(s): - VaultConfigUpdated(config) (src/DexFiVaultFactory.sol#513)	Low
Reentrancy in DexFiVaultFactory.addProfitTokensWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#317-336): External calls: - profitTokenConnector[beacon] = IDexFiProfit(address(new BeaconProxy(beacon,))) (src/DexFiVaultFactory.sol#325) - ! profitTokenConnector[beacon].initialize(profitToken_.defaultInitializeData) (src/DexFiVaultFactory.sol#331) Event emitted after the call(s): - ProfitTokensWhitelistAdded(output) (src/DexFiVaultFactory.sol#335)	Low

Finding	Impact
Reentrancy in DexFiVaultFactory.addFarmsWhitelist(IDexFiVaultFactory.InitializableConfig[]) (src/DexFiVaultFactory.sol#250-271): External calls: - farmCalculationConnector[beacon] = IDexFiFarm(address(new BeaconProxy(beacon,))) (src/DexFiVaultFactory.sol#258) - ! farmCalculationConnector[beacon].initialize(farm_.defaultInitializeData) (src/DexFiVaultFactory.sol#264) Event emitted after the call(s): - FarmsWhitelistAdded(output) (src/DexFiVaultFactory.sol#270)	Low
Reentrancy in DexFiVaultFactory.updateFarmsWhitelist(IDexFiVaultFactory.Beaconed[]) (src/DexFiVaultFactory.sol#278-294): External calls: - beacon.upgradeTo(info.source) (src/DexFiVaultFactory.sol#285) - ! farmCalculationConnector[info.beacon].reinitialize(info.defaultInitializeData) (src/DexFiVaultFactory.sol#287) Event emitted after the call(s): - FarmsWhitelistUpdated(farms_) (src/DexFiVaultFactory.sol#292)	Low
DexFiVaultProfitClaimer.claimByVaults(IDexFiVault[]) (src/DexFiVaultProfitClaimer.sol#31-39) has external calls inside a loop: profitStorage.claim(userAvailableToClaim,msg.sender) (src/DexFiVaultProfitClaimer.sol#36)	Low
DexFiVaultProfitClaimer.claimByProfitStorages(IDexFiVaultProfitStorage[]) (src/DexFiVaultProfitClaimer.sol#17-24) has external calls inside a loop: profitStorage.claim(userAvailableToClaim,msg.sender) (src/DexFiVaultProfitClaimer.sol#21)	Low
DexFiVaultProfitClaimer.claimByProfitStorages(IDexFiVaultProfitStorage[]) (src/DexFiVaultProfitClaimer.sol#17-24) has external calls inside a loop: userAvailableToClaim = profitStorage.availableToClaim(msg.sender) (src/DexFiVaultProfitClaimer.sol#20)	Low
DexFiVaultProfitClaimer.claimByVaults(IDexFiVault[]) (src/DexFiVaultProfitClaimer.sol#31-39) has external calls inside a loop: userAvailableToClaim = profitStorage.availableToClaim(msg.sender) (src/DexFiVaultProfitClaimer.sol#35)	Low

Finding	Impact
DexFiVaultProfitClaimer.claimByVaults(IDexFiVault[]) (src/DexFiVaultProfitClaimer.sol#31-39) has external calls inside a loop: profitStorage = IDexFiVaultProfitStorage(vault.profitStorage()) (src/DexFiVaultProfitClaimer.sol#34)	Low
DexFiVaultHarvester.harvest(address[]) (src/DexFiVaultHarvester.sol#23-35) has external calls inside a loop: vault.increaseHarvesterDebt(gasUsed[i] * tx.gasprice) (src/DexFiVaultHarvester.sol#33)	Low
DexFiVaultHarvester.harvest(address[]) (src/DexFiVaultHarvester.sol#23-35) has external calls inside a loop: keeper = vault.factory().integrationConfig().keeper (src/DexFiVaultHarvester.sol#29)	Low
DexFiVaultHarvester.harvest(address[]) (src/DexFiVaultHarvester.sol#23-35) has external calls inside a loop: vault.harvest() (src/DexFiVaultHarvester.sol#31)	Low
Reentrancy in DexFiVaultMigrator.migrate(IDexFiVault,IDexFiVault,uint256,uint256) (src/DexFiVaultMigrator.sol#75-100): External calls: - nativeAmount = from.withdraw(amount,msg.sender,0) (src/DexFiVaultMigrator.sol#82) - native.safeTransfer(toFactory.integrationConfig().treasury,nativeFeeAmount) (src/DexFiVaultMigrator.sol#85) - native.forceApprove(address(to),nativeAmount) (src/DexFiVaultMigrator.sol#88) - toSyntheticAmount = to.deposit(nativeAmount,msg.sender,minToSyntheticAmount) (src/DexFiVaultMigrator.sol#89) Event emitted after the call(s): - Migrated(msg.sender,from,to,amount,nativeAmount + nativeFeeAmount,nativeFeeAmount,nativeAmount,toSyntheticAmount) (src/DexFiVaultMigrator.sol#90-99)	Low
End of table for contracts	

Results summary:

The findings obtained as a result of the Slither scan were reviewed. The majority of Slither findings were determined false-positives.

THANK YOU FOR CHOOSING

// HALBORN