Smart Contract Security Audit Report

UFarm

1. Contents

1. Contents 2
2. General Information 3
2.1. Introduction 3
2.2. Scope of Work 3
2.3. Threat Model 3
2.4. Weakness Scoring 4
2.5. Disclaimer 4
3. Summary 5
3.1. Suggestions 5
4. General Recommendations 8
4.1. Security Process Improvement 8
5. Findings 9
5.1. totalCost parameter may be manipulated 9
5.2. Unsafe safeTransfer() 9
5.3. Centralization risks 10
5.4. Incorrect calculations in getCostControlledERC20 11
5.5. Possible DoS via malicious ERC20 asset 11
5.6. Not whitelisted tokens could be in the path in Unoswap Controllers 12
5.7. Function `delegateIncreaseLiquidity()` in not checking position owner 13
5.8. Possible DoS in _processWithdrawal 13
5.9. Incorrect calculations in getAmountInRes() 14
5.10. PriceOracle doesn't check If the Arbitrum sequencer is down in Chainlink feeds 15
5.11. Chainlink's price feed might return stale or incorrect results 16
5.12. Withdraw request lacks deadline check 16
5.13. Unhandled Chainlink revert 17
5.14. The 'Fund Editor' role is too powerful 18
5.15. Use of unchecked block 18
5.16. Signature malleability 19
5.17. Typo in `UniV2RemoveLiquidtyArgs` 20
5.18. depositLockupPeriod is not used 20
5.19. EnumerableSet can be used 21
5.20. Hardcoded values instead of variables 21
5.21. A ternary operation can be simplified 22
5.22. Double-check for zero value 23
5.23. Cast from uint256 to uint8 24
6. Appendix 25
6.1. About us 25

2. General Information

This report contains information about the results of the security audit of the UFarm (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 12/04/2023 to 12/15/2023.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/UFarmDigital/ufarm-evm-contracts. Initial review was done for the commit 3d87f9619918f8845765687b9ab94a840ab3bd67. The re-testing was done for the commit ae677fc012ddab6a9dd5ef88e183ca3287df91b0.

2.3. Threat Model

The assessment presumes the actions of an intruder who might have the capabilities of any role (an external user, token owner, token service owner, or a contract). The risks of centralization were not taken into account at the Customer's request.

The main possible threat actors are:

- Fund User,
- Fund managers,
- Protocol managers,
- Third parties (oracles, other protocols, etc).

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, and the impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises the best effort to perform its contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using limited resources.

3. Summary

As a result of this work, we have discovered a single critical exploitable security issue, which allowed to manipulate the value of the vault to inflate the attacker's share and drain the user funds.

The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement).

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of February 14, 2024.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
totalCost parameter may be manipulated	contracts/pool/UFarmPool.sol	**Critical**	Fixed
Unsafe safeTransfer()	contracts/main/shared/SafeOPS.sol	**High**	Fixed
Centralization risks		**Medium**	Not fixed
Incorrect calculations in getCostControlledERC20	contracts/controllers/UnoswapV2Controller.sol	**Medium**	Fixed
Possible DoS via malicious ERC20 asset	contracts/controllers/UnoswapV3Controller.sol	**Medium**	Fixed
Not whitelisted tokens could be in the path in Unoswap Controllers	contracts/controllers/UnoswapV2Controller.sol	**Medium**	Fixed
Function `delegateIncreaseLiquidity()` in not checking position owner	contracts/main/contracts/controllers/UnoswapV3Controller.sol	**Medium**	Fixed

Issue	Contract	Risk Level	Status
Possible DoS in _processWithdrawal	contracts/main/contracts/pool/UFarmPool.sol	Medium	Fixed
Incorrect calculations in getAmountInRes()	contracts/main/contracts/controllers/UnoswapV2Controller.sol	Medium	Fixed
PriceOracle doesn't check If the Arbitrum sequencer is down in Chainlink feeds	contracts/main/contracts/pool/PriceOracle.sol	Medium	Fixed
Chainlink's price feed might return stale or incorrect results	contracts/main/contracts/pool/PriceOracle.sol	Medium	Fixed
Withdraw request lacks deadline check	contracts/pool/UFarmPool.sol	Low	Acknowledged
Unhandled Chainlink revert	contracts/main/contracts/pool/PriceOracle.sol	Low	Fixed
The 'Fund Editor' role is too powerful	contracts/main/contracts/fund/UFarmFund.sol	Low	Fixed
Use of unchecked block	CoreWhitelist.sol, UFarmCore.sol, UFarmPool.sol	Low	Fixed
Signature malleability	contracts/main/shared/ECDSARecover.sol	Low	Fixed
Typo in `UniV2RemoveLiquidtyArgs`	contracts/main/contracts/controllers/UnoswapV2Controller.sol	Info	Fixed
depositLockupPeriod is not used	contracts/main/contracts/pool/IUFarmPool.sol	Info	Acknowledged
EnumerableSet can be used	contracts/main/contracts/core/UFarmCore.sol	Info	Fixed

Issue	Contract	Risk Level	Status
Hardcoded values instead of variables	contracts/main/contracts/core/UFarmCore.sol	**Info**	Acknowledged
A ternary operation can be simplified	contracts/main/contracts/pool/UFarmPool.sol	**Info**	Acknowledged
Double-check for zero value	contracts/main/contracts/core/CoreWhitelist.sol	**Info**	Fixed
Cast from uint256 to uint8	contracts/main/contracts/permissions/UFarmPermissionsModel.sol	**Info**	Fixed

4. General Recommendations

This section contains general recommendations on how to improve the overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. totalCost parameter may be manipulated

Risk Level: Critical

Status: Fixed in e089ebbdcd91924c4bd2174a822b6c235b3cdda8.

Contracts:

- contracts/pool/UFarmPool.sol

Location: Function: `getTotalCost()`.

Description:

`getTotalCost()` function calculates total value of assets in pool. This value also includes LP tokens. However, due to unsafe calculation of value of LP tokens `getTotalCost()` output may be manipulated, that allows attacker to gain extra value tokens when withdrawing from pool. Attacker can perform swap in a pair that is included in a `getTotalCost()` calculation, thus resulting in a cost increase, that allows him to withdraw more value tokens, than he actually should. Then he can swap back tokens that he received earlier and get profit.

Remediation:

Since function includes both Uniswap V2 and Uniswap V3 pools, several mitigation steps should be performed. For Uniswap V2 we would recommend to stick to the following solution when unwrapping LP tokens: https://cmichel.io/pricing-lp-tokens/

As for Uniswap V3 we would recommend to calculate TWAP instead of directly retrieving `sqrtPriceX96`

5.2. Unsafe safeTransfer()

Risk Level: High

Status: Fixed in bdada8aa

Contracts:

- contracts/main/shared/SafeOPS.sol

Location: Lines: `89, 99`. Function: `_safeTransferERC20, _safeTransferFromERC20`.

Description:

Function `_safeTransferERC20` and `_safeTransferFromERC20` have two issues:

1) Using the `IERC20` interface for token interactions could potentially cause a Denial of Service (DoS) due to the inaccurate implementation of this interface by some ERC20 tokens. Notable examples include USDT and BNB on the Ethereum mainnet.

This is because, starting from version 0.4.22, Solidity checks for the size of the return value. If the function interface suggests that the function returns a boolean and the function implementation doesn't return anything, Solidity will stop the function's execution and call revert.

2) The `transfer` and `transferFrom` functions should always return `true`. However, these return values are not checked. It also may lead to problems with some ERC-20 token implementations.

Remediation:

Consider fixing the problems above or using the SafeERC20 library made by OpenZeppelin.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol
- https://twitter.com/Uniswap/status/1072286773554876416

5.3. Centralization risks

Risk Level: Medium

Status: Not fixed

Description:

The overall design of the protocol assumes significant privileges concentrated in the roles of the fund managers and the protocol admins.

While this is understandable and expected for this type of product, there are still ways to ensure that the protocol operates in a more permissionless manner, and the users have more control over their funds.

Remediation:

Reduce the possibilities of the malicious actions of the privilege accounts, make them more transparent so that the loss of funds cannot become a side effect of seemingly innocuous action.

Reduce the possibilities of intentional or unintentional locking of the protocol, leave the options to perform emergency withdraw.

5.4. Incorrect calculations in getCostControlledERC20

Risk Level: Medium

Status: Fixed in bdada8aaadad6a1b927580b5d5f3f5663b2c6403

Contracts:

- contracts/controllers/UnoswapV2Controller.sol

Location: Function: `getCostControlledERC20`.

Description:

When calculating token amounts from lp token `getCostControlledERC20` function performs the following calculation: `(reserve0 * lpAmount) / totalSupply`. However, this calculation lacks `feeLiquidity` parameter that occurs if pair has fees. This leads to improper calculations of token amounts.

Remediation:

Consider adding `feeLiquidity` parameter the same way as done in https://github.com/Uniswap/v2-periphery/blob/0335e8f7e1bd1e8d8329fd300aea2ef2f36dd19f/contracts/libraries/UniswapV2LiquidityMathLibrary.sol#L90

5.5. Possible DoS via malicious ERC20 asset

Risk Level: Medium

Status: Fixed in bdada8aaadad6a1b927580b5d5f3f5663b2c6403

Contracts:

- contracts/controllers/UnoswapV3Controller.sol

Location: Function: `delegatedCollectAllFees`.

Description:

`delegatedCollectAllFees` collects fees from Uniswap V3 positions and the includes collected tokens to ERC20 assets via `addERC20` . However, it doesn't validate that added tokens are included in whitelist, which may result in addition of non-whitelisted ERC20 token to assets. Malicious fund member may transfer his ERC721 position token to pool address and then call `delegatedCollectAllFees` . In case non-whitelisted token will be added to assets `getTotalCost()` computation will revert because there will be no info for such token. This will make pool contract inoperable.

Remediation:

Consider checking that any token that is added to ERC20 assets is whitelisted.

5.6. Not whitelisted tokens could be in the path in Unoswap Controllers

Risk Level: Medium

Status: Fixed in commit e089ebbdcd91924c4bd2174a822b6c235b3cdda8

Contracts:

- contracts/controllers/UnoswapV2Controller.sol

Location: Function: `delegateSwapExactTokensForTokens`.

Description:

`delegateSwapExactTokensForTokens` function checks that only first and last token in swap path are added in whitelist. This leads to the fact that fund member may craft a malicious swap path, that will for example swap 10k USDT to 1 Wei WETH via intermediate pair. This issue results in a rug pull potential from a fund.

Remediation:

Consider checking that all tokens in swap path are added in whitelist.

5.7. Function `delegateIncreaseLiquidity`() in not checking position owner

Risk Level: Medium

Status: Fixed in bdada8aa

Contracts:

- contracts/main/contracts/controllers/UnoswapV3Controller.sol

Location: Lines: `405`. Function: `delegateIncreaseLiquidity`.

Description:

The `UniswapV3` contract's `delegateIncreaseLiquidity` function doesn't verify the ownership of the position where liquidity will be increased. This could potentially lead to the loss of funds from the `Ufarm` pool if the pool manager accidentally increases liquidity in an uncontrolled position.

Remediation:

Consider checking the ownership of the liquidity position before increasing its liquidity.

5.8. Possible DoS in _processWithdrawal

Risk Level: Medium

Status: Fixed in bdada8aa

Contracts:

- contracts/main/contracts/pool/UFarmPool.sol

Location: Lines: `541`. Function: `_processWithdrawal`.

Description:

Function `_processWithdrawal()` is too heavy in terms of gas consumption. It contains too many loops and external calls and already suffers from the `Stack too deep` error. With a lot of tokens and positions in the pool, this function may lead to the DoS during the withdrawal process due to the revert. Users may lose the ability to make withdrawals from the pool until the manager manually removes an excessive amount of tokens, leading to DoS.

Remediation:

Consider optimizations of withdrawal logic in the pool contract, integrating some off-chain computations.

5.9. Incorrect calculations in getAmountInRes()

Risk Level: Medium

Status: Fixed in bdada8aa

Contracts:

- contracts/main/contracts/controllers/UnoswapV2Controller.sol

Location: Lines: 573. Function: `getAmountInRes`.

Description:

The denominator in the function `getAmountInRes` is calculated incorrectly. This calculation may lead to DoS due to underflow or a greatly exaggerated value of `amointIn` in return.

```
function getAmountInRes(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut
    ) private pure returns (uint256 amountIn) {
        if (amountOut == 0) revert INSUFFICIENT_OUTPUT_AMOUNT();
        if (reserveIn == 0 || reserveOut == 0) revert
INSUFFICIENT_LIQUIDITY();
        uint256 numerator = reserveIn * (amountOut) * (1000);
        uint256 denominator = reserveOut - (amountOut) * (997); // incorrect
        amountIn = (numerator / denominator) + (1);
    }
```

Correct version:

```
uint256 denominator = (reserveOut - amountOut) * 997;
```

Remediation:

Consider fixing the calculations.

References:

- https://github.com/Uniswap/v2-periphery/blob/master/contracts/libraries/UniswapV2Library.sol#L53

5.10. PriceOracle doesn't check If the Arbitrum sequencer is down in Chainlink feeds

Risk Level: Medium

Status: Fixed in af73ff7a

Contracts:

- contracts/main/contracts/pool/PriceOracle.sol

Location: Lines: 119. Function: `getNormalizedPrice`.

Description:

Using Chainlink in L2 chains such as Arbitrum requires checking if the sequencer is down to avoid prices from looking like they are fresh although they are not.

The bug could be leveraged by malicious actors to take advantage of the sequencer downtime.

Remediation:

Consider using the next code example from ChainLink documentation before getting `latestRoundData` for the codebase on Arbitrum:

```
(
    /*uint80 roundID*/,
    int256 answer,
    uint256 startedAt,
    /*uint256 updatedAt*/,
    /*uint80 answeredInRound*/
) = sequencerUptimeFeed.latestRoundData();

// Answer == 0: Sequencer is up
// Answer == 1: Sequencer is down
bool isSequencerUp = answer == 0;
if (!isSequencerUp) {
    revert SequencerDown();
}

// Make sure the grace period has passed after the
// sequencer is back up.
uint256 timeSinceUp = block.timestamp - startedAt;
if (timeSinceUp <= GRACE_PERIOD_TIME) {
    revert GracePeriodNotOver();
}
```

References:

- https://docs.chain.link/data-feeds/l2-sequencer-feeds#example-code

5.11. Chainlink's price feed might return stale or incorrect results

Risk Level: Medium

Status: Fixed in af73ff7a

Contracts:

- contracts/main/contracts/pool/PriceOracle.sol

Location: Lines: `130`. Function: `getNormalizedPrice`.

Description:

The `PriceOracle` contract queries a Chainlink oracle using `latestRoundData()`. If Chainlink encounters issues when starting a new round or reaching consensus on the new value for the oracle (for example, Chainlink nodes abandon the oracle, network congestion occurs, or the Chainlink system is attacked), this may lead to the contract's consumers using outdated, stale or incorrect data (when oracles are unable to submit, no new round is initiated).

Remediation:

Consider checking the return values from ChainLink. Add the following checks:

```
...
( roundId, rawPrice, , updateTime, answeredInRound ) =
AggregatorV3Interface(XXXXX).latestRoundData();
require(rawPrice > 0, "Chainlink price <= 0");
require(updateTime != 0, "Incomplete round");
require(answeredInRound >= roundId, "Stale price");
...
```

References:

- https://github.com/code-423n4/2021-05-fairside-findings/issues/70
- https://consensys.io/diligence/audits/2021/09/fei-protocol-v2-phase-1/#chainlinkoraclewrapper-latestrounddata-might-return-stale-results

5.12. Withdraw request lacks deadline check

Risk Level: Low

Status: Acknowledged

Contracts:

- contracts/pool/UFarmPool.sol

Location: Function: `validateWithdrawalRequest`.

Description:

`WithdrawRequest` structure lacks deadline parameter, however `DepositRequest` structure has it. This leads to the fact that withdrawal request cannot be invalidated and will last forever until used

Remediation:

Consider adding deadline parameter to the withdrawal request

5.13. Unhandled Chainlink revert

Risk Level: Low

Status: Fixed in af73ff7a

Contracts:

- contracts/main/contracts/pool/PriceOracle.sol

Location: Lines: `130`. Function: `getNormalizedPrice`.

Description:

Chainlink's multisigs can instantaneously block access to price feeds as needed. As a result, to avoid denial of service situations, it's advisable to use a defensive approach when querying Chainlink price feeds, utilizing Solidity's try/catch structure. This ensures that even if the call to the price feed fails, the caller contract maintains control and can safely and explicitly handle any errors.

Remediation:

Consider surrounding the call to `latestRoundData()` with `try/catch` instead of calling it directly. In a scenario where the call reverts, the catch block can be used to call a fallback oracle or handle the error in any other suitable way.

5.14. The '`Fund Editor`' role is too powerful

Risk Level: Low

Status: Fixed in af73ff7a

Contracts:

- contracts/main/contracts/fund/UFarmFund.sol

Location: Lines: 441. Function: `_canUpdatePermissions`.

Description:

The `Fund Editor` role grants users the ability to edit fund information, invite new members, manage existing members' permissions, and block/unblock members, according to the documentation. However, this role contains a powerful permission, namely `Permissions.Fund.UpdatePoolPermissions`.This permission allows role-holders to grant themselves additional permissions and gain almost the same level of control as the fund owner, including managing funds.

Remediation:

Consider leaving the permission `Permissions.Fund.UpdatePoolPermissions` only to the fund owner.

5.15. Use of unchecked block

Risk Level: Low

Status: Fixed in af73ff7a

Contracts:

- CoreWhitelist.sol,
- UFarmCore.sol,
- UFarmPool.sol

Description:

The codebase has several uses for unchecked code blocks. They should be used with caution. It is better to avoid such blocks if it is not really necessary.

```
contracts/main/contracts/core/CoreWhitelist.sol:
    51: unchecked {
    75: unchecked {
    95: unchecked {
contracts/main/contracts/core/UFarmCore.sol:
    282: unchecked {
contracts/main/contracts/pool/UFarmPool.sol:
    818: unchecked {
```

Remediation:

Consider removing unchecked blocks.

5.16. Signature malleability

Risk Level: Low

Status: Fixed in c7b2b82f.

Contracts:

- contracts/main/shared/ECDSARecover.sol

Location: Lines: 105. Function: `recoverAddress`.

Description:

The code snippet provided shows that the value of v in the signature is incremented by 27 if it's less than 27.

```
contracts/main/shared/ECDSARecover.sol:
  110:         if (v < 27) v += 27;
```

The value of the s variable is not validated to be less than `0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0` too.

Both problems can lead to signature malleability, where a valid signature can have multiple valid representations, potentially leading to security vulnerabilities. It has no impact on your current implementations of the contracts but may lead to some problems in the future.

Remediation:

To mitigate this, it's recommended to enforce a strict check on the v and s values in the signature. This means that only values of 27 or 28 as v and only s less than

`0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0` should be accepted to prevent the signature from having multiple valid representations.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol

5.17. Typo in `UniV2RemoveLiquidtyArgs`

Risk Level: Info

Status: Fixed in af73ff7a

Contracts:

- contracts/main/contracts/controllers/UnoswapV2Controller.sol

Location: Lines: 284.

Description:

Structure `UniV2RemoveLiquidtyArgs` contains a typo in its name.

Remediation:

Rename the struct to `UniV2RemoveLiquidityArgs`.

5.18. depositLockupPeriod is not used

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/main/contracts/pool/IUFarmPool.sol

Location: Lines: 28.

Description:

The variable `depositLockupPeriod` from structure `CreationSettings` is not used anywhere in the code of the contracts.

Remediation:

Consider removing unused variables.

5.19. EnumerableSet can be used

Risk Level: Info

Status: Fixed in af73ff7a

Contracts:

- contracts/main/contracts/core/UFarmCore.sol

Location: Lines: 153. Function: `createFund`.

Description:

Contract `UFarmCore` has 3 different variables to store the data about existing funds. It increases the risk of future bugs with an inconsistent state. For example, in situations when some of these variables will not be updated by mistake along with others.

```
++fundsCount;
funds[fundsCount] = fund;
isFund[fund] = true;
```

Remediation:

Consider using EnumerableSet to store the data about existing funds. It will simplify the code and decrease the risks of future bugs.

5.20. Hardcoded values instead of variables

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/main/contracts/core/UFarmCore.sol

Location: Lines: 220. Function: `setProtocolCommission`.

Description:

The range for the commission in function `setProtocolCommission` is checked with hardcoded magic values. It may be forgotten in future updates and break the logic of protocol if the possible range is changed.

```
function setProtocolCommission(
        uint256 _protocolCommission
    )
        external
        override
        valueInRange(_protocolCommission, 0, 1e17)
        ownerOrHaveTwoPermissions(uint8(Permissions.UFarm.Member),
uint8(Permissions.UFarm.ManageFees))
    {
        protocolCommission = _protocolCommission;
    }
```

Remediation:

Consider replacing the hardcoded value with a variable.

5.21. A ternary operation can be simplified

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/main/contracts/pool/UFarmPool.sol

Location: Lines: 498. Function: `withdraw`.

Description:

Ternary operation in the calculation of `availableToWithdraw` function is unnecessary because the code has a check that `totalUserShares` is not lower than `mandatoryShares` right before the calculations.

```
if (totalUserShares < mandatoryShares) revert
InvalidWithdrawalAmount(totalUserShares, 0);

uint256 availableToWithdraw = totalUserShares > mandatoryShares
    ? totalUserShares - mandatoryShares
    : 0;
```

For gas optimization and code clearness ternary operation could be replaced with a subtraction.

```
if (totalUserShares < mandatoryShares) revert
InvalidWithdrawalAmount(totalUserShares, 0);

uint256 availableToWithdraw = totalUserShares - mandatoryShares;
```

Remediation:

Consider replacing the ternary operation with subtraction.

5.22. Double-check for zero value

Risk Level: Info

Status: Fixed in [af73ff7a]

Contracts:

- contracts/main/contracts/core/CoreWhitelist.sol

Location: Lines: 60. Function: `whitelistProtocolsWithControllers`.

Description:

Contract `CoreWhitelist` is checking the protocol variable not to be equal to zero twice. First time in the `whitelistProtocolsWithControllers` function, the second time in the modifier of `_whitelistProtocol` function. It is not necessary and can be reduced to one check for gas optimization.

```
function whitelistProtocolsWithControllers(
        bytes32[] memory _protocolNames,
        address[] memory _protocolControllers
    ) public virtual override {
        unchecked {
            uint256 controllersLength = _protocolControllers.length;

            if (_protocolNames.length != controllersLength) revert
ArraysLengthMismatch();
            _nonEmptyArray(controllersLength);

            for (uint256 i; i < controllersLength; ++i) {
                address controller = _protocolControllers[i];
                bytes32 protocol = _protocolNames[i];
                if (controller == address(0)) revert ZeroAddress();
                if (protocol == bytes32(0)) revert ZeroValue(); // first check

                _whitelistProtocol(protocol, controller);
            }
        }
```

```
    }

function _whitelistProtocol(
        bytes32 _protocol,
        address _controller
    ) internal nonZeroBytes32(_protocol) { // second check
        if (__protocols.add(_protocol)) {
            controllers[_protocol] = _controller;
            emit ProtocolAdded(_protocol, _controller);
        } else revert ActionAlreadyDone();
    }
```

Remediation:

Consider reducing the number of similar checks for gas optimization.

5.23. Cast from uint256 to uint8

Risk Level: Info

Status: Fixed in [af73ff7a]

Contracts:

- contracts/main/contracts/permissions/UFarmPermissionsModel.sol

Location: Lines: 79. Function: `_updatePermissions`.

Description:

If a revert occurs in function `_updatePermissions`, the value of `newPermissions` variable passed to `PermissionAlreadyGranted` will be incorrect due to the downcast from `uint256` to `uint8`.

```
function _updatePermissions(address account, uint256 newPermissions) internal
{
        if (_accountMask[account] != newPermissions) {
            _accountMask[account] = newPermissions;
            emit PermissionsUpdated(account, newPermissions);
        } else {
            revert PermissionAlreadyGranted(account, uint8(newPermissions));
        }
    }
```

Remediation:

Fix the size of the uint passed to `PermissionAlreadyGranted`.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.