DEFIMOON

be secure

Smart Contract Audit Report

December, 2023

Aspis Protocol

Aspis

DEFIMOON PROJECT

Audit and
Development

CONTACTS

defimoon.org
audit@defimoon.org
defimoon_org
defimoonorg
defimoon
defimoonorg

DEFIMOON

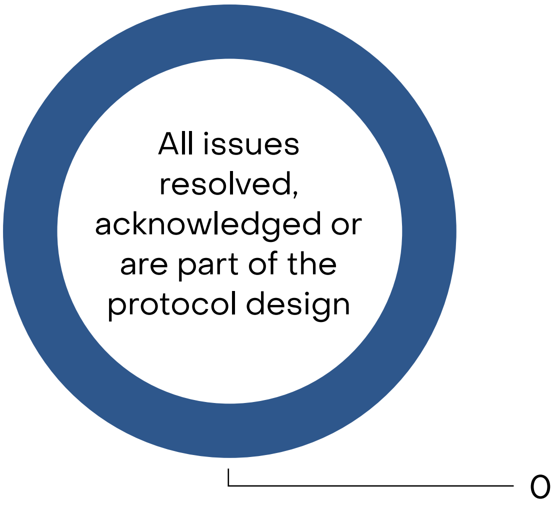
be secure

13 December 2023

This audit report was prepared by DefiMoon for Aspis Finance.

Audit information

Description	Aspis is a factory of on-chain funds where investors can trust their funds to the managers without need to trust to the manager, by securing their relatinships into on-chain agreement that ensures control and execution over the terms, including control over manager's operations
Audited files	contracts/aspis/*, contracts/decoders/*, contracts/votings/*
Timeline	14 August 2023 - 13 December 2023
Approved by	Artur Makhnach, Kirill Minyaev
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Manual Review
Project Site	https://app.aspis.finance/
Source code	cc05ccfb81c10cd0b7ea9370483df788cb9f31ed
Reaudit Source code	21074bdfe8fb562d775f6549b7452dd139c899eb
Network	EVM-like
Status	Passed



🟥	High Risk	A fatal vulnerability that can cause the loss of all Tokens / Funds.
🟧	Medium Risk	A vulnerability that can cause the loss of some Tokens / Funds.
🟩	Low Risk	A vulnerability which can cause the loss of protocol functionality.
!	Informational	Non-security issues such as functionality, style, and convention.

Disclaimer

This audit is not financial, investment, or any other kind of advice and could be used for informational purposes only. This report is not a substitute for doing your own research and due diligence should always be paid in full to any project. Defimoon is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Defimoon has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Defimoon is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. Defimoon has no connection to the project other than the conduction of this audit and has no obligations other than to publish an objective report. Defimoon will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Defimoon assumes that the provided information and materials were not altered, suppressed, or misleading. This report is published by Defimoon, and Defimoon has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Defimoon. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

Audit Information

Defimoon utilizes both manual and automated auditing approach to cover the most ground possible. We begin with generic static analysis automated tools to quickly assess the overall state of the contract. We then move to a comprehensive manual code analysis, which enables us to find security flaws that automated tools would miss. Finally, we conduct an extensive unit testing to make sure contract behaves as expected under stress conditions.

In our decision making process we rely on finding located via the manual code inspection and testing. If an automated tool raises a possible vulnerability, we always investigate it further manually to make a final verdict. All our tests are run in a special test environment which matches the "real world" situations and we utilize exact copies of the published or provided contracts.

While conducting the audit, the Defimoon security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Defimoon assesses the risks and assigns a risk level to each section together with an explanatory comment.

Audit overview

No major vulnerabilities have been found

All issues have been resolved, marked as part of the protocol design or acknowledged.

The Aspis team has done an excellent job of fixing vulnerabilities and bugs.
Each finding has been given due consideration to improve the safety and reliability of the Aspis Protocol.

The Aspis Protocol protocol can be deployed in production.

Summary of findings

ID	Description	Severity	Status
DFM-1	Reentrancy	High Risk	Resolved
DFM-2	Incorrect getting token balance	High Risk	Resolved
DFM-3	Lack of reverse logic when deleting trading tokens	High Risk	Protocol's Design
DFM-4	Changing trading tokens in real time	High Risk	Protocol's Design
DFM-5	Incorrect calculation of token price	Medium Risk	Protocol's Design
DFM-6	No checks for duplicates	Medium Risk	Resolved
DFM-7	Using user funds	Medium Risk	Protocol's Design
DFM-8	Incorrect slippage check condition	Medium Risk	Resolved
DFM-9	The slippage check logic is not used	Medium Risk	Resolved
DFM-10	Incorrect condition	Low Risk	Resolved
DFM-11	Using the wrong data type	Low Risk	Acknowledged
DFM-12	Changing the timing of the fundraising	Low Risk	Protocol's Design
DFM-13	Lack of mapping changing	Low Risk	Resolved
DFM-14	Withdrawal of user funds	Low Risk	Resolved
DFM-15	Value is not receiving	Low Risk	Acknowledged
DFM-16	Possible incorrect token creation	Low Risk	Resolved
DFM-17	Emergency stop does not apply to all functions	Low Risk	Protocol's Design
DFM-18	Incorrect removal of an element from an array	Low Risk	Resolved
DFM-19	Execution error with a large number of deposits	Low Risk	Acknowledged

Summary of other/ optimizations

ID	**Description**	**Severity**	
O-DFM-1	Redundant variable	Informational	Resolved
O-DFM-2	Optimization of conditions	Informational	Resolved
O-DFM-3	Unused function	Informational	Resolved
O-DFM-4	Redundant use of SafeMath	Informational	Resolved
O-DFM-5	Type not explicitly specified	Informational	Resolved
O-DFM-6	Unused variable	Informational	Resolved
O-DFM-7	Local variable optimization	Informational	Resolved
O-DFM-8	Lots of duplicate functionality	Informational	Resolved/ Not Actual
O-DFM-9	Excessive use of iterating over arrays	Informational	Resolved
O-DFM-10	Incorrect function type specified	Informational	Resolved
O-DFM-11	Implicit return	Informational	Resolved
O-DFM-12	Unused storage variables	Informational	Acknowledg ed
O-DFM-13	Redundant condition	Informational	Resolved
O-DFM-14	Redundant use of type int256	Informational	Resolved
O-DFM-15	Typo in error message	Informational	Resolved
O-DFM-16	Redundant use of the storage variable	Informational	Resolved
O-DFM-17	Function does not match the name	Informational	Acknowledg ed
O-DFM-18	Expression Simplification	Informational	Resolved
O-DFM-19	Redundant use of delete	Informational	Resolved
O-DFM-20	Additional checks on deletion	Informational	Resolved
O-DFM-21	Reverse iteration optimization	Informational	Resolved
O-DFM-22	Checking the deposit amount	Informational	Resolved
O-DFM-23	Using calldata instead of memory	Informational	Resolved
O-DFM-24	Visibility modifiers for functions and variables	Informational	Partially Resolved
O-DFM-25	Loops optimizations	Informational	Partially Resolved

Application security checklist

Compiler errors	Passed
Possible delays in data delivery	Passed
Timestamp dependence	Passed
Integer Overflow and Underflow	Passed
Race Conditions and Reentrancy	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Private user data leaks	Passed
Malicious Events Log	Passed
Scoping and Declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Design Logic	Passed
Cross-function race conditions	Passed

Detailed Audit Information

Contract Programming

Solidity version not specified	Passed
Solidity version too old	Passed
Integer overflow/underflow	Passed
Function input parameters lack of check	Passed
Function input parameters check bypass	Passed
Function access control lacks management	Passed
Critical operation lacks event log	Passed
Human/contract checks bypass	Passed
Random number generation/use vulnerability	Passed
Fallback function misuse	Passed
Race condition	Passed
Logical vulnerability	Passed
Other programming issues	Passed

Code Specification

Visibility not explicitly declared	Passed
Variable storage location not explicitly declared	Passed
Use keywords/functions to be deprecated	Passed
Other code specification issues	Passed

Gas Optimization

Assert () misuse	Passed
High consumption ‘for/while’ loop	Passed
High consumption ‘storage’ storage	Passed
“Out of Gas” Errors	Passed

Findings

DFM-1 «Reentrancy» | AspisPool

Severity: High Risk

Status: Resolved

Description: The transferAsset function sends ETH, which makes it possible to launch a reentrancy attack by calling deposit tokens when receiving ETH. The transferAsset function uses _tokenSupply as arguments, which means that after the deposit, the transferAsset function will continue executing with modified token balances, but with an outdated _tokenSupply value, resulting in more tokens being calculated.

Recommendation: To protect against reentrancy, we recommend changing the logic of the functions or using the nonReentrant modifier or sending ETH at the last moment.

DFM-2 «Incorrect getting token balance» | AspisPool

Severity: High Risk

Status: Resolved

Description: The getBalance function does not use return to the token balance.

Recommendation: Use return for the token balance, otherwise the function will return 0 as the token balance, which will lead to incorrect calculations.

DFM-3 «Lack of reverse logic when deleting trading tokens» | AspisConfiguration | AspisPool

Severity: High Risk

Status: Protocol's Design

Comment: Trading tokens collection is managed by Aspis guardian and can only be called by him, which is a trusted actor. Aspis guardian will modify collections of trading and deposit tokens very responsibly and only when necessary, for example, it may remove broken tokens or tokens with discontinued oracle feed.

Description: In the AspisConfiguration::removeFromTradingTokens function, deleting an address does not change the value in the AspisConfiguration::tradingTokens mapping.

In the AspisConfiguration::_addDepositTokens function, only trading tokens can be added as a token deposit, however, when a trading token is deleted, it is not removed from the token deposit.

As a result, if the trading token is deleted, it will still remain as a deposit token and users will be able to make deposits, but the AspisPool::getCurrentTokenPrice function will calculate an incorrect price.

Recommendation: We recommend removing the token from AspisConfiguration::depositTokens when deleting a trading token.

DFM-4 «Changing trading tokens in real time» | AspisConfiguration | AspisPool

Severity: High Risk

Status: Protocol's Design

Comment: Calling the AspisRegistry::removeSupportedTradingTokens function is only available for Aspis Guardian (Protocol's team).

This is an emergency functionality that ONLY Aspis guardian may activate. For example, in case token chainlink feed becomes deprecated. It's guardian's responsibility to make sure it doesn't lead to value loss.

Description: The AspisConfiguration::removeFromTradingTokens function can be called after the fundraising has started, which will lead to violations of the AspisPool::transferAsset and AspisPool::getCurrentTokenPrice functions if deposits have been made in the remote token before.

Recommendation: We recommend disabling the use of AspisConfiguration::removeFromTradingTokens and other functions that may affect the operation of the protocol after fundraising has begun.

DFM-5 «Incorrect calculation of token price» | AspisPool | AspisGovernanceERC20

Severity: Medium Risk

Status: Protocol's Design

Comment: We acknowledge that sending ETH directly to pool will alter the price of LP token, but thats is part of business logic and how these pools are suppose to function.

Description: The getCurrentTokenPrice function uses address(this).balance or token.balanceOf(address(this)) to calculate _poolValue, but uses getTokenSupply() when calculating price.

The difference is that the balance of the contract can be changed by simply sending ETH (or via selfdestruct()) or tokens to it, but token.totalSupply() is only changed when deposit or withdraw is called. In such a case, anyone can artificially increase the deposit price and reduce the withdraw price for those who have previously deposited.

In addition, AspisGovernanceERC20 includes mint and burn functions that can be called via voting and that change token.totalSupply() but do not change token balances, which will also affect the price calculation.

Recommendation: We recommend redesigning the pricing mechanism using balance and total supply records instead of using real-time data.

DFM-6 «No checks for duplicates» | AspisConfiguration | AspisPool

Severity: Medium Risk

Status: Resolved

Description: The _addDepositTokens, _addToWhitelist and _addToTrustedProtocols functions do not have checks for duplicates, which can lead to protocol errors. For example, the AspisPool::getCurrentTokenPrice function will calculate the price incorrectly if the tokens are repeated.

Recommendation: We recommend adding mapping(address => bool) public isExistent and using them to check for duplicates and whether the address is in the array, so as not to use iteration.

DFM-7 «Using user funds» | AspisPool

Severity: Medium Risk

Status: Protocol's Design

Comment: Most of the things are intended behaviour of the business logic. Users fund are moved only after a proposal is made and voted. So users are always made aware of their assets being transferred.

Description: The directAssetTransfer, execute, and approveTokenTransfer functions can use user funds, causing users to receive fewer tokens or ETH when they call withdraw.

First, the directAssetTransfer and execute functions use _ethValue but are nonpayable, so the _ethValue will be ETH deposited by users.

Secondly, user tokens can be used in any of these functions, since there are no balance checks.

Third, the directAssetTransfer function uses the block.timestamp > configuration.finishTime() constraint, but finishTime can be changed at any time (see DFM-12). In addition, the end of fundraising does not mean that users have already withdrawn their funds.

Recommendation: If this is not part of the mechanics of your protocol, we recommend adding logic to prevent spending users funds. Otherwise, we recommend warning users that their funds may be spent.

DFM-8 «Incorrect slippage check condition» | AspisPool

Severity: Medium Risk

Status: Resolved

Description: The meetsSlippageTolerance function uses a multiplication by 100 to check for slippage, but SLIPPAGE_TOLERANCE_PERCENTAGE is 500, which causes _slippage > SLIPPAGE_TOLERANCE_PERCENTAGE to always be false.

Recommendation: You should multiply by 10000 if you want to use 5% as a valid slippage.

DFM-9 «The slippage check logic is not used» | AspisPool

Severity: Medium Risk

Status: Resolved

Description: The decodeAndCall function uses executeLowLevelCall to make a call to the contract, after which meetsSlippageTolerance is used to check the slippage, but meetsSlippageTolerance will never be used because executeLowLevelCall uses an assembly return operation that terminates the execution.

Recommendation: We recommend using the default Solidity return operation.

DFM-10 «Incorrect condition» | AspisLibrary

Severity: Low Risk

Status: Resolved

Description: If _freezePeriod == 0, then the withdraw period should always be in effect and the isWithdrawalWithinWindow function should return true, but if _currentRelativeDay == 0, then the function returns false.

Recommendation: A corrected and simplified implementation is specified in O-DFM-2.

DFM-11 «Using the wrong data type» | AspisLiquidityCalculator

Severity: Low Risk

Status: Acknowledged

Description: The DataFeed structure uses the uint256 type to store the decimals variable, but in the addPriceFeed function the oracleAddress and decimals variables are placed in the same storage slot (32 bytes). Since address is 20 bytes, there are only 12 bytes left to store decimals (like uint96), but if decimals greater than type(uint96).max is passed, the data will be written incorrectly.

Recommendation: We recommend using no more than uint96 to write decimals in the DataFeed structure.

DFM-12 «Changing the timing of the fundraising» | AspisConfiguration

Severity: Low Risk

Status: Protocol's Design

Comment: Intended by the protocol’s design. Since changing the timing of the fundraising can be passed only through voting, for example, to increase the duration of the fundraising period.

Description: In the setFundraisingFinishTime function, the finishTime variable can be changed even during fundraising, which can negatively affect the user experience.

Recommendation: We recommend that you disable the ability to change key variables while fundraising is active.

DFM-13 «Lack of mapping changing» | AspisConfiguration

Severity: Low Risk

Status: Resolved

Description: In the removeFromTradingTokens function, deleting an address does not change the value in the tradingTokens mapping.

Recommendation: You can fix it like this:

```
// ...
for (uint8 j=0; j < tradingTokensArray.length; j++) {
    if (tradingTokensArray[j] == _tradingToken) {
        tradingTokens[_tradingToken] = false;
        uint256 last = tradingTokensArray.length - 1;
        tradingTokensArray[j] = tradingTokensArray[last];
        tradingTokensArray.pop();
        break;
    }
}
// ...
```

DFM-14 «Withdrawal of user funds» | AspisPool | *Decoders

Severity: Low Risk

Status: Resolved

Description: The AspisPool::decodeAndCall function is designed to swap tokens, but the recipient is specified dynamically and may not be equal to the address of the AspisPool contract, as a result of which a part of the tokens that does not exceed slippage can be sent to another address.

Recommendation: We recommend that you statically specify the address of the contract as the recipient or add the appropriate check.

DFM-15 «Value is not receiving» | AspisGuardian

Severity: Low Risk

Status: Acknowledged

Comment: Payable modifier is not required since the function doesnt expect ether to be sent for calling this function. It uses the _ethValue to transfer ETH of the pool

Description: The execute function uses _ethValue when making the call, but the function is nonpayable.

Recommendation: We recommend adding the payable modifier or dropping _ethValue if it's not needed.

DFM-16 «Possible incorrect token creation» | AspisPoolFactory

Severity: Low Risk

Status: Resolved

Description: The createDAOToken function deploys the token without using a proxy, although AspisGovernanceERC20 uses an upgradeable architecture.

Recommendation: Make sure you're unwrapping the token correctly and fix it if you're not.

DFM-17 «Emergency stop does not apply to all functions» | AspisPool

Severity: Low Risk

Status: Protocol's Design

Comment: This is intended by the protocol's design since investors need to be able to withdraw their funds from the vault in case of emergency. That's why this function was created. The emergency stop is meant for emergencies, to stop all possible activities from the manager and stop all voting so investors can withdraw their funds in case of the manager's misconduct or any vulnerability found.

Description: The withdraw and withdrawCommission functions can be called when an emergency stop is active. The emergency stop is probably meant for emergencies or contract upgrades, so it's best to disable all main functions for that time.

If we talk about upgrades, then sometimes additional utility functions, setters or functions for data migration can be called after the deployment of the contract, so it is better not to call the main functions in the interval between the deployment and the call of additional initialization functions.

If we talk about emergencies (bugs or vulnerabilities), then it is definitely better to disable the use of key functions until a fix and upgrade.

Recommendation: We recommend disabling all basic functions while emergency stop is active. Of course, in this case, user funds remain locked in the contract, so to mitigate this problem, you can add a timestamp, after which it will be possible to use some functions even despite emergency stop.

DFM-18 «Incorrect removal of an element from an array» | AspisPool

Severity: Low Risk

Status: Resolved

Description: The withdraw function uses delete to remove an element from an array, but delete does not reduce the size of the array, it only clears the value of the variable, which can result in the array containing empty values.

Recommendation: We recommend using pop() to remove an element from an array
(example in O-DFM-21).

DFM-19 «Execution error with a large number of deposits» | AspisPool

Severity: Low Risk

Status: Acknowledged

Comment: Execution error can not block the entire pool, it can only block the individual investor who tries to make hundreds or thousands of small deposits into the Vault. It is very unlikely that a good intended user will try to do thousands of transactions to individual Vault. In such case, he will only harm himself and not stop the entire pool's operations.

Description: Each new user deposit is recorded in a dynamic array, and the withdraw function uses an iteration over all of the user's deposits, resulting in an "out of gas" error with a large number of iterations and the user will never be able to call withdraw.

Recommendation: We recommend adding an additional withdraw function, which can be used to withdraw only one or several deposits, so as not to iterate over the entire array.

Other/ Optimization

O-DFM-1 «Redundant variable» | AspisLibrary

Status: Resolved

Description: Redundant variable MIN_TIME_UNIT.

LOC:9

```
uint256 private constant MIN_TIME_UNIT = 1 seconds;
```

LOC:51

```
uint256 _countPastSeconds = (_currentTime - _fundraisingFinishTime) /
MIN_TIME_UNIT;
```

The minimum unit of time in Solidity is the second, so MIN_TIME_UNIT = 1, and dividing by 1 is redundant.

Recommendation: We recommend deleting the MIN_TIME_UNIT variable for optimization and gas saving purposes.

O-DFM-2 «Optimization of conditions» | AspisLibrary

Status: Resolved

Description: The check conditions in the isWithdrawalWithinWindow function can be simplified.

LOC:57-65

```
if (_currentRelativeDay == 0) {
    return false;
} else if (_currentRelativeDay > 0) {
    if (_currentRelativeDay > _freezePeriod) {
        return true;
    } else {
        return false;
    }
}
```

else if is redundant, because if _currentRelativeDay != 0, then _currentRelativeDay > 0 will always be true because the data type is uint256.

Recommendation: Could be simplified like this:

```
if (_currentRelativeDay >= _freezePeriod) {
    return true;
}
return false;
```

P.S. Also includes a fix from DFM-10.

O-DFM-3 «Unused function» | AspisLibrary

Status: Resolved

Description: The minimun function is not used.

LOC:78-84

```
function minimum(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a > b) {
        return b;
    } else {
        return a;
    }
}
```

Recommendation: Function may be removed.

O-DFM-4 «Redundant use of SafeMath» | AspisLibrary

Status: Resolved

Description: Since version 0.8.0, the definition of overflow and underflow of variables is built into the Solidity compiler and the use of the SafeMath library does not make sense, but only takes up the contract bytecode. You are using version 0.8.10.

Recommendation: You can replace using the SafeMath library with regular arithmetic operations.

O-DFM-5 «Type not explicitly specified» | AspisLiquidityCalculator

Status: Resolved

Description: The type is not explicitly specified for the variable aspisGuardian.

LOC:22

```
address immutable aspisGuardian;
```

Recommendation: We recommend explicitly specifying visibility types to avoid errors when interacting with contracts.

O-DFM-6 «Unused variable» | AspisLiquidityCalculator

Status: Resolved

Description: The local variable decimals is not used.

LOC:71

```
int256 decimals = int256(10 ** uint256(_decimals));
```

Recommendation: The variable can be removed.

O-DFM-7 «Local variable optimization» | ACL

Status: Resolved

Description: Instead of reusing freezeHash(_where, _role) , the permission variable can be used.

LOC:198-200

```
bytes32 permission = freezeHash(_where, _role);
if(freezePermissions[permission]) revert ACLData.ACLRoleFrozen({where: _where, 
role: _role});
freezePermissions[freezeHash(_where, _role)] = true;
```

For example, like this:

```
bytes32 permission = freezeHash(_where, _role);
if(freezePermissions[permission]) revert ACLData.ACLRoleFrozen({where: _where, 
role: _role});
freezePermissions[permission] = true;
```

Recommendation: Try to use already existing local variables.

O-DFM-8 «Lots of duplicate functionality» | AspisConfiguration | AspisRegistry

Status: Resolved/ Not Actual

Description: Most "add" and "remove" functions use the same logic for the same data types.

Recommendation: We recommend using a single function for solving the same tasks. This will allow you to follow the best development practices and make fewer mistakes when writing or editing functionality.

Since all "add" and "remove" functions operate on the same address data type, the functions can be modified like this:

```
function removeFromTradingTokens(address[] memory _tradingTokens) external {
    _aspisAuth();
    _removeFromAddressArray(tradingTokensArray, _tradingTokens);
}

function _removeFromAddressArray(
    address[] storage _storageAddressArray,
    address[] memory _memoryAddressArray
) private {
    uint256 l0 = _memoryAddressArray.length;
    uint256 l1 = _addressArray.length;
    require(l1 >= l0, "Array length error");

    for (uint256 i; i < l0; ) {
        for (uint256 j; j < l1; ) {
            if (_addressArray[j] == _memoryAddressArray[i]) {
                _addressArray[j] = _addressArray[l1 - 1];
                _addressArray.pop();
                unchecked { --l1; }
                break;
            }
            unchecked { ++j; }
        }
        unchecked { ++i; }
    }
}
```

P.S. Also includes optimizations from O-DFM-25.

O-DFM-9 «Excessive use of iterating over arrays» | AspisConfiguration | AspisRegistry

Status: Resolved

Description: All "remove" functions and some "check" functions use array iteration, which is a gas-inefficient solution for a large number of iterations and may even cause an "out of gas" error.

Recommendation: We recommend using additional mapping to index variables in an array, which makes the asymptotic complexity of a single "remove" or "check" operation equal to O(1).

For example, like this:

```
address[] public addressesArray;
mapping(address => uint256) private addressesIdx;

function addAddress(address toAdd) puplic {
    addressesIdx[toAdd] = addressesArray.length;
    addressesArray.push(toAdd);
}

function removeAddress(address toRemove) puplic {
    uint256 idx = addressesIdx[toRemove];
    uint256 lastElement = addressesArray[addressesArray.length - 1];
    addressesArray[idx] = lastElement;
    addressesIdx[lastElement] = idx;
    addressesArray.pop();
}
```

To check if an element is in an array, you can add another mapping, like this:

```
address[] public addressesArray;
mapping(address => uint256) private addressesIdx;
mapping(address => bool) public isAddressExistent;

function addAddress(address toAdd) puplic {
    require(!isAddressExistent[toAdd], "Already existent");
    isAddressExistent[toAdd] = true;

    addressesIdx[toAdd] = addressesArray.length;
    addressesArray.push(toAdd);
}

function removeAddress(address toRemove) puplic {
    require(isAddressExistent[toRemove], "Non-existent");
    isAddressExistent[toRemove] = false;

    uint256 idx = addressesIdx[toRemove];
    uint256 lastElement = addressesArray[addressesArray.length - 1];
    addressesArray[idx] = lastElement;
    addressesIdx[lastElement] = idx;
    addressesArray.pop();
}
```

If you don't want to use so many structures in contract storage, you can use the element's index to check if it's in the array, but in that case you need to reserve the first element in the array, like this:

```
address[] public addressesArray;
mapping(address => uint256) private addressesIdx;
```

```
constructor() {
    addressesArray.push(address(0));
}

function addAddress(address toAdd) puplic {
    require(addressesIdx[toAdd] == 0, "Already existent");
    require(toAdd != address(0), "Invalid address");

    addressesIdx[toAdd] = addressesArray.length;
    addressesArray.push(toAdd);
}

function removeAddress(address toRemove) puplic {
    require(addressesIdx[toRemove] != 0, "Non-existent");
    require(toRemove != address(0), "Invalid address");

    uint256 idx = addressesIdx[toRemove];
    uint256 lastElement = addressesArray[addressesArray.length - 1];
    addressesArray[idx] = lastElement;
    addressesIdx[lastElement] = idx;
    addressesIdx[toRemove] = 0;
    addressesArray.pop();
}

function isAddressExistent(address addr) public view returns (bool) {
    return addressesIdx[addr] != 0;
}
```

In this case, you need to ignore the first element of the array in further interactions with it.

The proposed approaches will help not only save gas with a large number of iterations, but also help to avoid duplicates.

O-DFM-10 «Incorrect function type specified» | AspisConfiguration

Status: Resolved

Description: The _aspisAuth function does not change the state of the contract, but is of type "Write Contract".

LOC:345-347

```
function _aspisAuth() internal {

    require(msg.sender == pool, "Unauthorized access");

}
```

Recommendation: Use the view modifier.

O-DFM-11 «Implicit return» | AspisConfiguration | AspisPool

Status: Resolved

Description: Functions AspisConfiguration::isDepositToken, AspisConfiguration::isTradingToken, AspisConfiguration::isTrustedProtocol, and AspisPool::isRageQuitFeeRequired expect a bool variable to be returned, but there is no explicit return.

Recommendation: This does not affect the logic of the function, since false is returned by default, but we recommend using an explicit return to follow a consistent development pattern and best practices.

O-DFM-12 «Unused storage variables» | AspisConfiguration

Status: Acknowledged

Description: The maxCap and spendingLimit variables are set but not used in contract functionality.

Recommendation: We recommend checking the need for these variables and modifying the logic of their use or deleting them.

O-DFM-13 «Redundant condition» | AspisConfiguration

Status: Resolved

Description: A redundant condition is used when setting the canChangeManager variable.

L:44

```
canChangeManager = _poolconfig[10] > 0 ? true : false;
```

Recommendation: Expression ? true : false is redundant and can be changed like this:

```
canChangeManager = _poolconfig[10] > 0;
```

O-DFM-14 «Redundant use of type int256» | AspisConfiguration | AspisPool

Status: Resolved

Description: The AspisConfiguration::setConfiguration function accepts an array of 15 int256 variables, 13 of which are positive and are converted to uint256 or bool. This solution is not optimal.

Recommendation: This approach is used in AspisPool::validateDepositLimit, which can be changed as follows:

```
if(_minDepositLimit > 0) {
    require(_depositValue / (10**SUPPORTED_USD_DECIMALS) >= _minDepositLimit,
"Minimum Deposit Error");
}

if(_maxDepositLimit != type(uint256).max) {
    require(_currentDepositValue / (10**SUPPORTED_USD_DECIMALS) <=
_maxDepositLimit, "Maximum Deposit Limit Error");
}
```

Or even like this, provided that _maxDepositLimit is always set:

```
require(_depositValue / (10**SUPPORTED_USD_DECIMALS) >= _minDepositLimit,
"Minimum Deposit Error");

require(_currentDepositValue / (10**SUPPORTED_USD_DECIMALS) <= _maxDepositLimit,
"Maximum Deposit Limit Error");
```

We recommend following design best practices and using 0 as the minimum value (or no value) and type(uint256).max as the maximum value (or no value). This will not only make the protocol work more optimized, but also make the protocol logic simpler.

O-DFM-15 «Typo in error message» | AspisPool

Status: Resolved

Description: The error message in revert contains a typo.

L:135

```
revert("Fundraing over or not started yet");
```

Recommendation: Change to «Fundraising over or not started yet".

O-DFM-16 «Redundant use of the storage variable» | AspisPool

Status: Resolved

Description: The _tempETHBalance variable is stored in the contract storage, but is used only in the deposit function. The variable can be replaced with a local one and passed as an argument to the getCurrentTokenPrice function.

Recommendation: Can be changed like this:

```
function deposit() {
    // ...
    (uint256 _price, ) = getCurrentTokenPrice(
        _token == ETH && msg.value != 0 ? _amount : 0
    );
    // ...
}

function withdraw() {
    // ...
    (uint256 _price, ) = getCurrentTokenPrice(0);
    // ...
}

function getCurrentTokenPrice(
    uint256 _tempETHBalance
) internal returns (uint256 _price, uint256 _poolValue) {
// ...
}
```

O-DFM-17 «Function does not match the name» | AspisPool

Status: Acknowledged

Description: The directAssetTransfer function performs a call, which is not what its name suggests.

Recommendation: Please make sure the function contains the correct logic and change it if necessary or change the name.

O-DFM-18 «Expression Simplification» | AspisPool

Status: Resolved

Description: The expression in function validateProposal can be simplified.

L:389-399

```
if (selector == PROPOSAL_UPDATE_MANAGER && configuration.canChangeManager() ==
true) {
    return true;
} else if (selector == PROPOSAL_REMOVE_PROTOCOLS) {
     return true;
} else {
    if (_creator == manager) {
        return true;
    }
}

return false;
```

Recommendation: Can be changed like this:

```
return (

    (selector == PROPOSAL_UPDATE_MANAGER && configuration.canChangeManager())

    || selector == PROPOSAL_REMOVE_PROTOCOLS

    || _creator == manager

)
```

O-DFM-19 «Redundant use of delete»| AspisConfiguration | AspisRegistry

Status: Resolved

Description: The AspisConfiguration::removeDepositTokens, AspisConfiguration::removeFromWhitelist, AspisConfiguration::removeFromTradingTokens, AspisConfiguration::removeFromTrustedProtocols, AspisRegistry::removeSupportedTradingProtocols and AspisRegistry::removeSupportedTradingTokens functions use delete before setting the variable to another value, but often this does not help save gas, but makes the execution of the function more expensive in terms of gas.

This happens because delete sets the value of the variable to zero, and the operation of changing the zero variable is more expensive in gas.

Recommendation: We recommend removing delete in these functions.

O-DFM-20 «Additional checks on deletion»| AspisConfiguration | AspisRegistry

Status: Resolved

Description: The functions AspisConfiguration::removeFromWhitelist, AspisConfiguration::removeFromTradingTokens, AspisRegistry::removeSupportedTradingProtocols and AspisRegistry::removeSupportedTradingTokens use iteration over all elements of the input array, but there are mappings that store true if the object is in the array.

Recommendation: To avoid unnecessary iterations of the loop, you can add an additional check, for example:

```
function removeSupportedTradingTokens(
    address[] memory _tokens
) external isAspisGuardian {
    // ...
    for (uint8 i = 0; i < _tokens.length; i++) {
        address _token = _tokens[i];
        if (aspisSupportedTradingTokens[_token]) {
            for(uint8 j=0; j < aspisSupportedTradingTokensArray.length; j++) {
                // ...
            }
        }
    }
}
```

O-DFM-21 «Reverse iteration optimization»| AspisPool

Status: Resolved

Description: The withdraw function uses reverse iterations over a loop using int256 and conversions to uint256, which is inefficient:

L:206-216

```
for (int256 i = int256(_deposits.length) - 1; i >= 0; i--) {
    uint256 _depositPrice = _deposits[uint256(i)].price;
    uint256 _depositAmount = _deposits[uint256(i)].amount;

    delete depositsOfUser[msg.sender][uint256(i)];

    _weightedAveragePrice += (_depositAmount * _depositPrice)/_amount;
}
```

Recommendation: We recommend using the simplified version like this:

```
uint256 i = _deposits.length;
for (i; i > 0; ) {
    unchecked { --i; }
    _weightedAveragePrice += (_deposits[i].amount * _deposits[i].price) /
_amount;
    depositsOfUser[msg.sender].pop();
}
```

P.S. Also includes optimization from O-DFM-25 and fix from DFM-18.

O-DFM-22 «Checking the deposit amount»| AspisPool

Status: Resolved

Description: Deposit tokens can be dynamically added, so be careful when checking tokens. Tokens may contain internal fees or use an upgradeable pattern and change functionality upon upgrade, resulting in the number of tokens received may not be as expected.

Recommendation: We recommend adding a check in deposit function for the number of received tokens like this:

```
IERC20 depositToken = IERC20(_token);

uint256 balBefore = depositToken.balanceOf(address(this));
depositToken.safeTransferFrom(msg.sender, address(this), _amount);
uint256 balAfter = depositToken.balanceOf(address(this));

require(balAfter - balBefore >= _amount, "Error");
```

O-DFM-23 «Using calldata instead of memory» | *GLOBAL

Status: Resolved

Many functions accept memory arrays or memory bytes, but you can use calldata where possible to save gas.

Also, if you use calldata instead of memory in the AspisPool::validateProposal function, you can get the selector using a bytes slice, like this:

```
bytes4 selector = bytes4(_proposal[:4]);
```

O-DFM-24 «Visibility modifiers for functions and variables» | *GLOBAL

Status: Partially Resolved

Description: Some functions and variables can be changed from public to external or from internal to private.

Recommendation: We recommend using limit visibility modifiers where possible to make the contract bytecode smaller and deploy gas cheaper.

If utility functions are not used in inheritance, then they can be changed to private.

If executable or view functions are not used inside the contract, then they can be changed to external.

For example, the AspisPool::directAssetTransfer function can be changed to external because it cannot be called inside the contract without using call due to the condition require(msg.sender == address(this), "Unauthorized call").

O-DFM-25 «Loops optimizations» | *GLOBAL

Status: Partially Resolved

Contract uses a large number of loops that can be greatly optimized for the gas to be used.

First, it's better to use unit256 instead of uint8 as i. It makes no sense to use uint8, since this variable is local and is not written to storage slots. In addition, the slot size in Solidity is 32 bytes, which means that using uint256 does not require any additional conversions.

Second, it's better to declare the constraint as a separate variable instead of using the .length method, which avoids having to get the length each time.

Thirdly, using unchecked for increment will save gas by ignoring built-in SafeMath checks.

We want to demonstrate the effectiveness of optimization with a small example. All function calls were independent and carried out on new contracts.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.11;

contract GasTest {

    uint256 private variable;
    uint256[] private arr;

    constructor() {
        arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    }

    // 83136 gas
    // function test() external {
    //     for (uint8 i; i < arr.length; i++) {
    //         variable = arr[i];
    //     }
    // }

    // 82922 gas
    // function test() external {
    //     for (uint256 i; i < arr.length; i++) {
    //         variable = arr[i];
    //     }
    // }

    // 81695 gas
    // function test() external {
    //     uint256 l = arr.length;
    //     for (uint256 i; i < l; i++) {
    //         variable = arr[i];
    //     }
    // }

    // 81485 gas
    // function test() external {
    //     for (uint256 i; i < arr.length; ) {
    //         variable = arr[i];
    //         unchecked { ++i; }
    //     }
    // }

    // 80258 gas
    // function test() external {
    //     uint256 l = arr.length;
    //     for (uint256 i; i < l; ) {
    //         variable = arr[i];
    //         unchecked { ++i; }
    //     }
    // }
}
```

This approach may slightly increase the cost of deploying the contract, but it will save a lot of gas when using functions, especially with a large number of iterations.

Automated Analyses

Slither

Slither's automatic analysis not found vulnerabilities, or these false positives results .

Methodology

Manual Code Review

We prefer to work with a transparent process and make our reviews a collaborative effort. The goal of our security audits is to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, review open issue tickets, and investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system to make a final decision.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Appendix A — Finding Statuses

Resolved	Contracts were modified to permanently resolve the finding
Mitigated	The finding was resolved by other methods such as revoking contract ownership or updating the code to minimize the effect of the finding
Protocol's Design	Assumed by the protocol design as a necessary functionality that will work properly within this application
Acknowledged	Project team is made aware of the finding
Open	The finding was not addressed
Not Actual	Not relevant after protocol logic changes