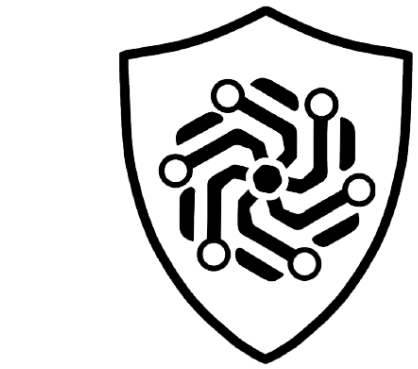


Smart Contract Audit Report

October, 2023

DEFIMOON PROJECT

Audit and
Development

CONTACTS

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

DEFIMOON

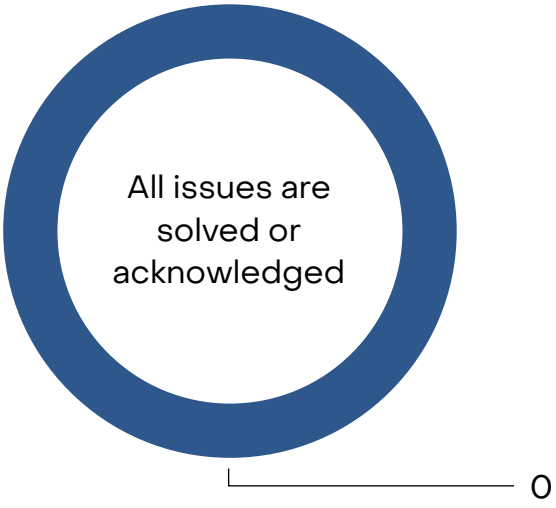
be secure

19 October 2023

This audit report was prepared by DefiMoon for XDCS.

Audit information

Description	Escrow-based staking protocol
Timeline	17 September 2023 - 19 October 2023
Approved by	Artur Makhnach, Kirill Minyaev
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Manual Review
Source code	https://github.com/yodaplus/xdcs-masternode-staking/tree/991e235eb9bea6f6f51b139145d61cca62a1ec30
Reaudit Source code	https://github.com/yodaplus/xdcs-masternode-staking/tree/e5819f5ae64183354c55b154ae679d7e8a30a630
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

Audited Files:

/nft-registry/
    NFT.sol
    NFTRegistry.sol
/CustodianContract.sol
/EscrowTypes.sol
/PoolContract.sol
/PoolController.sol
/ReasonCodes.sol
/TimeOracle.sol
/TokenBase.sol
/TokenCreatorTvT.sol
/TokenTvT.sol
/TokenTvTTypes.sol
/WhitelistManager.sol
/WXDCOZ.sol

Major vulnerabilities have been found.

High-risk vulnerabilities and minor issues were found, and recommendations were made to optimize and improve Solidity development practices.

We recommend that you read all the findings and prepare fixes or refute the findings. Some findings may be false positive - this is due to the presence of a small number of comments in the code and the lack of detailed documentation.

One of the main problems is the presence of a large number of iterations of loops, which can cost a lot of gas or run into the gas block limit. We recommend using algorithms with O(1) complexity instead of using loops wherever possible, and trying to use gas-safety solutions in other places.

Summary of findings

ID	Description	Severity	Status
DFM-1	Possible to change the nodeToken in the pool	High Risk	Resolved
DFM-2	Tokens mint to the wrong address	High Risk	Resolved
DFM-3	Removing from the array does not change the value in the mapping	High Risk	Resolved
DFM-4	Values are not updated in real time	High Risk	Partially Resolved
DFM-5	Mature balance may not be accurate	High Risk	Resolved
DFM-6	Accrual of rewards through brute force	High Risk	Acknowledged
DFM-7	Tokens are always burned by the owner, but can be sent to anyone	High Risk	Resolved
DFM-8	When transferring ownership, no tokens are	Medium Risk	Resolved
DFM-9	Incorrect condition for maturity balance	Medium Risk	Resolved
DFM-10	Finding an element through brute force	Medium Risk	Resolved
DFM-11	Inefficient array formation	Medium Risk	Resolved
DFM-12	Phantom fallback	Medium Risk	Resolved
DFM-13	Possibility of changing maturity period	Medium Risk	Acknowledged
DFM-14	Possible to perform swap issuance after a timeout	Medium Risk	Resolved
DFM-15	Potentially incorrect logic	Medium Risk	Resolved
DFM-16	Lack of checks during manual distribution of	Medium Risk	Resolved
DFM-17	Phantom maturity balance	Low Risk	Partially Resolved
DFM-18	Incorrect array iteration when deleting	Low Risk	Resolved
DFM-19	Incorrect sequence of actions	Low Risk	Resolved
DFM-20	Empty elements are not removed from the array	Low Risk	Resolved
DFM-21	Possibly incorrect usage of variables	Low Risk	Resolved
DFM-22	Timestamp is used incorrectly	Low Risk	Resolved
DFM-23	Incorrectly removing elements from an array	Low Risk	Resolved

ID	Description	Severity	Status
DFM-24	Lack of fees check	Low Risk	Resolved
DFM-25	Check fees when setting	Low Risk	Partially Resolved
DFM-26	Lack of duplicates check	Low Risk	Resolved
DFM-27	Multiply before division to improve calculation	Low Risk	Resolved
DFM-28	Using a multiplier to calculate rewards	Low Risk	Resolved
DFM-29	Calculate the commission first	Low Risk	Resolved
DFM-30	More secure interaction with tokens and ETH	Low Risk	Resolved
DFM-31	Potential loss of owner	Low Risk	Resolved
DFM-32	Disabling initializing	Informational	Resolved
DFM-33	Extra check	Informational	Resolved
DFM-34	Changing base uri	Informational	Acknowledged
DFM-35	Visibility modifier not explicitly specified	Informational	Resolved
DFM-36	Using an existing modifier	Informational	Resolved
DFM-37	Unused function	Informational	Resolved
DFM-38	The error description does not match the condition	Informational	Resolved
DFM-39	Duplicate function	Informational	Resolved
DFM-40	Unused validator address	Informational	Resolved
DFM-41	Typo	Informational	Resolved
DFM-42	Use readable errors	Informational	Resolved
DFM-43	Redundant use of SafeMath	Informational	Resolved
DFM-44	Field indexing in events	Informational	Resolved
DFM-45	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	Not 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	Not Passed
High consumption ‘storage’ storage	Passed
“Out of Gas” Attack	Passed

Findings

DFM-1 «Possible to change the nodeToken in the pool contract» | CustodianContract

Severity: High Risk

Status: Resolved

Description: The CustodianContract::publishToken function deploys a new TokenTvT, which is set to PoolContract::TOKEN_TVT_ADDRESS, without any check that PoolContract::TOKEN_TVT_ADDRESS is not already set.

PoolContract::TOKEN_TVT_ADDRESS can also be changed by PoolContract::owner.

Changing PoolContract::TOKEN_TVT_ADDRESS will cause serious disruption to the protocol.

Recommendation: We recommend adding a check that PoolContract::TOKEN_TVT_ADDRESS is not already set and disable it changing.

DFM-2 «Tokens mint to the wrong address» | CustodianContract

Severity: High Risk

Status: Resolved

Description: In the CustodianContract::publishToken function, when deploying a TokenTvT contract, CustodianContract::owner is specified as TokenTvTInput.owner, but the function can be called not only by CustodianContract::owner.

As a result, when the TokenTvT contract is deployed, tokens are minted to the CustodianContract::owner address, but then TokenTvT::owner is set as msg.sender, which has no tokens on its balance.

Recommendation: We recommend setting msg.sender as TokenTvTInput.owner and consider the recommendation from DFM-8.

DFM-3 «Removing from the array does not change the value in the mapping» | TokenTvT

Severity: High Risk

Status: Resolved

Description: In the TokenTvT::refreshInvestorList function, when removing a user from the array, _isTokenHolder[_tokenHolders[i]] is not set to false. As a result, the user will no longer be re-added to the _tokenHolders array in the TokenTvT::onIssue function, which means his balance will not be taken into account when calculating TokenTvT::amountStakedNft and TokenTvT::amountStakedNonNft.

Recommendation: We recommend setting _isTokenHolder[_tokenHolders[i]] to false when removing a user from the _tokenHolders array. The fix is given in DFM-18.

DFM-4 «Values are not updated in real time» |

TokenTvT
NFTRegistry
NFT

Severity: High Risk

Status: Partially Resolved

Comment: It is still relevant when transferring NFTs, but this is the mechanics of the protocol - a slice of the array is made at the time the list of investors is updated and does not take into account subsequent changes.

Description: The variables _tokenHolders, _isTokenHolder, _isNFTHolder, amountStakedNft and amountStakedNonNft are not updated in real time, although TokenTvT and NFT can be transferred using the transfer and transferFrom functions.

For example, in this way you can use only one NFT token to mark an unlimited number of addresses as _isTokenHolder = true (call onIssue, send the token to another address, and so on). Or if NFTRegistry::removeNFT or NFTRegistry::addNFT is called, these variables will not be affected until TokenTvT::refreshInvestorList is called.

Recommendation: We recommend using hooks to update these variables in real time - this will not only solve this vulnerability, but also eliminate the use of a large number of loops, which are suboptimal and can lead to an "out of gas" error with a large number of iterations.

For the TokenTvT contract, you can use _afterTokenTransfer or even allow the transfer and transferFrom functions to be called only by the EscrowManager contract.

For NFT contracts, it is more difficult to use _afterTokenTransfer because there can be multiple NFT contracts and TokenTvT contracts that are not directly related to each other, so we recommend reconsidering the current approach and, for example, using a separate NFT contract for each separate TokenTvT contract.

DFM-5 «Mature balance may not be accurate» | TokenTvT

Severity: High Risk

Status: Resolved

Description: The TokenTvT::matureBalanceOf function can return a non-zero balance, as a result of which the EscrowManager::swapRedemption function can be called, but when calling TokenTvT::onRedeem the user's mature balance will not be reduced.

This is because in the TokenTvT::matureBalanceOf function there is a condition subscriptionPeriod < _custodianContract.getTimestamp() && isStaked == false, which is not in the TokenTvT::onRedeem function.

Recommendation: The logic in the TokenTvT::onRedeem function must match the logic in the TokenTvT::matureBalanceOf function. You also need to add a check when setting subscriptionPeriod and maturityPeriod so that subscriptionPeriod is always greater than maturityPeriod.

Additionally, we recommend adding a check require(remainingValue == 0, "Insufficient mature balance") to the TokenTvT::onRedeem function.

DFM-6 «Accrual of rewards through brute force» | PoolContract

Severity: High Risk

Status: Acknowledged

Comment: Is a protocol mechanic.

Description: The distributeRewards function updates rewards for all users by iterating through the tokenHoldersNFT and tokenHoldersNonNFT arrays. As the number of elements in the arrays increases, brute force will waste more gas, leading to an “out of gas” error.

Recommendation: We recommend using the O(1) algorithm to assign rewards to users.
For example, you can use PancakeMasterChief (using accumulated rewards per share) as a reference.

DFM-7 «Tokens are always burned by the owner, but can be sent to anyone» |

TokenTvT
EscrowManager

Severity: High Risk

Status: Resolved

Description: Any address can be passed to the TokenTvT::redeem function as tradeTokenDestination, but payment tokens will still be sent to the user from the issuerSettlementAddress address, and trade tokens will be burned from the owner address (in the EscrowManager::swapRedemption function ITokenHooks(escrowOrder.tradeToken).burnTokens is called).

Thus, the user can both save trade tokens and receive payment tokens.

Recommendation: We recommend making the extended TokenTvT::redeem function private or internal.

DFM-8 «When transferring ownership, no tokens are transferred» |

TokenTvT
TokenBase

Severity: Medium Risk

Status: Resolved

Description: The TokenTvT contract is designed in such a way that tokens available for swap are stored on the owner's balance, but when ownership is transferred the tokens are not transferred to the new owner.

As a result, swaps will not be available until the new owner mints the tokens for himself, and the old owner will have tokens left that he can use.

Recommendation: We recommend transferring tokens along with the transfer of ownership.

In addition, the logic of storing tokens on the owner's balance is inconvenient. We recommend storing tokens on a separate contract or on the balance of the TokenTvT contract itself - this will help make the code simpler and more predictable.

DFM-9 «Incorrect condition for maturity balance» | TokenTvT

Severity: Medium Risk

Status: Resolved

Comment: Not actual.

Description: The matureBalanceOfPending, matureBalanceOf and onRedeem functions use a condition to check maturity maturityPeriod < _custodianContract.getTimestamp(). This condition is repeated for each iteration of the loop, but neither maturityPeriod nor _custodianContract.getTimestamp() can be changed while the loop is running.

Recommendation: We assume that maturityPeriod should not be a timestamp, but a period of time (based on the name and logic of the code). In this case, the onRedeem function should look like this (also includes DFM-20 fix):

```
mapping(address => uint256) private maturityBucketStartIndex;

// ...

uint256 remainingValue = value;
uint256[] storage maturityBuckets =
    _issuedTokensMaturityBuckets[subscriber];

uint256 currentTimestamp = _custodianContract.getTimestamp();
uint256 l = maturityBuckets.length;
uint256 i = maturityBucketStartIndex[subscriber]

while (i < l && remainingValue > 0) {
    uint256 issueTmestamp = maturityBuckets[i];
    if (currentTimestamp - issueTmestamp <= maturityPeriod) { break; }

    uint256 currentBucketBalance =
        _issuedTokensByMaturityBucket[subscriber][issueTmestamp];

    if (currentBucketBalance > remainingValue) {
        _issuedTokensByMaturityBucket[subscriber][issueTmestamp] =
            currentBucketBalance - remainingValue;
        delete remainingValue;
    } else {
        delete _issuedTokensByMaturityBucket[subscriber][issueTmestamp];
        unchecked { maturityBucketStartIndex[subscriber] = i + 1; }
        remainingValue -= currentBucketBalance;
    }

    unchecked { ++i; }
}
```

Otherwise, if you need to use the maturityPeriod < _custodianContract.getTimestamp() condition, you should avoid using an array to store the user's balance.

DFM-10 «Finding an element through brute force» | TokenTvT

Severity: Medium Risk

Status: Resolved

Description: The onRedeem function uses brute force to find the subscriber address in the _tokenHolders array, which is not an optimal approach. As the number of elements in the _tokenHolders array increases, brute force will waste more gas, leading to an “out of gas” error.

Recommendation: We recommend using O(1) algorithms where possible.
You can use mapping to store element indexes like this:

```
mapping(address => uint256) private tokenHolderIdx;

// ...
// on add

if (!_isTokenHolder[subscriber]) {
    tokenHolderIdx[subscriber] = _tokenHolders.length;
    _tokenHolders.push(subscriber);
    _isTokenHolder[subscriber] = true;
}

// ...
// on remove

if (balanceOf(subscriber) == 0) {
    uint256 lastIdx = _tokenHolders.length - 1;
    _tokenHolders[tokenHolderIdx[subscriber]] = _tokenHolders[lastIdx];
    tokenHolderIdx[lastIdx] = tokenHolderIdx[subscriber];
    _tokenHolders.pop();
    _isTokenHolder[subscriber] = false;
}
```

DFM-11 «Inefficient array formation» | TokenTvT

Severity: Medium Risk

Status: Resolved

Description: The getNFTHolders and getNonNFTHolders functions use iterate over the _tokenHolders array to form new arrays, which is not an optimal approach. As the number of elements in the _tokenHolders array increases, brute force will waste more gas, leading to an “out of gas” error.

Additionally, this approach will cause the array to contain empty elements (address(0)).

Recommendation: To get rid of empty elements you can reduce the length of the array to the actual length like this:

```
address[] memory nftHolders = new address[](_tokenHolders.length);
uint256 j;
uint256 l = _tokenHolders.length;
for (uint256 i = 0; i < l; ) {
    address holderAddress = _tokenHolders[i];
    if (_isNFTHolder[holderAddress]) {
        nftHolders[j] = holderAddress;
        unchecked { ++j; }
    }
    unchecked { ++i; }
}
assembly {
    mstore(nftHolders, j)
}
return nftHolders;
```

But we recommend using two separate arrays for nft holders and non-nft holders to reduce the use of loops.

DFM-12 «Phantom fallback» | PoolContract

Severity: Medium Risk

Status: Resolved

Description: The fallback function is declared, but does not contain any logic. This means that calling any functions (using any signatures) will not raise an error, even if the function does not actually exist in the contract. It is critical that a contract always returns an error in the case of a non-existent behavior scenario, otherwise the state of one contract may change as expected while the state of another contract does not, breaking the contracts.

Recommendation: Avoid using fallback as it is not used and receive is already used to receive ETH.

DFM-13 «Possibility of changing maturity period» | TokenTvT

Severity: Medium Risk

Status: Acknowledged

Comment: Is a protocol mechanic. It was prohibited to set maturityPeriod less than subscriptionPeriod.

Description: The TokenTvT contract contains the setMaturityPeriod function, which changes the value of the maturityPeriod variable. Using the setMaturityPeriod function, owner can increase maturityPeriod, as a result of which maturityPeriod can become greater than subscriptionPeriod (see DFM-5) or delay the time users access the EscrowManager::swapRedemption function.

Recommendation: We recommend allowing only downgrade maturityPeriod.

DFM-14 «Possible to perform swap issuance after a timeout» | EscrowManager

Severity: Medium Risk

Status: Resolved

Comment: Not actual.

Description: The cancelIssuance function can only be called when itmeout has passed, but the swapIssuance function can also be called even if itmeout has passed.

Recommendation: We recommend that you disable calling swapIssuance when the timeout has passed.

DFM-15 «Potentially incorrect logic» | EscrowManager

Severity: Medium Risk

Status: Resolved

Description: In the EscrowManager::swapRedemption function, the escrowConditionsFlag and timeoutFlag checks are duplicated three times.

In addition, the timeoutFlag check does not play any role - it does not affect the result; for the function to be executed, escrowConditionsFlag must be true.

Also, tokens are always burned, regardless of which of the conditions has been met.
In addition, tokens are always burned from the TokenTvT::owner address (see DFM-7).

Recommendation: We recommend checking that the logic in the EscrowManager::swapRedemption function is correct, paying close attention to the conditions, and making sure that the ITokenHooks(escrowOrder.tradeToken).burnTokens call is necessary.

DFM-16 «Lack of checks during manual distribution of rewards» | PoolContract

Severity: Medium Risk

Status: Resolved

Description: The distributeRewardsManual function does not check that there are enough free funds on the contract balance for distribution. As a result, users may be credited with rewards that are not on the contract balance.

Recommendation: We recommend using msg.value instead of the amount argument.

DFM-17 «Phantom maturity balance» |

TokenTvT
TokenBase

Severity: Low Risk

Status: Partially Resolved

Comment: Still relevant when sending tokens to the owner's address.

Description: The user can call TokenTvT::burn (from ERC20Burnable), TokenTvT::burnFrom (from ERC20Burnable) or transfer to burn or transfer (only to the owner) his tokens, but his mature balance will not change.

Recommendation: We recommend not leaving phantom entries and changing them when necessary.

In addition, we did not see the need to use ERC20Burnable, we recommend against it.

DFM-18 «Incorrect array iteration when deleting» | TokenTvT

Severity: Low Risk

Status: Resolved

Description: The TokenTvT::refreshInvestorList function incorrectly implements a loop when removing elements. Removal is implemented by replacing the element and pop(), as a result of which the new element will take the place of the current one (will have the same index) and will not be checked.

Recommendation: When an element is removed, there is no need to increment i (also includes the DFM-3 and DFM-19 fixes):

```
uint256 l = _tokenHolders.length;
for (uint256 i; i < l; ) {
    address tokenHolder = _tokenHolders[i];
    uint256 holderBalance = balanceOf(tokenHolder);

    if (nftRegistry.walletHoldsToken(tokenHolder)) {
        _isNFTHolder[tokenHolder] = true;
        amountStaked += holderBalance;
    } else {
        _isNFTHolder[tokenHolder] = false;
        amountNonStaked += holderBalance;
    }

    if (holderBalance == 0) {
        _tokenHolders[i] = _tokenHolders[l - 1];
        _tokenHolders.pop();
        _isTokenHolder[tokenHolder] = false;
        unchecked {
            --l;
        }
    } else {
        unchecked {
            ++i;
        }
    }
}
```

DFM-19 «Incorrect sequence of actions» | TokenTvT

Severity: Low Risk

Status: Resolved

Description: In the TokenTvT::refreshInvestorList function, if an address is removed from the TokenTvT::_tokenHolders array, then the nftRegistry::walletHoldsToken function will not be called for it, since another address will take its place.

Additionally, if the last element of the array is removed, an "out of bounds" error will be thrown when trying to get an element with a non-existent index.

Recommendation: We recommend that you remove an element from the array last action, as shown in DFM-18.

DFM-20 «Empty elements are not removed from the array» | TokenTvT

Severity: Low Risk

Status: Resolved

Description: In the TokenTvT::onRedeem function, when the bucket balance is equal to zero, the element is not removed from the _issuedTokensMaturityBuckets array, as a result of which empty elements will accumulate at the beginning of the array, and the length of the array will always increase - in this case, each time more iterations will be required and will be spent more gas until the "out of gas" error occurs.

Recommendation: We recommend avoiding unnecessary iterations and removing empty elements from the array. We see that your array is sorted by ascending timestamp, which means you can't use replace and pop(), so you can store the index of the first non-empty element, like this:

```
mapping(address => uint256) private maturityBucketStartIndex;

// ...

uint256 remainingValue = value;
uint256[] storage maturityBuckets =
    _issuedTokensMaturityBuckets[subscriber];

uint256 l = maturityBuckets.length;
uint256 i = maturityBucketStartIndex[subscriber]

while (
    i < l
    && remainingValue > 0
    && maturityPeriod < _custodianContract.getTimestamp()
) {

    uint256 maturityTimestamp = maturityBuckets[i];
    uint256 currentBucketBalance =
        _issuedTokensByMaturityBucket[subscriber][maturityTimestamp];

    if (currentBucketBalance > remainingValue) {
        _issuedTokensByMaturityBucket[subscriber][maturityTimestamp] =
            currentBucketBalance - remainingValue;
        delete remainingValue;
    } else {
        delete _issuedTokensByMaturityBucket[subscriber][maturityTimestamp];
        unchecked { maturityBucketStartIndex[subscriber] = i + 1; }
        remainingValue -= currentBucketBalance;
    }

    unchecked { ++i; }
}
```

DFM-21 «Possibly incorrect usage of variables» | TokenTvT

Severity: Low Risk

Status: Resolved

Description: The maturityPeriod and subscriptionPeriod variables imply the use of periods, but are compared to a timestamp.

We assume that either the variables are named incorrectly or are being used incorrectly (like DFM-9).

Recommendation: Please check the name and usage of the variables and correct if necessary.

DFM-22 «Timestamp is used incorrectly» | TokenTvT

Severity: Low Risk

Status: Resolved

Description: The redeem function uses block.timestamp instead of _custodianContract.getTimestamp().

Recommendation: We recommend using _custodianContract.getTimestamp(), following the same pattern.

DFM-23 «Incorrectly removing elements from an array» | CustodianContract

Severity: Low Risk

Status: Resolved

Description: The _removeRoleAddresses function uses delete to remove an element, but it does not remove the element from the array, it clears it.

Recommendation: Use replace and pop() like this:

```
uint256 il = addresses.length;
for (uint256 i; i < il; ) {
    uint256 jl = userAddresses.length;
    for (uint256 j; j < jl; ) {
        address userAddress = userAddresses[j];
        if (userAddress == addresses[i]) {
            delete _addressToUserPrimaryAddress[userAddress];
            userAddresses[j] = userAddresses[jl - 1];
            userAddresses.pop();
            break;
        }
        unchecked { ++j; }
    }
    unchecked { ++i; }
}
```

DFM-24 «Lack of fees check» | PoolContract

Severity: Low Risk

Status: Resolved

Description: The distributeRewards function from reserveFunds takes two fees: DEV_FEES.percentage and PRIME_NUMBER_FEES.percentage, but there is no guarantee that the sum of DEV_FEES and PRIME_NUMBER_FEES does not exceed 100.

Recommendation: We recommend adding a check when setting these fees to ensure that their amount does not exceed 100.

DFM-25 «Check fees when setting» | PoolContract

Severity: Low Risk

Status: Partially Resolved

Description: The values ADMIN_FEES_PERCENT, NFT_HOLDERS_HAIRCUT and NON_NFT_HOLDERS_CUT are not checked in any way during setting, and fees DEV_FEES.percentage and PRIME_NUMBER_FEES.percentage are checked only at the time of use, which is an inefficient gas approach and does not allow you to immediately know if the values were set incorrectly.

Recommendation: We recommend adding checks when setting these values.

DFM-26 «Lack of duplicates check» | NFTRegistry

Severity: Low Risk

Status: Resolved

Description: The addNFT function lacks a check for duplicates.

Recommendation: We recommend adding a duplicate check to avoid double additions.

DFM-27 «Multiply before division to improve calculation accuracy» | PoolContract

Severity: Low Risk

Status: Resolved

Description: Since Solidity only works with integers, when dividing before multiplying, the results may be rounded down heavily or even be equal to 0 if the numerator is less than the denominator.

Recommendation: We recommend changing the calculation of the totalNodeReward variable in the calculateRewards function as follows:

```
uint256 totalNodeReward = currentBalance * 1e20 / (amountStakedNFT +
amountStakedNonNFT);
```

DFM-28 «Using a multiplier to calculate rewards» | PoolContract

Severity: Low Risk

Status: Resolved

Description: In the calculateRewards function, the rewardNFT and rewardNonNFT variables are divided by 1e18 before being returned. Thus, rewardNFT and rewardNonNFT store reward share, but reward share can be less than 1 (when less than 1 ETH is awarded for 1 token) and in this case 0 will be returned as rewardNFT and rewardNonNFT.

Recommendation: We recommend not dividing rewardNonNFT and rewardNFT by 1e18 in the calculateRewards function, but dividing it in the distributeRewards function after multiplying to improve the accuracy of calculations like this:

```
uint256 reward = balance * rewardNFT / 1e18;
```

DFM-29 «Calculate the commission first» | PoolContract

Severity: Low Risk

Status: Resolved

Description: In the calculateRewards function, feesDeducted is calculated like this:

```
currentBalance = currentBalance.mul(100 - ADMIN_FEES_PERCENT).div(100);
feesDeducted = value - currentBalance;
```

As a result, fee is rounded up, not currentBalance.

Recommendation: We recommend calculating the fee first, and then changing the value of the balance variable like this:

```
feesDeducted = currentBalance * ADMIN_FEES_PERCENT / 100;
currentBalance -= feesDeducted;
```

Thus, for small value or a high ADMIN_FEES_PERCENT, the currentBalance will not be equal to zero.

DFM-30 «More secure interaction with tokens and ETH» | *

Severity: Low Risk

Status: Resolved

Recommendation: We recommend using SafeERC20 to interact with ERC20 contract functions and using Address.sendValue to send ETH.

DFM-31 «Potential loss of owner» | *

Severity: Low Risk

Status: Resolved

Description: Most of the contract inherit the Ownable contract from OpenZeppelin which includes the renounceOwnership function. This function resets the owner of the contract without the possibility of restoring it, which can lead to irreparable consequences if this function is called, since most of the functionality of contracts is available only to the owner.

Also, the Ownable::transferOwnership function is not safe either.

Recommendation: Most of the functions in your contracts require owner permissions, and as a result, loss of permissions can become critical. The best solution would be to stop using OpenZeppelin's renounceOwnership function. For example, like this:

```
function renounceOwnership() public override onlyOwner {
    revert("Renounce ownership disabled");
}
```

It's also best practice to use transfer the owner in two steps, like this.

DFM-32 «Disabling initializing» |

CustodianContract
EscrowManager

Severity: Information

Status: Resolved

Description: Since the upgradeable version of the contract is used, the initialize function is not called on the implementation contract and can be called by anyone. It is better to block the call of this function on the implementation contract.

Recommendation: We recommend disabling the initialize function, as recommended by OpenZeppelin: «Locks the contract, preventing any future reinitialization. This cannot be part of an initializer call. Calling this in the constructor of a contract will prevent that contract from being initialized or reinitialized to any version. It is recommended to use this to lock implementation contracts that are designed to be called through proxies.»

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}
```

DFM-33 «Extra check» | CustodianContract

Severity: Information

Status: Resolved

Description: The _removeRoleAddresses and _addRoleAddresses functions do not need to check primaryArgNotUser for ErrorCondition.WRONG_CALLER.

Recommendation: Remove unnecessary check like this:

```
if (senderNotOwner && senderNoPrimaryArgMatch) {
    throwError(ErrorCondition.WRONG_CALLER);
}

if (primaryArgNotUser) {
    throwError(ErrorCondition.USER_DOES_NOT_EXIST);
}
```

DFM-34 «Changing base uri» | NFT

Severity: Information

Status: Acknowledged

Comment: The current implementation is used for testing only and will be changed in the future.

Description: A hardcoded link to the server is used as the _baseURI, which is not the best practice.

Recommendation: Since the link may change in the future or the server may no longer be available, we recommend adding the ability to change the _baseURI as well as the ability to set a suffix, for example for ".json". Like this:

```
string private baseURI_;
string public uriSuffix;

function _baseURI() internal pure override returns (string memory) {
    return baseURI_;
}

function setURISuffix(
    string calldata _uriSuffix
) external onlyOwner {
    uriSuffix = _uriSuffix;
}

function setBaseURI(
    string calldata _baseURI_
) external onlyOwner {
    baseURI_ = _baseURI_;
}

function tokenURI(
    uint256 tokenId
) public view override returns (string memory) {
    return string(abi.encodePacked(super.tokenURI(tokenId), uriSuffix));
}
```

DFM-35 «Visibility modifier not explicitly specified» |

NFTRegistry
EscrowManager

Severity: Information

Status: Resolved

Description: The visibility modifier for the NFTRegistry::launchedNfts, EscrowManager::custodianContract variables is not explicitly specified.

Recommendation: We recommend explicitly specifying visibility modifiers to avoid potential compiler and development issues.

DFM-36 «Using an existing modifier» | EscrowManager

Severity: Information

Status: Resolved

Description: The swapRedemption, swapIssuance and cancelIssuance functions contain the same logic that already exists in the onlyOrderType modifier.

Recommendation: We recommend using existing functionality.

DFM-37 «Unused function» | PoolContract

Severity: Information

Status: Resolved

Description: The calculateAmountStaked function is not used in the contract code and cannot be called externally.

Recommendation: We recommend removing this function.

DFM-38 «The error description does not match the condition» | TokenTvT

Severity: Information

Status: Resolved

Description: The redeem function checks the condition block.timestamp < subscriptionPeriod, but returns the error ErrorCondition.INSUFFICIENT_BALANCE.

Recommendation: Change the error to something more appropriate.

DFM-39 «Duplicate function» | EscrowManager

Severity: Information

Status: Resolved

Description: The checkIssuanceEscrowConditionsIssuer function and checkIssuanceEscrowConditionsIssuerToken contain the same functionality.

Recommendation: The checkIssuanceEscrowConditionsIssuer function can be removed.

DFM-40 «Unused validator address» |

PoolController
PoolContract

Severity: Information

Status: Resolved

Description: The PoolController and PoolContract contracts store VALIDATOR_ADDRESS and validator addresses, which are not used anywhere.

Recommendation: We recommend removing unused variables.

DFM-41 «Typo» | PoolContract

Severity: Information

Status: Resolved

Description: Typo in TermsUpdated event.

Recommendation: Change "haircut" to "haircut".

DFM-42 «Use readable errors» | *

Severity: Information

Status: Resolved

Description: Contracts often contain asserts or require without an error description, which complicates interaction with contracts and affects debugging and user experience.

Recommendation: We recommend providing at least minimal descriptions or error codes.

DFM-43 «Redundant use of SafeMath» | PoolContract

Severity: Information

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 0.8.0.

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

DFM-44 «Field indexing in events»

Severity: Information

Status: Resolved

Description: The contract uses events for all major operations, but does not use field indexing.

Recommendation: We recommend using the indexing of the main fields in events to simplify the search for them. Events can be an important part of the integration of smart contracts with the UI of the protocol, and can also be used to collect statistics and analyze data.

DFM-45 «Loops optimizations» | *

Severity: Information

Status: Partially Resolved

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

First, 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.

Second, 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
Acknowledged	Project team is made aware of the finding
Open	The finding was not addressed