OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 24	2021-11-20	Zapmore, DoD4uFN	Audit Final

Audit Notes

Audit Date	2021-11-05 - 2021-11-20
Auditor/Auditors	DoD4uFN, MrTeaThyme
Auditor/Auditors Contact Information	contact@obeliskauditing.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB566232221

Disclaimer

This audit is not financial, investment, or any other kind of advice and is for informational purposes only. This report is not a substitute for doing your own research and due diligence. Obelisk is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Obelisk 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. Obelisk 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. The audit is paid by the project but neither the auditors nor Obelisk has any other connection to the project and has no obligations other than to publish an objective report. Obelisk 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. Obelisk assumes that the provided information and material were not altered, suppressed, or misleading. This report is published by Obelisk, and Obelisk 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 Obelisk.

Obelisk Auditing

Defi is a relatively new concept but has seen exponential growth to a point where there is a multitude of new projects created every day. In a fast-paced world like this, there will also be an enormous amount of scams. The scams have become so elaborate that it's hard for the common investor to trust a project, even though it could be legit. We saw a need for creating high-quality audits at a fast phase to keep up with the constantly expanding market. With the Obelisk stamp of approval, a legitimate project can easily grow its user base exponentially in a world where trust means everything. Obelisk Auditing consists of a group of security experts that specialize in security and structural operations, with previous work experience from among other things, PricewaterhouseCoopers. All our audits will always be conducted by at least two independent auditors for maximum security and professionalism.

As a comprehensive security firm, Obelisk provides all kinds of audits and project assistance.

Audit Information

The auditors always conducted a manual visual inspection of the code to find security flaws that automatic tests would not find. Comprehensive tests are also conducted in a specific test environment that utilizes exact copies of the published contract.

While conducting the audit, the Obelisk 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, Obelisk assesses the risks and assigns a risk level to each section together with an explanatory comment. Take note that the comments from the project team are their opinion and not the opinion of Obelisk.

Table of Contents

Project Information

Name	DarkMatterDefi
Description	Dark Matter DeFi (DMD) is a project that was created with the vision to bring NFT staking with a storyline to the world of DeFi farming. NFT's are everywhere but none really are born with an evolving storyline behind them and we aim to change that.
Website	https://www.darkmatterdefi.com/
Contact	@billcashidy
Contact information	@billcashidy on TG
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Fantom

Audit of DarkMatter

> **The DarkMatter team corrected issues found that could be fixed without redeploying. Low-Risk issues found which would require redeploying is still open.**

Obelisk was commissioned by DarkMatter on the 19th of October 2021 to conduct a comprehensive audit of DarkMatters' contracts. The following audit was conducted between the 5th of November 2021 and the 20th of November 2021. Two of Obelisk's security experts went through the related contracts manually using industry standards to find if any vulnerabilities could be exploited either by the project team or users.

The audit of DarkMatter was conducted on already deployed contracts. The most severe issue found was that the timelock was only set to 6 hours (issue #1). The project team quickly moved to a 72-hour timelock which is the recommended length for everyone to be able to react to changes. White initiating the contract on-chain, the start time could be set to something else rather than the current time, however as we audited the already deployed contracts, and this was not changed during deployment, this is not an issue in this specific deployment.

Issue #3 and issue #8 both are still open, as they require deployment to be solved on-chain. Issue #3 has very little impact on the user in practice hence a low-risk assessment. Issue #8 is something most of these forks have in common and is related to the reward going slightly down over a longer timeframe, which is not very noticeable in practice, hence the low-risk assessment.

Regarding issue #5, the currently deployed pools did not use tokens with unusual transfer codes. Further deployed pools are not checked.

The informational findings are good to know while interacting with the project but don't directly damage the project in its current state, hence it's up to the project team if they deem that it's worth solving these issues.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the DarkMatter project**.**

Please read the full document for a complete understanding of the audit.

Summary Table

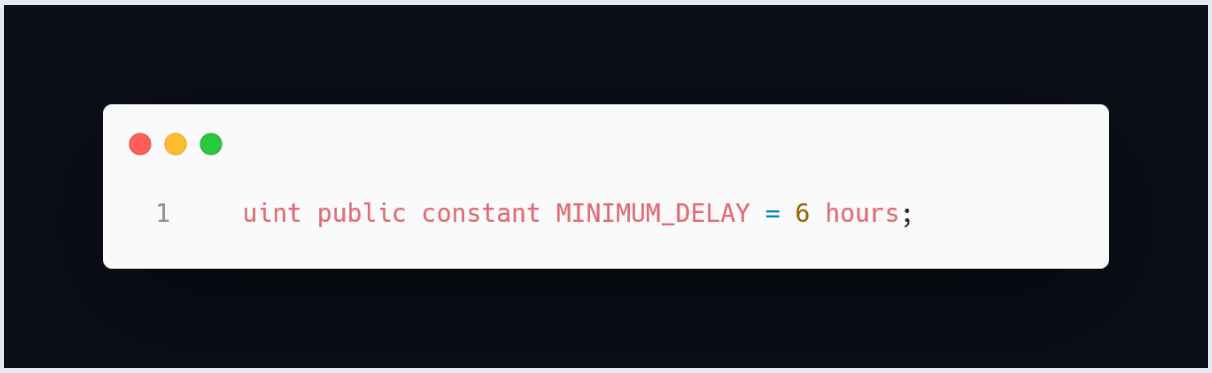
Finding	ID	Severity	Status
Timelock Delay Minimum Delay Is Short	#0001	Medium Risk	Closed
Start Time Can Be Set To The Past	#0002	Low Risk	Mitigated
Unbounded Loop	#0003	Low Risk	Open
No Check For Pool Existence	#0004	Informational	Open
Use Safe Math	#0005	Low Risk	Mitigated
Missing Zero Checks	#0006	Informational	Open
No Events Emitted For Changes To Protocol Values	#0007	Informational	Open
User Funds And Reward Tokens Are Not Separated	#0008	Low Risk	Open

Findings

Manual Analysis

Timelock Delay Minimum Delay Is Short

FINDING ID	#0001
SEVERITY	Medium Risk
STATUS	Closed
LOCATION	Timelock.sol -> 22

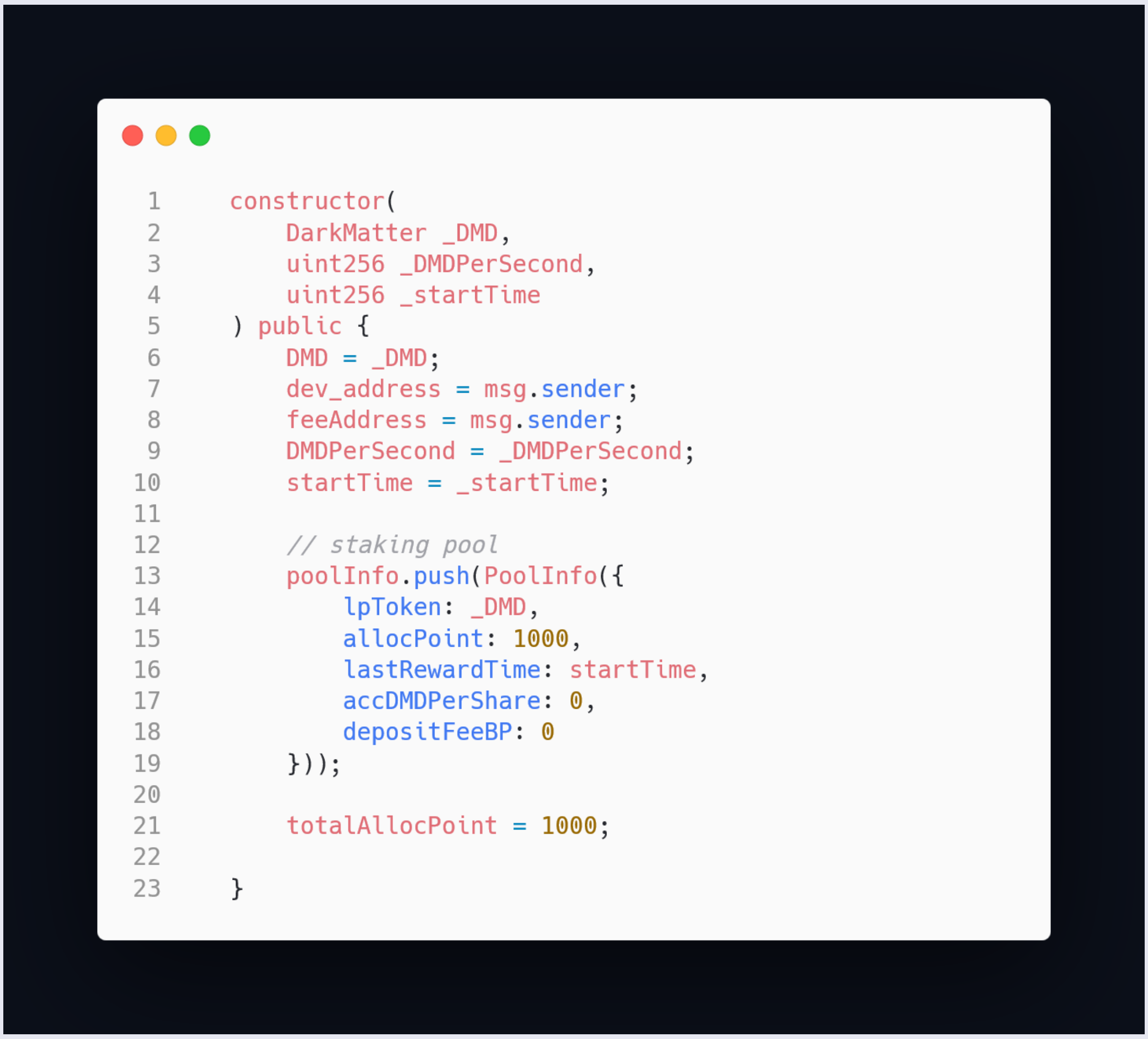


```
uint public constant MINIMUM_DELAY = 6 hours;
```

DESCRIPTION	The timelock delay is set to 6 hours. Obelisk recommends a timelock delay for all functionality of at least 72 hours.
RECOMMENDATION	Set the minimum delay of timelock to 72 hours.
RESOLUTION	The project team has implemented the recommended fix and deployed a new timelock. Reviewed in Timelock 0xc33252B8F55750AF89e012Da075F893175126648

Start Time Can Be Set To The Past

FINDING ID	#0002
SEVERITY	Low Risk
STATUS	Mitigated
LOCATION	MasterChef_DarkMatter.sol -> 1858-1880



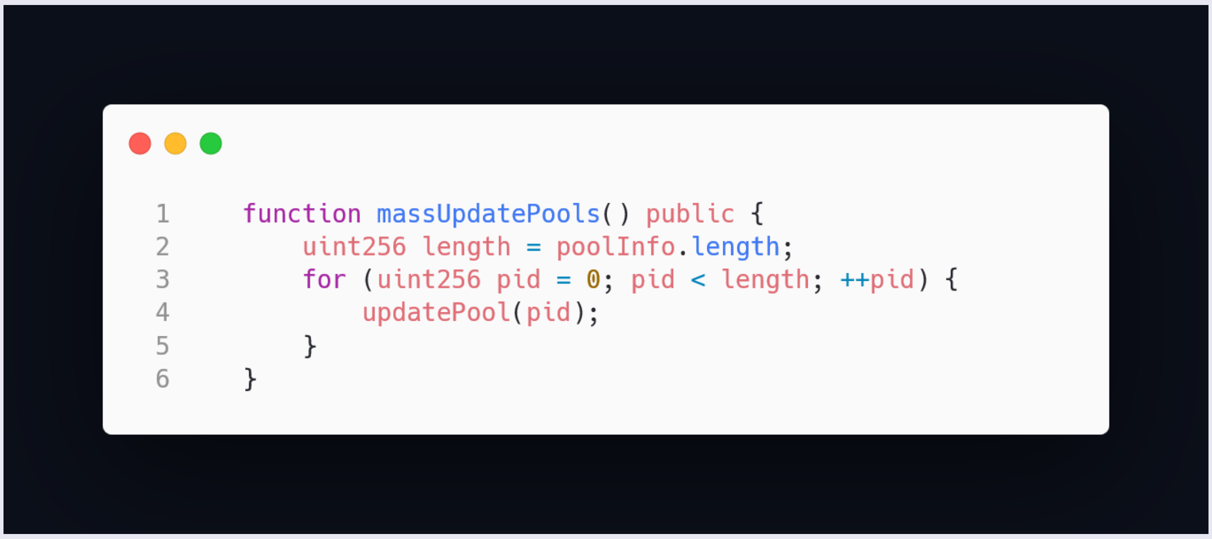
```
constructor(
    DarkMatter _DMD,
    uint256 _DMDPerSecond,
    uint256 _startTime
) public {
    DMD = _DMD;
    dev_address = msg.sender;
    feeAddress = msg.sender;
    DMDPerSecond = _DMDPerSecond;
    startTime = _startTime;

    // staking pool
    poolInfo.push(PoolInfo({
        lpToken: _DMD,
        allocPoint: 1000,
        lastRewardTime: startTime,
        accDMDPerShare: 0,
        depositFeeBP: 0
    }));

    totalAllocPoint = 1000;

}
```

DESCRIPTION	The value of *startTime* can be set to any value since it's directly assigned to a constructor parameter.
RECOMMENDATION	Ensure that *startTime* is not in the past.
RESOLUTION	The required statement that was added, is checking if the start time is not equal to block.timestamp. Which means that it still can be set to the past. The changes were not deployed on-chain. However, the

	start time was deployed correctly, mitigating the issue. Reviewed in commit 9f0a76e1751660b626df12c95de61e9244b22e87 Changes not deployed on chain - MasterChef_DarkMatter 0x7C36c64811219CF9B797C5D9b264d9E7cdade7a4

Unbounded Loop

FINDING ID	#0003
SEVERITY	Low Risk
STATUS	Open
LOCATION	MasterChef_DarkMatter.sol -> 1944-1949

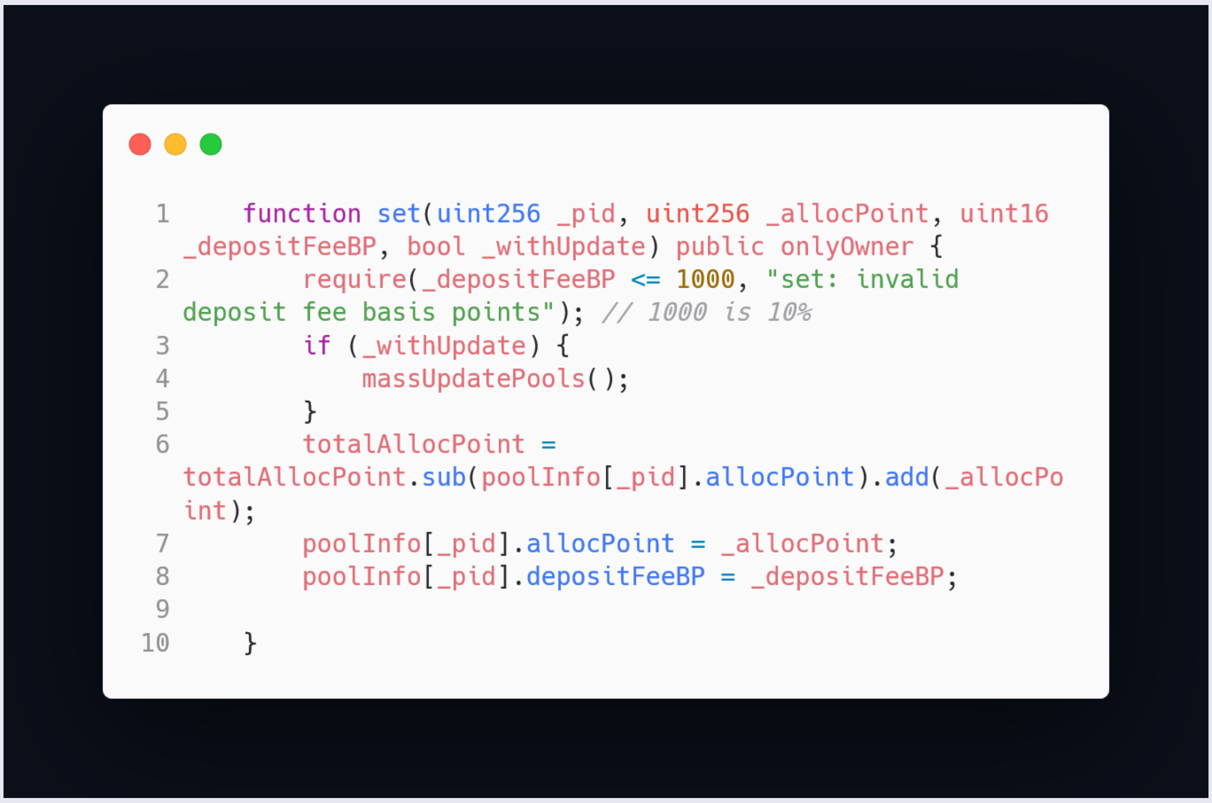


```
function massUpdatePools() public {
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(pid);
    }
}
```

DESCRIPTION	Iterating over an unbounded array can cause transactions to revert due to the gas limit. This can be bypassed in most cases by either updating pools individually or in the case of adding pools, not using *withUpdate*. However, when updating *DMDPerSecond*, this may become an issue.
RECOMMENDATION	Provide a limit to the size of the array. Alternatively, pass a lower and upper index as parameters and iterate over a range.
RESOLUTION	Team comment: 'No solution, leaving it as is just like every other project.' Reviewed in commit 9f0a76e1751660b626df12c95de61e9244b22e87 Changes not deployed on chain - MasterChef_DarkMatter 0x7C36c64811219CF9B797C5D9b264d9E7cdade7a4

No Check For Pool Existence

FINDING ID	#0004
SEVERITY	Informational
STATUS	Open
LOCATION	MasterChef_DarkMatter.sol -> 1913-1922

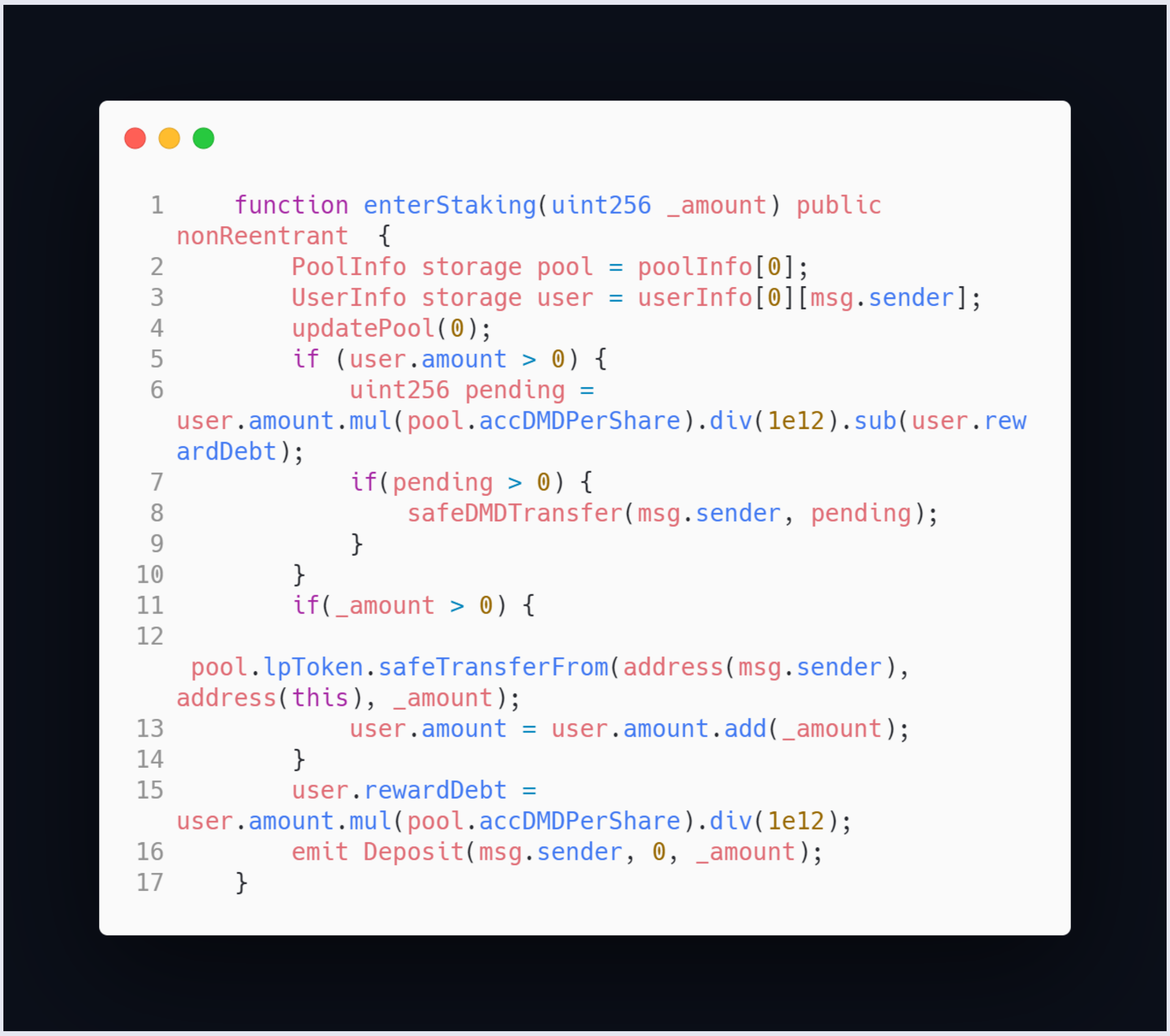


```
    function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner {
        require(_depositFeeBP <= 1000, "set: invalid deposit fee basis points"); // 1000 is 10%
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
        poolInfo[_pid].depositFeeBP = _depositFeeBP;

    }
```

DESCRIPTION	There is no check for whether a pool exists when trying to set its parameters.
RECOMMENDATION	Add a check.
RESOLUTION	The project team has implemented the recommended fix but has not deployed the changes. Reviewed in commit 9f0a76e1751660b626df12c95de61e9244b22e87 Changes not deployed on chain - MasterChef_DarkMatter 0x7C36c64811219CF9B797C5D9b264d9E7cdade7a4

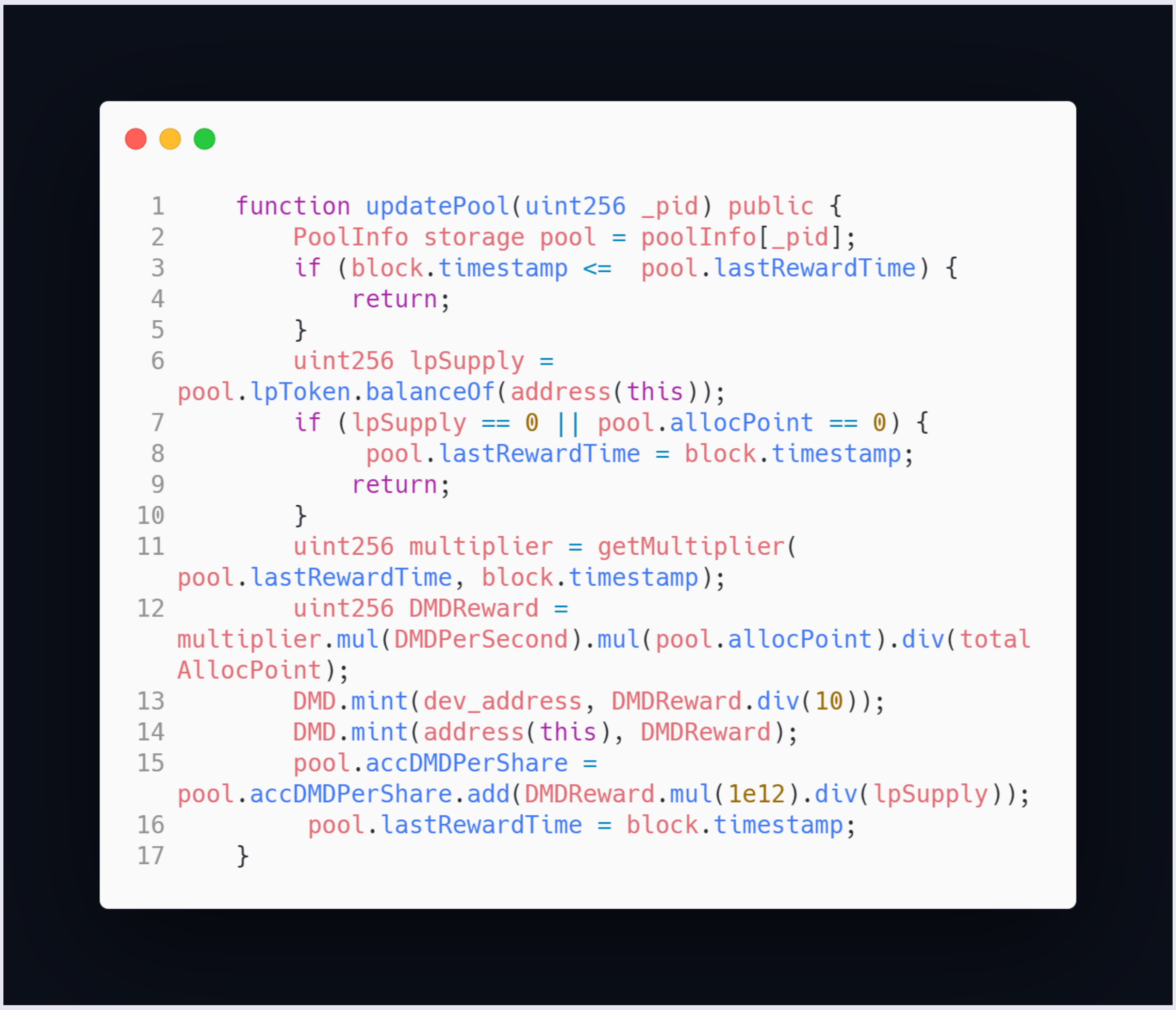
User Funds And Reward Tokens Are Not Separated

FINDING ID	#0008
SEVERITY	Low Risk
STATUS	Open
LOCATION	MasterChef_DarkMatter.sol -> 2049-2065



```
    function enterStaking(uint256 _amount) public nonReentrant  {
        PoolInfo storage pool = poolInfo[0];
        UserInfo storage user = userInfo[0][msg.sender];
        updatePool(0);
        if (user.amount > 0) {
            uint256 pending = user.amount.mul(pool.accDMDPerShare).div(1e12).sub(user.rewardDebt);
            if(pending > 0) {
                safeDMDTransfer(msg.sender, pending);
            }
        }
        if(_amount > 0) {
            pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
            user.amount = user.amount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accDMDPerShare).div(1e12);
        emit Deposit(msg.sender, 0, _amount);
    }
```

LOCATION	MasterChef_DarkMatter.sol -> 1980-1996



```
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.timestamp <=  pool.lastRewardTime) {
            return;
        }
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (lpSupply == 0 || pool.allocPoint == 0) {
             pool.lastRewardTime = block.timestamp;
            return;
        }
        uint256 multiplier = getMultiplier(pool.lastRewardTime, block.timestamp);
        uint256 DMDReward = multiplier.mul(DMDPerSecond).mul(pool.allocPoint).div(totalAllocPoint);
        DMD.mint(dev_address, DMDReward.div(10));
        DMD.mint(address(this), DMDReward);
        pool.accDMDPerShare = pool.accDMDPerShare.add(DMDReward.mul(1e12).div(lpSupply));
         pool.lastRewardTime = block.timestamp;
    }
```

DESCRIPTION	The reward tokens, as well as *pool.lpToken* of pool 0, are the DMD token. Because the unclaimed rewards are held in the masterchef, it does not distinguish between user deposits and unclaimed rewards. Therefore, the rewards for pool 0 are diluted.
RECOMMENDATION	Distinguish the user funds from the DMD minted as rewards.
RESOLUTION	No changes were made. Reviewed in commit 9f0a76e1751660b626df12c95de61e9244b22e87 Changes not deployed on-chain - MasterChef_DarkMatter 0x7C36c64811219CF9B797C5D9b264d9E7cdade7a4

Static Analysis

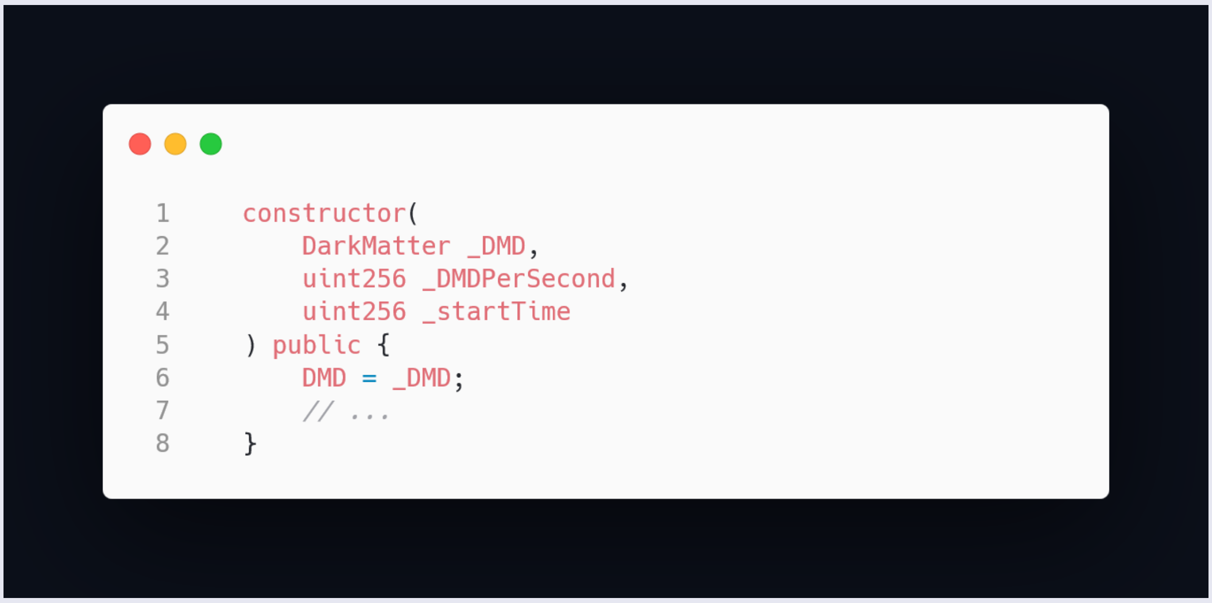
Use Safe Math

FINDING ID	#0005
SEVERITY	Low Risk
STATUS	Mitigated
LOCATION	MasterChef_DarkMatter.sol -> 1985: _*amount* = *pool.lpToken.balanceOf(address(this)) - balanceBefore;*

DESCRIPTION	SafeMath should be used when subtracting. Tokens that have malicious transfer code can potentially exploit this to drain the pool.
RECOMMENDATION	Use Openzeppelin's SafeMath functions. These SafeMath functions are used to catch subtraction underflows.
RESOLUTION	The project team has implemented the recommended fix but has not deployed the changes. Deployed pools did not use tokens with unusual transfer codes. Further analysis of the pool tokens was not conducted. Reviewed in commit 3e30dc7dbfc98c37143f66a2f3d33beef1220a00 Changes not deployed on chain - MasterChef_DarkMatter 0x7C36c64811219CF9B797C5D9b264d9E7cdade7a4

Missing Zero Checks

FINDING ID	#0006
SEVERITY	Informational
STATUS	Open
LOCATION	DarkMatter.sol -> 342: *function setMasterChef(address _address) public onlyOwner* DarkMatter.sol -> 348: *function setlockliquidity(address _address) public onlyOwner* DarkMatter.sol -> 476: *function setPresale(address _presale) external onlyOwner* MasterChef_DarkMatter.sol -> 1858-1880



```
constructor(
    DarkMatter _DMD,
    uint256 _DMDPerSecond,
    uint256 _startTime
) public {
    DMD = _DMD;
    // ...
}
```

DESCRIPTION	The contract address values can be set to zero address in various constructors, initializers, and setter functions. Zero addresses may cause incorrect contract behavior.
RECOMMENDATION	Add a check to ensure contract values are never set to invalid zero addresses.
RESOLUTION	The values *_dev_Address* and *_feeAddress*, are not being set at the constructor. The value of *_DMD* is still missing from MasterChef_DarkMatter's constructor. Reviewed in commit [9f0a76e1751660b626df12c95de61e9244b22e87](9f0a76e1751660b626df12c95de61e9244b22e87)

	Changes not deployed on chain - MasterChef_DarkMatter 0x7C36c64811219CF9B797C5D9b264d9E7cdade7a4

No Events Emitted For Changes To Protocol Values

FINDING ID	#0007
SEVERITY	Informational
STATUS	Open
LOCATION	DarkMatter.sol -> 342: *function setMasterChef(address _address) public onlyOwner* DarkMatter.sol -> 348: *function setlockliquidity(address _address) public onlyOwner* DarkMatter.sol -> 380: *function setDeflationController(address _address) external onlyOwner* DarkMatter.sol -> 431: *function addMinter(address _addMinter) public onlyOwner returns (bool)* DarkMatter.sol -> 439: *function removeMinter(address _removeMinter) public onlyOwner returns (bool)* DarkMatter.sol -> 476: *function setPresale(address _presale) external onlyOwner* MasterChef_DarkMatter.sol -> 1893: *function add(uint256 _allocPoint, IERC20 _lpToken, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner nonDuplicated(_lpToken)* MasterChef_DarkMatter.sol -> 1913: *function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner* MasterChef_DarkMatter.sol -> 2096: *function setDMDPerSecond (uint256 _DMDPerSecond) public onlyOwner*

DESCRIPTION	Functions that change important variables should emit events such that users can more easily monitor the change.
RECOMMENDATION	Emit events from these functions.
RESOLUTION	The *AddMinter* and *RemoveMinter* events have not been declared. Changes were not deployed. Reviewed in commit [9f0a76e1751660b626df12c95de61e9244b22e87](9f0a76e1751660b626df12c95de61e9244b22e87) Changes not deployed on chain MasterChef_DarkMatter 0x7C36c64811219CF9B797C5D9b264d9E7cdade7a4

	DarkMatter 0x90E892FED501ae00596448aECF998C88816e5C0F

On-Chain Analysis

No Findings

Appendix A - Reviewed Documents

Document	Address
DarkMatter.sol	0x90E892FED501ae00596448aECF998C88816e5C0F
MasterChef_DarkMatter.sol	0x7C36c64811219CF9B797C5D9b264d9E7cdade7a4
Timelock.sol	Old 0x92fa6F5FC4b768E8E32b6Bd3803D5f24759e720d Current - DMD 0xabd9f9ab95804a6d22a485094bb4c3b544a2a831 Current - Masterchef 0xc33252B8F55750AF89e012Da075F893175126648

Externally Owned Accounts

Timelock Admin: 0x365e82adAD2C86D38Bec033be04768c8eCd108e4
Fee Address: 0x1173581baCC46453A2E83550C03FA8B996644b1C

External Contracts

These contracts are not part of the audit scope.

DeflationController: 0xf9C87Dd60cB5b77077Cb281eb21551B7d34c013F
Presale: 0x4aad757bAe148528335339F350Ab08E70094e190
Masterchef Dev - Distribute: 0x46F0E9cC7B8dEA67c6e250b22aC417EcE879c509

Appendix B - Risk Ratings

Risk	Description
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.

Appendix C - Finding Statuses

Closed	Contracts were modified to permanently resolve the finding.
Mitigated	The finding was resolved by other methods such as revoking contract ownership. The issue may require monitoring, for example in the case of a time lock.
Partially Closed	Contracts were updated to fix the issue in some parts of the code.
Partially Mitigated	Fixed by project specific methods which cannot be verified on chain. Examples include compounding at a given frequency.
Open	The finding was not addressed.

Appendix D - Audit Procedure

A typical Obelisk audit uses a combination of the three following methods:

Manual analysis consists of a direct inspection of the contracts to identify any security issues. Obelisk auditors use their experience in software development to spot vulnerabilities. Their familiarity with common contracts allows them to identify a wide range of issues in both forked contracts as well as original code.

Static analysis is software analysis of the contracts. Such analysis is called “static” as it examines the code outside of a runtime environment. Static analysis is a powerful tool used by auditors to identify subtle issues and to verify the results of manual analysis.

On-chain analysis is the audit of the contracts as they are deployed on the block-chain. This procedure verifies that:

- deployed contracts match those which were audited in manual/static analysis;
- contract values are set to reasonable values;
- contracts are connected so that interdependent contract function correctly;
- and the ability to modify contract values is restricted via a timelock or DAO mechanism. (We recommend a timelock value of at least 72 hours)

Each obelisk audit is performed by at least two independent auditors who perform their analysis separately.

After the analysis is complete, the auditors will make recommendations for each issue based on best practice and industry standards. The project team can then resolve the issues, and the auditors will verify that the issues have been resolved with no new issues introduced.

Our auditing method lays a particular focus on the following important concepts:

- Quality code and the use of best practices, industry standards, and thoroughly tested libraries.
- Testing the contract from different angles to ensure that it works under a multitude of circumstances.
- Referencing the contracts through databases of common security flaws.

Follow Obelisk Auditing for the Latest Information

ObeliskOrg ObeliskOrg

OBELISK

Part of Tibereum Group