Munchables Audit Report

Version 1.0

Ardeshir Gholami

June 14, 2024

Munchables Audit Report

Ardeshir Gholami

14 June 2024

Prepared by: Ardeshir Lead Auditors: - 4rdiii

Table of Contents

Protocol Summary

Munchables is a GameFi project with a twist. A web3 point farming game in which Keepers nurture creatures to help them evolve, deploying strategies to earn them rewards in competition with other players.

Disclaimer

I make all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond the following commit hash:

~~~
## Scope
```/src/managers/LockManager.sol     ```


# Executive Summary
Munchables is a GameFi project with a twist.
~~~

~~~

The objective of the game is to earn as many Munch Points as possible. In crypto terms, you could call this "point farming".

Built on top of Blast, Munchables leverages the unique on-chain primitives to create a reward-filled journey.
Players collect Munchables and keep them safe, fed and comfortable in their snuggery.
Once in a snuggery, a Munchable can start earning rewards for that player.
A variety of factors influence the rewards earned, so players will have to be smart when choosing which Munchables to put in their snuggery and the strategies they use to play the game.

Issues found

Severity	number of issues found
High	1
Medium	0
Low	0
Info	0
Total	1

Findings
High
[H-1] `LockManager::lockOnBehalf` Can Be Used to Lock Dust Amounts On behalf Of a Player To Stop Them From Unlocking Their Funds

Impact
As there is no minimum lock amount set for the `lockOnBehalf` method, a malicious actor can deposit dust amounts on behalf of a user, extending their lock duration by the player-set duration and preventing them from ever unlocking by repeating this action.
`lockOnBehalf` has no minimum Amount check:
```javascript
function lockOnBehalf(
        address _tokenContract,
        uint256 _quantity,
        address _onBehalfOf
    )
        external
        payable
        notPaused
        onlyActiveToken(_tokenContract)
        onlyConfiguredToken(_tokenContract)
        nonReentrant
    {
        address tokenOwner = msg.sender;
        address lockRecipient = msg.sender;
        if (_onBehalfOf != address(0)) {
~~~

```
                lockRecipient = _onBehalfOf;
            }

            _lock(_tokenContract, _quantity, tokenOwner, lockRecipient);
        }
```

It's notable that if the player changes the lock duration to zero, it still won't work because you can't reduce the lock duration, and it will revert with `LockDurationReducedError()`. _lock internal function updates the `lockedToken.unlockTime` even with a dust lock amount:

```
function _lock(
        address _tokenContract,
        uint256 _quantity,
        address _tokenOwner,
        address _lockRecipient
    ) private {
        .
        .
        .
        // add remainder from any previous lock
@>      uint32 _lockDuration = playerSettings[_lockRecipient].
    lockDuration;

@>      if (_lockDuration == 0) {
@>          _lockDuration = lockdrop.minLockDuration;
@>      }
        if (
            lockdrop.start <= uint32(block.timestamp) &&
            lockdrop.end >= uint32(block.timestamp)
        ) {
            if (
                _lockDuration < lockdrop.minLockDuration ||
                _lockDuration >
                uint32(configStorage.getUint(StorageKey.MaxLockDuration
                    ))
            ) revert InvalidLockDurationError();
            if (msg.sender != address(migrationManager)) {
                // calculate number of nfts
                remainder = quantity % configuredToken.nftCost;
                numberNFTs = (quantity - remainder) / configuredToken.
                    nftCost;

                if (numberNFTs > type(uint16).max) revert
                    TooManyNFTsError();

                // Tell nftOverlord that the player has new unopened
                    Munchables
                nftOverlord.addReveal(_lockRecipient, uint16(numberNFTs
                    ));
            }
        }

        // Transfer erc tokens
        if (_tokenContract != address(0)) {
            IERC20 token = IERC20(_tokenContract);
            token.transferFrom(_tokenOwner, address(this), _quantity);
        }

        lockedToken.remainder = remainder;
        lockedToken.quantity += _quantity;
@>      lockedToken.lastLockTime = uint32(block.timestamp);
@>      lockedToken.unlockTime =
            uint32(block.timestamp) +
            uint32(_lockDuration);

        // set their lock duration in playerSettings
        playerSettings[_lockRecipient].lockDuration = _lockDuration;

        emit Locked(
            _lockRecipient,
            _tokenOwner,
            _tokenContract,
            _quantity,
            remainder,
            numberNFTs,
            _lockDuration
        );
    }
```

Proof of Concept Here are the attack steps: 1. User locks 100 ethers. 2. User sets lock duration to 90 days. 3. 90 days pass. 4. User intends to call unlock, but the attacker has called `lockOnBehalf` earlier, so it will revert.

```
    function test_DOS_lockOnBehalf() public {
        hacker = makeAddr("hacker");
        vm.deal(hacker, 1 ether);
        uint256 dustAmount = 1 wei;
        uint256 lockAmount = 100e18;
        deployContracts();
        // register me
        amp.register(MunchablesCommonLib.Realm(3), address(0));
        // lock tokens
        lm.lock{value: lockAmount}(address(0), lockAmount);
        lm.setLockDuration(90 days);
        uint256 unlockTime = block.timestamp + 90 days;
        vm.warp(unlockTime);

        /// attacker calls onlockbehalf before the user calls the
           unlock
        vm.prank(hacker);
        lm.lockOnBehalf{value: dustAmount}(
            address(0),
            dustAmount,
            address(this)
        );
        /// user calls unlock
        vm.expectRevert(ILockManager.TokenStillLockedError.selector);
        lm.unlock(address(0), lockAmount);

                ///user intentds to change lock time
        vm.expectRevert(ILockManager.LockDurationReducedError.selector)
            ;
        lm.setLockDuration(0);
    }
```

Tools Used Manual Review, Foundry #### Recommended Mitigation Steps To fix this, a minimum lock amount can be set for different tokens in `lockOnBehalf` function to prevent malicious actors from attacking by increasing the cost of the attack.

```
function lockOnBehalf(
        address _tokenContract,
        uint256 _quantity,
        address _onBehalfOf
    )
        external
        payable
        notPaused
        onlyActiveToken(_tokenContract)
        onlyConfiguredToken(_tokenContract)
        nonReentrant
    {
        address tokenOwner = msg.sender;
        address lockRecipient = msg.sender;
        if (_onBehalfOf != address(0)) {
            lockRecipient = _onBehalfOf;
        }
+       if (_quantity < MIN_LOCK_AMOUNT){
+           revert("lock amount is less than minimum");
+       }

        _lock(_tokenContract, _quantity, tokenOwner, lockRecipient);
    }
```