MythX

REPORT 60658047B857B00011F17F11

Created	Thu Apr 01 2021 08:11:51 GMT+0000 (Coordinated Universal Time)
Number of analyses	1

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
230e5e7a-02f6-4a62-b10b-acfca2b19933	/contracts/tokenlock.sol	9

Analysis 230e5e7a-02f6-4a62-b10b-acfca2b19933

MythX

Started	Thu Apr 01 2021 08:12:01 GMT+0000 (Coordinated Universal Time)
Finished	Thu Apr 01 2021 08:27:05 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Client Tool	Mythx-Vscode-Extension
Main Source File	/Contracts/Tokenlock.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	2	7

ISSUES

MEDIUM SWC-107 Write to persistent state following external call

The contract account state is accessed after an external call to a user defined address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/tokenlock.sol

Locations

```

locks[address(_token)][msg.sender].push(TokenSet({
amount: depositAmount,
until: _until,
withdrawn: 0
}));

emit Deposit(address(_token), msg.sender, depositAmount, _until);
emit Locked(depositAmount, _until);
}
```

MEDIUM SWC-128 Loop over unbounded data structure.

Gas consumption in function "withdraw" in contract "TokenLock" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

/contracts/tokenlock.sol

Locations

```
for (uint256 i; i<tokenSets.length; i++) {
uint256 withdrawn;
uint256 availableAmount = tokenSets[i].amount.sub(tokenSets[i].withdrawn);

if (tokenSets[i].until <= block.timestamp) {
```

LOW

SWC-107

A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

/contracts/tokenlock.sol

Locations

```
uint256 beforeLockBal = _token.balanceOf(address(this));
_token.transferFrom(msg.sender, address(this), _amount);
uint256 afterLockBal = _token.balanceOf(address(this));

uint256 depositAmount = afterLockBal.sub(beforeLockBal);
```

LOW

SWC-107

A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

/contracts/tokenlock.sol

Locations

```
}

emit Withdraw(address(_token), msg.sender, transferAmount, block.timestamp);
}
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

/contracts/tokenlock.sol

Locations

```
// Don't believe the amount, token could have deflation
uint256 beforeLockBal = _token.balanceOf(address(this));
_token.transferFrom(msg.sender, address(this), _amount);
uint256 afterLockBal = _token.balanceOf(address(this));
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

/contracts/tokenlock.sol

Locations

```
uint256 beforeLockBal = _token.balanceOf(address(this));
_token.transferFrom(msg.sender, address(this), _amount);
uint256 afterLockBal = _token.balanceOf(address(this));

uint256 depositAmount = afterLockBal.sub(beforeLockBal);
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/contracts/tokenlock.sol

Locations

```
function deposit(IBEP20 _token, uint256 _amount, uint256 _until) external {
require(_until > block.timestamp, "Not in future");
require(_amount > 0, "no amount");

uint256 balance = _token.balanceOf(msg.sender);
require(balance >= _amount, "insufficient funds");
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/contracts/tokenlock.sol

Locations

```

if (tokenSets[i].until <= block.timestamp) {
if (availableAmount <= currentAmount) {
withdrawn = availableAmount;
} else {
withdrawn = currentAmount;
}
}

locks[address(_token)][msg.sender][i].withdrawn = withdrawn;

currentAmount = currentAmount.sub(withdrawn);
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

/contracts/tokenlock.sol

Locations

```

uint256 balance = _token.balanceOf(msg.sender);
require(balance >= _amount, "insufficient funds");

// Don't believe the amount, token could have deflation
uint256 beforeLockBal = _token.balanceOf(address(this));
_token.transferFrom(msg.sender, address(this), _amount);
```