Smart Contract Security Audit Report

The SlowMist Security Team received the Movie Bloc team's application for smart contract security audit of the MBL Token on November 26, 2019. The following are the details and results of this smart contract security audit:

The Token name :

MBL

The Contract Filename and SHA256 :

mbl.py(SHA256):

c0eebf61f3834bf3e66827355b97ce4d78a868cd5b8a248d17f2af8cf8d06401

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Compiler version security	Passed
		Hard-coded address security	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Design Logic Audit	-	Passed
7	“False Deposit” vulnerability Audit	-	Passed
8	Malicious Event Log Audit	-	Passed
9	Scoping and Declarations Audit	-	Passed
10	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
11	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X001911280001

Audit Date : November 28, 2019

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. The contract does not have the Overflow and the Race Conditions issue. The owner can freeze any account's transaction.

The source code:

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

```
OntCversion = '2.0.0'
from ontology.interop.System.Storage import GetContext, Get, Put, Delete
from ontology.interop.System.Runtime import Notify, CheckWitness
from ontology.interop.System.Action import RegisterAction
from ontology.builtins import concat
from ontology.interop.Ontology.Runtime import Base58ToAddress
from ontology.interop.System.ExecutionEngine import GetCallingScriptHash
TransferEvent = RegisterAction("transfer", "from", "to", "amount")
ApprovalEvent = RegisterAction("approval", "owner", "spender", "amount")
ContractFreezeEvent = RegisterAction("FreezeState", "state")
UserFreezeEvent = RegisterAction("FreezeState", "user","state")
ctx = GetContext()
NAME = 'Movie Bloc'
SYMBOL = 'MBL'
DECIMALS = 8
FACTOR = 100000000  # 10^{DECIMALS} = 10^8
OWNER = Base58ToAddress("AJd13sLekkjHeLS68oW2f6ZuHLPvTDcBnn")  # TODO: Input Owner Address
TOTAL_AMOUNT = 30000000000 # TODO: Input Desired Supply Value
BALANCE_PREFIX = bytearray(b'\x01')
```

```
APPROVE_PREFIX = b'\x02'
CONTRACT_FREEZE_PREFIX = b'\x03'
USER_FREEZE_PREFIX = b'\x04'
SUPPLY_KEY = 'TotalSupply'


def Main(operation, args):
    if operation == 'init':
        return init()
    if operation == 'name':
        return name()
    if operation == 'symbol':
        return symbol()
    if operation == 'decimals':
        return decimals()
    if operation == 'totalSupply':
        return totalSupply()
    if operation == 'balanceOf':
        require(len(args) == 1, "Invalid Argument Number")
        acct = args[0]
        return balanceOf(acct)
    if operation == 'transfer':
        require(len(args) == 3, "Invalid Argument Number")
        from_acct = args[0]
        to_acct = args[1]
        amount = args[2]
        return transfer(from_acct,to_acct,amount)
    if operation == 'transferFrom':
        require(len(args) == 4, "Invalid Argument Number")
        spender = args[0]
        from_acct = args[1]
        to_acct = args[2]
        amount = args[3]
        return transferFrom(spender,from_acct,to_acct,amount)
    if operation == 'approve':
        require(len(args) == 3, "Invalid Argument Number")
        owner   = args[0]
        spender = args[1]
        amount = args[2]
        return approve(owner,spender,amount)
    if operation == 'allowance':
        require(len(args) == 2, "Invalid Argument Number")
```

```
            owner = args[0]
            spender = args[1]
            return allowance(owner,spender)
        if operation == 'freeze':
            return freeze()
        if operation == "viewFreeze":
            return viewFreeze()
        if operation == 'freezeUser':
            require(len(args) == 1, "Invalid Argument Number")
            acct = args[0]
            return freezeUser(acct)
        if operation == 'unfreezeUser':
            require(len(args) == 1, "Invalid Argument Number")
            acct = args[0]
            return unfreezeUser(acct)
        if operation == 'viewFreezeUser':
            require(len(args) == 1, "Invalid Argument Number")
            acct = args[0]
            return viewFreezeUser(acct)
        return False
def init():
        requireIsAddress(OWNER)
        requireWitness(OWNER)
        require(totalSupply() == 0, "Contract already inited")

        total = TOTAL_AMOUNT * FACTOR
        Put(ctx,SUPPLY_KEY,total)

        Put(ctx,concat(BALANCE_PREFIX,OWNER),total)
        Put(ctx,CONTRACT_FREEZE_PREFIX,"False")
        TransferEvent("", OWNER, total)
        return True

def name():
        return NAME
def symbol():
        return SYMBOL
def decimals():
        return DECIMALS
def totalSupply():
        return Get(ctx, SUPPLY_KEY)
```

```
def balanceOf(account):
    requireIsAddress(account)
    key = getBalanceKey(account)
    Notify([Get(ctx, key)])
    return Get(ctx, key)

def transfer(from_acct,to_acct,amount):
    requireIsAddress(from_acct)
    requireIsAddress(to_acct)

    requireWitness(from_acct)

    requireNotContractFreeze()
    requireNotFreeze(from_acct)
    requireNotFreeze(to_acct)

    fromKey = getBalanceKey(from_acct)
    fromBalance = Get(ctx,fromKey)

    require(fromBalance >= amount, "Exceeded tokens held")

    if amount == fromBalance:
        Delete(ctx,fromKey)
    else:
        Put(ctx,fromKey,fromBalance - amount)

    toKey = getBalanceKey(to_acct)
    toBalance = Get(ctx,toKey)
    Put(ctx,toKey,toBalance + amount)

    TransferEvent(from_acct, to_acct, amount)
    return True
def approve(owner,spender,amount):
    requireIsAddress(owner)
    requireIsAddress(spender)
    requireWitness(owner)

    require(amount >= 0, "Zero or more tokens")
    require(amount <= balanceOf(owner), "Exceeded tokens held")

    key = getAllowanceKey(owner, spender)
```

```
    Put(ctx, key, amount)

    ApprovalEvent(owner, spender, amount)
    return True
def transferFrom(spender,from_acct,to_acct,amount):
    requireIsAddress(spender)
    requireIsAddress(from_acct)
    requireIsAddress(to_acct)

    requireWitness(spender)

    requireNotContractFreeze()
    requireNotFreeze(spender)
    requireNotFreeze(from_acct)
    requireNotFreeze(to_acct)

    fromKey = getBalanceKey(from_acct)
    fromBalance = Get(ctx, fromKey)
    require(amount <= fromBalance, "Exceed tokens held")

    approveKey = getAllowanceKey(from_acct, spender)
    approvedAmount = Get(ctx,approveKey)
    require(amount <= approvedAmount, "Exceeded tokens approved")

    if amount == approvedAmount:
        Delete(ctx, approveKey)
    else:
        Put(ctx, approveKey, approvedAmount - amount)

    if amount == fromBalance:
        Delete(ctx,fromKey)
    else:
        Put(ctx, fromKey, fromBalance - amount)

    toKey = getBalanceKey(to_acct)
    toBalance = Get(ctx, toKey)
    Put(ctx, toKey, toBalance + amount)

    TransferEvent(from_acct, to_acct, amount)
    return True
def allowance(owner,spender):
```

```
    key = getAllowanceKey(owner, spender)
    Notify([Get(ctx, key)])

    return Get(ctx,key)
```

//SlowMist// The owner can freeze any account transaction

```
## Freeze Function
def freeze():
    onlyOwner()
    freeze_state = Get(ctx,CONTRACT_FREEZE_PREFIX)
    if freeze_state == "False":
        Put(ctx, CONTRACT_FREEZE_PREFIX,"True")
        ContractFreezeEvent("true")
    else:
        Put(ctx, CONTRACT_FREEZE_PREFIX,"False")
        ContractFreezeEvent("false")
    return True
def viewFreeze():
    freeze_state = Get(ctx,CONTRACT_FREEZE_PREFIX)
    ContractFreezeEvent(freeze_state)
    return freeze_state
def freezeUser(account):
    onlyOwner()
    requireIsAddress(account)
    userkey = concat(USER_FREEZE_PREFIX,account)
    Put(ctx, userkey, "True")
    UserFreezeEvent(account, 'true')
    return True
def unfreezeUser(account):
    onlyOwner()
    requireIsAddress(account)
    userkey = concat(USER_FREEZE_PREFIX,account)
    Put(ctx, userkey, "False")
    UserFreezeEvent(account, 'false')
    return True
def viewFreezeUser(account):
    requireIsAddress(account)
    userkey = concat(USER_FREEZE_PREFIX, account)
    freeze_state = Get(ctx, userkey)
    UserFreezeEvent(account,freeze_state)
```

```
    return freeze_state
    ## Helper Function
def requireNotFreeze(address):
    key = Get(ctx, concat(USER_FREEZE_PREFIX, address))
    require(key == "False" or key == None, "Address is frozen")
def requireNotContractFreeze():
    key = Get(ctx, CONTRACT_FREEZE_PREFIX)
    require(key == "False", "Contract is frozen")
def getBalanceKey(address):
    key = concat(BALANCE_PREFIX, address)
    return key
def getAllowanceKey(owner, spender):
    key = concat(concat(APPROVE_PREFIX, owner), spender)
    return key
def requireIsAddress(address):
    require(len(address) == 20, "Address has invalid length")
def requireWitness(address):
    require(CheckWitness(address), "Address is not witness")
def onlyOwner():
    require(CheckWitness(OWNER), "not Owner")
def require(condition,msg):
    if not condition:
        raise Exception(msg)
    return True
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account