Smart Contract Security Audit Report

The SlowMist Security Team received the EnreachDAO team's application for smart contract security audit of the EnreachDAO token on Mar. 02, 2021. The following are the details and results of this smart contract security audit:

Token name :

EnreachDAO

File name and hash(SHA256) :

Nreach.vy: 440a86de2ebd3fce5e430e38004126b1dd99f62c72e7b7e535154773b5fb929e

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 authority audit	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False top-up” vulnerability Audit	-	Passed
9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed

11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : Passed

Audit Number : 0X002103090002

Audit Date : Mar. 09, 2021

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. The total amount of tokens in the contract remains unchanged. The contract does not have the Overflow and the Race Conditions issue.

The source code:

```
# @version ^0.2.11
# @dev Implementation of multi-layers space and time rebasing ERC-20 token standard.
# @dev copyright kader@enreach.io and kashaf@enreach.io
# based on https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

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

from vyper.interfaces import ERC20

implements: ERC20

event Transfer:
    sender: indexed(address)
    receiver: indexed(address)
    value: uint256
```

```
event Approval:
    owner: indexed(address)
    spender: indexed(address)
    value: uint256

event ReceivedEther:
    sender: indexed(address)
    value: uint256

event OwnershipTransferred:
    previousOwner: indexed(address)
    newOwner: indexed(address)

# EIP-20 compliant name symbol and decimals
name: public(String[64])
symbol: public(String[32])
decimals: public(uint256)

# additional decimals used for calculations
scale: public(uint256)

# exponent
expanse: public(int128)
extent: public(uint256)
extent_max: public(uint256)

# temporal timer
initpulse: public(uint256)
nextpulse: public(uint256)

struct Account:
    amount: uint256
    lode: uint256
    expanse: int128



struct Lode:
    total: uint256
    total_e: uint256
    expanse: int128
    tax_id: uint256
    itaxfree: bool
    etaxfree: bool

NUM_OF_TEMPORAL_LODES: constant(uint256) = 25
STAKING_LODE: constant(uint256) = NUM_OF_TEMPORAL_LODES   # 25
FROZEN_LODE: constant(uint256) = STAKING_LODE + 1  #26
RESERVE_LODE: constant(uint256) = FROZEN_LODE + 1  #27
SAFE_LODE: constant(uint256) = RESERVE_LODE + 1  #28
RESERVED1_LODE: constant(uint256) = SAFE_LODE + 1  #29
NUM_OF_LODES: constant(uint256) = 32
NUM_OF_TAX_POLICIES: constant(uint256) = 4


owner: address
currentLode: public(uint256)
transferLocked: public(bool)
taxOn: public(bool)
temporal_tax_num: public(uint256)
temporal_tax_num2: public(uint256)
temporal_tax_den: public(uint256)
tax_numerators: public(uint256[NUM_OF_LODES][NUM_OF_TAX_POLICIES])
tax_numeratorsum: public(uint256[NUM_OF_TAX_POLICIES])
tax_denominator: public(uint256[NUM_OF_TAX_POLICIES])
tax_toflush: public(uint256[NUM_OF_TAX_POLICIES])
tax_airdrop_num: public(uint256)
tax_airdrop_den: public(uint256)
lodes: Lode[NUM_OF_LODES]

accounts: HashMap[address, Account]
allowances: HashMap[address, HashMap[address, uint256]]
privileged: HashMap[address, bool]
arbtrust: HashMap[address, bool]


@internal
def _deallocate0(_debtor: address) -> uint256:
    """
    @dev deallocate all funds from a wallet
    @param _debtor The address to deallocate all the funds from.
    @return An uint256 specifying the amount of scaled tokens remaining
    """
    debtor: Account = self.accounts[_debtor]
    slode: Lode = self.lodes[debtor.lode]
    amount_e: uint256 = debtor.amount
    if amount_e == 0:
        self.accounts[_debtor] = empty(Account)
        return 0
    if debtor.expanse != slode.expanse:
        amount_e = shift(debtor.amount, debtor.expanse - slode.expanse)
    amount_s: uint256 = amount_e * slode.total / slode.total_e
    self.accounts[_debtor] = empty(Account)
    self.lodes[debtor.lode].total -= amount_s
    self.lodes[debtor.lode].total_e -= amount_e
    return amount_s

@internal
def _deallocate(_debtor: address, _amount_s: uint256):
    """
    @dev deallocate funds from a wallet
    @param _debtor The address to deallocate the funds from.
    @param _amount_s scaled amount of funds.
    """
    debtor: Account = self.accounts[_debtor]
    slode: Lode = self.lodes[debtor.lode]
    if debtor.expanse != slode.expanse:
        self.accounts[_debtor].amount = shift(debtor.amount, debtor.expanse - slode.expanse)
        self.accounts[_debtor].expanse = slode.expanse
    amount_e: uint256 = _amount_s * slode.total_e / slode.total
    self.accounts[_debtor].amount -= amount_e
    if self.accounts[_debtor].amount < self.scale:
        amount_e += self.accounts[_debtor].amount
        self.accounts[_debtor].amount = 0
        amount_s: uint256 = amount_e * slode.total / slode.total_e
        self.lodes[debtor.lode].total -= amount_s
    else:
        self.lodes[debtor.lode].total -= _amount_s
    self.lodes[debtor.lode].total_e -= amount_e
    if self.accounts[_debtor].amount == 0:
        self.accounts[_debtor] = empty(Account)



@internal
def _allocate(_creditor: address, _amount_s: uint256):
    """
    @dev deallocate funds from a wallet and from a lode
    @param _creditor The address to allocate the funds to.
    @param _amount_s The address to allocate the scaled funds to.
    """
    creditor: Account = self.accounts[_creditor]
    if (creditor.amount ==0) and (creditor.lode ==0):
        if _creditor.is_contract:
            creditor.lode = FROZEN_LODE
            self.accounts[_creditor].lode = FROZEN_LODE
        else:
            creditor.lode = self.currentLode
            self.accounts[_creditor].lode = self.currentLode
    dlode: Lode = self.lodes[creditor.lode]
    if creditor.amount != 0:
        self.accounts[_creditor].amount = shift(creditor.amount, creditor.expanse - dlode.expanse)
    self.accounts[_creditor].expanse = dlode.expanse
    if dlode.total_e == 0:
        self.lodes[creditor.lode].total_e += _amount_s
        self.accounts[_creditor].amount += _amount_s
    else:
        amount_e: uint256 = _amount_s * dlode.total_e / dlode.total
        self.lodes[creditor.lode].total_e += amount_e
        self.accounts[_creditor].amount += amount_e
    self.lodes[creditor.lode].total += _amount_s




@external
def setLode(_wallet:address, _lode:uint256):
    """
    @dev set the lode of a wallet
    @param _wallet The address of the wallet
    @param _lode The lode to which to allocate the wallet
    """
    if (msg.sender == self.owner):
        assert (_lode < NUM_OF_LODES) #, "Out of bounds lode"
    elif (self.privileged[msg.sender] == True):
        assert _lode < NUM_OF_TEMPORAL_LODES #, "Out of bounds lode or access to priviledged lode"
    else:
        raise "Unauthorized"
    amount: uint256 = self._deallocate0(_wallet)
    self.accounts[_wallet].lode = _lode
    self._allocate(_wallet, amount)

@external
def setTaxStatus(_status: bool):
    """
    @dev tax Status (On->True or Off)
    @param _status status of tax
    """
    assert msg.sender == self.owner
    self.taxOn = _status

@external
def setTax(_tax_id:uint256, _tax_numerators:uint256[NUM_OF_LODES], _tax_denominator:uint256):
    """
    @dev set the taxes of a tax_id
    @param _tax_id the tax id
    @param _tax_numerators Tax numerator per lode
    @param _tax_denominator Tax denominator
    """
    assert (msg.sender == self.owner)
    self.tax_numerators[_tax_id] = _tax_numerators
    self.tax_denominator[_tax_id] = _tax_denominator
    sum:uint256 = 0
    for i in range(NUM_OF_LODES):
        sum += _tax_numerators[i]
    self.tax_numeratorsum[_tax_id] = sum




@external
def setLodeTaxId(_lode:uint256, _tax_id:uint256):
    """
    @dev set the tax_id of a lode
    @param _lode the lode number
    @param _tax_id Tax id
    """
    assert (msg.sender == self.owner)
    self.lodes[_lode].tax_id = _tax_id

@external
def setPrivileged(_wallet: address, _status: bool):
    """
    @dev change Privileged status of wallet
    @param _wallet The address of the wallet
    @param _status Which status to set to the wallet
    """
    assert (msg.sender == self.owner)
    self.privileged[_wallet] = _status

@external
def setArbTrusted(_wallet: address, _status: bool):
    """
    @dev change ArbTrust status of wallet
    @param _wallet The address of the wallet
    @param _status Which status to set to the wallet
    """
    assert (msg.sender == self.owner)
    self.arbtrust[_wallet] = _status

@view
@external
def isPrivileged(_wallet: address) -> bool:
    """
    @dev check Privileged status of wallet
    @param _wallet The address of the wallet
    @return A bool specifiying if the wallet is priviledged
    """
    return self.privileged[_wallet]

@view
@external
def getLode(_wallet:address) -> uint256:
    """
    @dev get account lode
    @param _wallet The address of the wallet
    @return An uint256 specifying the lode of the wallet
    """
    assert (msg.sender == self.owner) or self.privileged[msg.sender]
    return self.accounts[_wallet].lode


@view
@internal
def getBalance(_wallet : address) -> uint256:
    """
    @dev get balance of wallet
    @param _wallet The address of the wallet
    @return An uint256 specifying the scaled balance of the wallet
    """
    account: Account = self.accounts[_wallet]
    lode: Lode = self.lodes[account.lode]
    if lode.total_e == 0:
        return 0
    else:
        return shift(account.amount, account.expanse - lode.expanse) * lode.total / lode.total_e

@view
@external
def balanceLode(_wallet : address) -> (uint256, uint256, uint256, int128, int128):
    """
    @dev get detailed balance of a wallet
    @param _wallet the wallet
    @return internal balance of wallet, lode scaled balance, lode internal balance, account and lode expanse
    """
    assert (msg.sender == self.owner) or self.privileged[msg.sender] or (_wallet == msg.sender)
    account: Account = self.accounts[_wallet]
    lode: Lode = self.lodes[account.lode]
    return (account.amount, lode.total, lode.total_e, account.expanse, lode.expanse)

@view
@external
def lodeBalance(_lode: uint256) ->  (uint256, uint256, int128):
    """
    @dev get balance of a lode
    @param _lode lode number
    @return lode scaled balance, lode internal balance and lode expanse
    """
    assert (msg.sender == self.owner) or self.privileged[msg.sender]
    lode: Lode = self.lodes[_lode]
    return (lode.total, lode.total_e, lode.expanse)



@external
def setLodeTaxFree(_lode: uint256, _itaxfree: bool, _etaxfree: bool):
    """
    @dev set lode tax excemptions rules
    @param _lode lode number
    @param _itaxfree is tax free on credit
    @param _etaxfree is tax free on debit
    """
    assert (msg.sender == self.owner)
    self.lodes[_lode].itaxfree = _itaxfree
    self.lodes[_lode].etaxfree = _etaxfree

@view
@external
def getLodeTaxFree(_lode: uint256) -> (bool, bool, uint256):
    """
    @dev get lode tax rules
    @param _lode lode number
    @return _itaxfree, _etaxfree and tax_id
    """
    assert (msg.sender == self.owner) or self.privileged[msg.sender]
    return (self.lodes[_lode].itaxfree, self.lodes[_lode].etaxfree, self.lodes[_lode].tax_id)



@external
def __init__(_name: String[64], _symbol: String[32], _decimals: uint256, _supply: uint256, _transferLocked: bool,
    _tax_nums: uint256[NUM_OF_LODES], _tax_denom: uint256):
    self.owner = msg.sender
    self.tax_numerators[0] = _tax_nums
    for i in range(NUM_OF_LODES):
        self.tax_numeratorsum[0] += _tax_nums[i]
    self.tax_denominator[0] = _tax_denom
    self.tax_airdrop_num = 1
    self.tax_airdrop_den = 20
    self.temporal_tax_num = 10000
    self.temporal_tax_num2 = 2664
    self.temporal_tax_den = 30000
    self.transferLocked = _transferLocked
    self.taxOn = not _transferLocked
    self.scale = 10 ** _decimals
    init_supply: uint256 = _supply * 10 ** _decimals
    self.extent = init_supply * self.scale
    self.extent_max =  init_supply * self.scale * self.scale
    a_supply: uint256 = init_supply * self.scale
    self.name = _name
    self.symbol = _symbol
    self.decimals = _decimals
    self.accounts[msg.sender].amount = a_supply
    self.lodes[self.accounts[msg.sender].lode] = Lode({total: a_supply, total_e: a_supply, expanse: 0, itaxfree:False,
etaxfree:False, tax_id:0})
    self.lodes[STAKING_LODE] = Lode({total: 0, total_e: 0, expanse: 0, itaxfree: True, etaxfree: True, tax_id:0})
    self.lodes[RESERVE_LODE] = Lode({total: 0, total_e: 0, expanse: 0, itaxfree: True, etaxfree: False, tax_id:0})
    self.lodes[RESERVED1_LODE] = Lode({total: 0, total_e: 0, expanse: 0, itaxfree: True, etaxfree: True, tax_id:0})
    log Transfer(ZERO_ADDRESS, msg.sender, init_supply)
    log OwnershipTransferred(ZERO_ADDRESS, msg.sender)




@view
@external
def totalSupply() -> uint256:
    """
    @dev Total number of tokens in existence. EIP-20 function totalSupply()
    @return total supply
    """
    sum:uint256 = 0
    for i in range(NUM_OF_LODES):
        sum += self.lodes[i].total
    return sum / self.scale
```

```
@view
@external
def balanceOf(_wallet : address) -> uint256:
    """
    @dev Total number of tokens in existence. EIP-20 function balanceOf(address _owner)
    @return balance
    """
    return self.getBalance(_wallet) / self.scale

@view
@external
def allowance(_owner : address, _spender : address) -> uint256:
    """
    @dev Function to check the amount of tokens that an owner allowed to a spender.
        EIP-20 function allowance(address _owner, address _spender)
    @param _owner The address which owns the funds.
    @param _spender The address which will spend the funds.
    @return An uint256 specifying the amount of tokens still available for the spender.
    """
    return self.allowances[_owner][_spender]

@external
def setTemporalTax(_num: uint256, _num2: uint256, _den: uint256):
    """
    @dev modify the temporal tax
    @param _num tax numerator
    @param _num2 tax arb
    @param _den tax denominator
    """
    assert msg.sender == self.owner
    assert _den != 0
    self.temporal_tax_num = _num
    self.temporal_tax_num2 = _num2
    self.temporal_tax_den = _den

@internal
def temporalTax() -> bool:
    """
    @dev This function trigger a temporal tax event if required.
    @return True if tax event happened, False otherwise
    """
    if (self.initpulse != 0):
        self.currentLode = ((self.nextpulse - self.initpulse) / 86400) % NUM_OF_TEMPORAL_LODES
        if (block.timestamp > self.nextpulse):
            tax: uint256 = self.lodes[self.currentLode].total * self.temporal_tax_num / self.temporal_tax_den
            self.lodes[self.currentLode].total -= tax
            self.lodes[RESERVE_LODE].total += tax
            self.nextpulse += 86400
            if self.currentLode == 0:
                if (self.temporal_tax_den - self.temporal_tax_num) != 0:
                    self.extent = self.extent * self.temporal_tax_den / (self.temporal_tax_den - self.temporal_tax_num)
                    if self.extent  > self.extent_max:
                        self.extent /= 2
                        self.expanse += 1
            if self.lodes[self.currentLode].expanse != self.expanse:
                self.lodes[self.currentLode].total_e = shift(self.lodes[self.currentLode].total_e,
                    self.lodes[self.currentLode].expanse - self.expanse)
                self.lodes[self.currentLode].expanse = self.expanse
            return True
    return False

@external
def changeTaxAirDrop(_num: uint256, _den:uint256):
    assert (msg.sender == self.owner)
    assert (_den != 0)
    self.tax_airdrop_num = _num
    self.tax_airdrop_den = _den

@external
@view
def simTaxAirDrop() -> uint256:
    sum:uint256 = 0
    for tax_id in range(NUM_OF_TAX_POLICIES):
        tax:uint256 = self.tax_toflush[tax_id]
        if tax != 0:
            sum += tax * self.tax_airdrop_num / self.tax_airdrop_den
    return sum/self.scale

@internal
def distributeTax(_to:address):
    airdrop:uint256 = 0
    for tax_id in range(NUM_OF_TAX_POLICIES):
        tax:uint256 = self.tax_toflush[tax_id]
        if tax != 0:
            airdrop0:uint256 = tax * self.tax_airdrop_num / self.tax_airdrop_den
            airdrop += airdrop0
            tax -= airdrop0
            tax_num:uint256 = self.tax_numeratorsum[tax_id]
            for i in range(NUM_OF_LODES):
                self.lodes[i].total +=  tax * self.tax_numerators[tax_id][i] / tax_num
        self.tax_toflush[tax_id] = 0
    if airdrop != 0:
        self._allocate(_to, airdrop)
    self.temporalTax()



@external
def triggerDistributeTax():
    self.distributeTax(msg.sender)

@external
def triggerTemporalTax() -> bool:
    """
    @dev This function trigger a temporal tax event if required.
    @return True if tax event happened, False otherwise
    """
    return self.temporalTax()

@view
@external
def transferedAfterTax(_debtor: address, _creditor: address, _value: uint256) -> uint256:
    """
    @dev evaluate amount sent during Transfer
    @param _debtor The address to transfer from.
    @param _creditor The address to transfer to.
    @param _value The amount to be transferred.
    @return amount remaining to be transferred
    """
    amount: uint256 = _value * self.scale
    d_lode: uint256 = self.accounts[_debtor].lode
    c_lode: uint256 = self.accounts[_creditor].lode
    tax_id: uint256 = self.lodes[d_lode].tax_id
    if (not self.lodes[d_lode].etaxfree) and (not self.lodes[c_lode].itaxfree) and self.taxOn:
        tax: uint256 = amount * self.tax_numeratorsum[tax_id] / self.tax_denominator[tax_id]
        amount -= tax
    if self.arbtrust[_debtor] and self.arbtrust[_creditor]:
        tax:uint256 = amount * self.temporal_tax_num2 / self.temporal_tax_den
        amount -= tax
    return amount / self.scale



@internal
def _transfer(_debtor: address, _creditor: address, _value: uint256):
    """
    @dev Transfer token for a specified address
    @param _debtor The address to transfer from.
    @param _creditor The address to transfer to.
    @param _value The amount to be transferred.
    """
    #if (block.timestamp > self.nextpulse) and (self.initpulse != 0):
    #     self.temporalTax()
    amount: uint256 = _value * self.scale
    d_lode: uint256 = self.accounts[_debtor].lode
    c_lode: uint256 = self.accounts[_creditor].lode
    tax_id: uint256 = self.lodes[d_lode].tax_id
    self._deallocate(_debtor, amount)
    if (not self.lodes[d_lode].etaxfree) and (not self.lodes[c_lode].itaxfree) and self.taxOn:
        tax: uint256 = amount * self.tax_numeratorsum[tax_id] / self.tax_denominator[tax_id]
        amount -= tax
        self.tax_toflush[tax_id] += tax
    if self.arbtrust[_debtor] and self.arbtrust[_creditor]:
        tax:uint256 = amount * self.temporal_tax_num2 / self.temporal_tax_den
        amount -= tax
        self.lodes[RESERVED1_LODE].total += tax
    if (self.initpulse != 0):
        if (self.currentLode != d_lode) and (d_lode < NUM_OF_TEMPORAL_LODES):
            amount0: uint256 = self._deallocate0(_debtor)
            if amount0 != 0:
                self.accounts[_debtor].lode = self.currentLode
                self._allocate(_debtor, amount0)
    self._allocate(_creditor, amount)



@external
def transfer(_to : address, _value : uint256) -> bool:
    """
    @dev Transfer token for a specified address. EIP-20 function transfer(address _to, uint256 _value)
    @param _to The address to transfer to.
    @param _value The amount to be transferred.
    """
    assert (self.transferLocked == False) or self.privileged[msg.sender] or (msg.sender == self.owner), "You are not allowed
to make transfer"
    self._transfer(msg.sender, _to, _value)
    log Transfer(msg.sender, _to, _value)
    return True #SlowMist# The return value conforms to the EIP20 specification



@external
def transferFrom(_from : address, _to : address, _value : uint256) -> bool:
    """
     @dev Transfer tokens from one address to another. EIP function transferFrom(address _from, address _to, uint256
_value)
     @param _from address The address which you want to send tokens from
     @param _to address The address which you want to transfer to
     @param _value uint256 the amount of tokens to be transferred
    """
    assert (self.transferLocked == False) or self.privileged[msg.sender] or self.privileged[_from] or (msg.sender ==
self.owner), "You are not allowed to make transfer"
    self._transfer(_from, _to, _value)
    self.allowances[_from][msg.sender] -= _value
    log Transfer(_from, _to, _value)
    return True #SlowMist# The return value conforms to the EIP20 specification


@internal
def _approve(_owner: address, _spender : address, _value : uint256) -> bool:
    """
    @dev Approve the passed address to spend the specified amount of tokens on behalf of _owner.
    @param _owner The address which will provide the funds.
    @param _spender The address which will spend the funds.
    @param _value The amount of tokens to be spent.
    """
    self.allowances[_owner][_spender] = _value
    log Approval(_owner, _spender, _value)
    return True


@external
def approve(_spender : address, _value : uint256) -> bool:
    """
    @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
        Beware that changing an allowance with this method brings the risk that someone may use both the old
        and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
        race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
        https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
        EIP-20 function approve(address _spender, uint256 _value)

    @param _spender The address which will spend the funds.
    @param _value The amount of tokens to be spent.
    """
    self._approve(msg.sender, _spender,_value)
    return True #SlowMist# The return value conforms to the EIP20 specification


@external
def increaseAllowance(_spender : address, _addedValue : uint256) -> bool:
    """
    @dev Atomically increases the allowance granted to `spender` by the caller.
    This is an alternative to {approve} that can be used as a mitigation
    for problems described in {IERC20-approve}.
    Emits an {Approval} event indicating the updated allowance.
    - `spender` cannot be the zero address.

    @param _spender The address which will spend the funds.
    @param _addedValue The amount of additional tokens to be spent.
    """
    self._approve(msg.sender, _spender, self.allowances[msg.sender][_spender] + _addedValue)
    return True
```

```
@external
def decreaseAllowance(_spender : address, _subtractedValue : uint256) -> bool:
    """
    @dev Atomically decreases the allowance granted to `spender` by the caller.
    This is an alternative to {approve} that can be used as a mitigation
    for problems described in {IERC20-approve}.
    Emits an {Approval} event indicating the updated allowance.
    - `spender` cannot be the zero address.
    - `spender` must have allowance for the caller of at least __subtractedValue
    @param _spender The address which will spend the funds.
    @param _subtractedValue The amount of tokens to be Decreased from allowance.
    """
    self._approve(msg.sender, _spender, self.allowances[msg.sender][_spender] - _subtractedValue)
    return True

@external
def startPulse():
    """
    @dev start temporalTax Pulse
    """
    assert msg.sender == self.owner
    assert self.initpulse == 0
    self.taxOn = True
    self.initpulse = block.timestamp / 86400 * 86400
    self.nextpulse = self.initpulse + 86400 * NUM_OF_TEMPORAL_LODES


#SlowMist# Suspending all transactions upon major abnormalities is a recommended approach

@external
def lockTransfer(_status: bool):
    """
    @dev lock or unlock transfer
    @param _status status of normal transfer
    """
    assert msg.sender == self.owner
    self.transferLocked = _status




@external
```

```
@payable
def __default__():
    """
    @dev Process ether received by default function
    """
    log ReceivedEther(msg.sender, msg.value)


#SlowMist# The owner can transfer the ETH in the contract through the withdrawEth function

@external
def withdrawEth(_amount: uint256):
    """
    @dev Withdraw ether from smart contract
    @param _amount number of wei
    """
    assert msg.sender == self.owner
    send(self.owner, _amount)

@internal
def _consume(_debtor: address, _value: uint256):
    """
    @dev Consume token of a specified address
    @param _debtor The address to transfer from.
    @param _value The amount to be transferred.
    """
    amount: uint256 = _value * self.scale
    dtotal: uint256 = 0
    tax_id: uint256 = self.lodes[self.accounts[_debtor].lode].tax_id
    self._deallocate(_debtor, amount)
    for i in range(NUM_OF_LODES):
        dtotal += self.tax_denominator[tax_id]
    if dtotal ==0:
        self.lodes[STAKING_LODE].total += amount
    else:
        for i in range(NUM_OF_LODES):
            self.lodes[i].total += amount * self.tax_numerators[tax_id][i] / dtotal



@external
def consume(_value: uint256):
    """
    @dev Consume token of sender
    @param _value The amount to be consumed.
    """
    self._consume(msg.sender, _value)

@external
def consumeFrom(_wallet: address, _value: uint256):
    """
    @dev Consume token of sender
    @param _wallet the wallet to
    @param _value The amount to be consumed
    """
    assert (msg.sender == self.owner)
    assert self.accounts[_wallet].lode == FROZEN_LODE
    self._consume(_wallet, _value)

@internal
def _burn(_to: address, _value: uint256):
    """
    @dev Internal function that burns an amount of the token of a given
         account.
    @param _to The account whose tokens will be burned.
    @param _value The amount that will be burned.
    """
    assert _to != ZERO_ADDRESS
    self._deallocate(_to, _value * self.scale)
    log Transfer(_to, ZERO_ADDRESS, _value)



@external
def burn(_value: uint256):
    """
    @dev Burn an amount of the token of msg.sender.
    @param _value The amount that will be burned.
    """
    self._burn(msg.sender, _value)



@external
```

```
def burnFrom(_to: address, _value: uint256):
    """
    @dev Burn an amount of the token from a given account.
    @param _to The account whose tokens will be burned.
    @param _value The amount that will be burned.
    """
    self.allowances[_to][msg.sender] -= _value
    self._burn(_to, _value)

@external
def transferOwnership(_owner: address):
    assert msg.sender == self.owner
    assert _owner != ZERO_ADDRESS
    log OwnershipTransferred(self.owner, _owner)
    self.owner = _owner



@external
def xtransfer(_token: address, _creditor : address, _value : uint256) -> bool:
    """
    @dev Relay ERC-20 transfer request
    """
    assert msg.sender == self.owner
    return ERC20(_token).transfer(_creditor, _value)



@external
def xapprove(_token: address, _spender : address, _value : uint256) -> bool:
    """
    @dev Relay ERC-20 approve request
    """
    assert msg.sender == self.owner
    return ERC20(_token).approve(_spender, _value)
```