JohnWick Sec

Guard the internet of value

ETH SMART CONTRACT AUDIT REPORT

JOHNWICK SECURITY LAB

WWW.JOHNWICK.IO

John Wick Security Lab received the **Lendefi** (company/team) **LendefiTokenVesting** project smart contract code audit requirements on **2021/02/05.**

Project Name：Lendefi(LendefiTokenVesting)

Smart Contract Address：

https://etherscan.io/address/0xce8996314f80200974bba394caa19fc1d41225f9#code

Audit Number：20210206

Audit Date：20210206

Audit Category and Result:

Category	Sub-category	Result(Pass/Not Pass)
Contract vulnerability	Integer overflow	Pass
	Race condition	Pass
	Denial of service	Pass
	Logical vulnerability	Pass
	Hardcoded address	Pass
	Function input parameter check	Not Pass
	Function access control bypass	Pass
	Random number generation	Pass
	Random number use	Pass
Contract specification	Solidity compiler version	Pass
	Event use	Pass
	fallback function use	Pass
	Constructor use	Pass
	Function visibility declaration	Pass
	Variable storage declaration	Pass
	Deprecated keyword use	Pass
	ERC20/223 standard	Pass
	ERC721 standard	Pass
Business risk	Able to arbitrarily create token	Pass
	Able to arbitrarily destroy token	Pass
	Able to arbitrarily suspend tx.	Pass
	"Short address" attack	Pass
	"Fake recharge" attack	Pass
GAS optimization	assert()/require()	Pass
	Loop(for/while) optimization	Pass
	Storage optimization	Pass
Automated fuzzing		Pass

(Other unknown security vulnerabilities and Ethereum design flaws are not included in

this audit responsibility)

Audit Result：PASS

Auditor：John Wick Security Lab

(**Disclaimer:** The John Wick Security Lab issues this report based on the facts that have occurred or existed before the issuance of this report and assumes corresponding responsibility in this regard. For the facts that occur or exist after the issuance of this report, the John Wick Security Lab cannot judge the security status of its smart contracts and does not assume any responsibility for it. The safety audit analysis and other contents of this report are based on the relevant materials and documents provided by the information provider to the John Wick Security Lab when the report is issued (referred to as the information provided). The John Wick Security Lab assumes that there is no missing, falsified, deleted, or concealed information provided. If the information provided is missing, falsified, deleted, concealed, or the information provider's response is inconsistent with the actual situation, the John Wick Security Lab shall not bear any responsibility for the resulting loss and adverse effects.)

Audit Details：

//JohnWick:

```
521: contract LendefiTokenVesting is Ownable {
522:     using SafeMath for uint256;
523:     using SafeERC20 for IERC20;
```

The contract uses the `SafeMath` function library to handle potential integer overflows, which is in line with recommended practices.

//JohnWick: **[Low Risk]**

```
638:     function removeVesting(uint256 _vestingId) public onlyOwner {
639:         Vesting storage vesting = vestings[_vestingId];
640:         require(beneficiaryAddress[vesting.beneficiaryIndex] != address(0x0), INVALID_VESTING_ID);
641:         require(!vesting.released , VESTING_ALREADY_RELEASED);
642:         vesting.released = true;
643:         tokensToVest = tokensToVest.sub(vesting.amount);
644:         emit TokenVestingRemoved(_vestingId, beneficiaryAddress[vesting.beneficiaryIndex], vesting.amount);
645:     }
```

`removeVesting(uint256 _vestingId) public onlyOwner` function does not check the `_vestingId` parameter. If `_vestingId` is greater than `vestingId`, this function can also be executed successfully, and set the non-existent `vesting.released` to `true`.

```
673:     function release(uint256 _vestingId) public {
```

`release(uint256 _vestingId) public` has the same problem as above.

Although this will not cause serious problems, we still recommend checking the validity of `_vestingId`.

Note: The line number of the code involved in the audit details is based on the verified contract source code uploaded by the project party at etherscan.io, which is also displayed as a backup in the `Smart Contract Source Code` section of this report.

Smart Contract Source Code:

```
/**
 *Submitted for verification at Etherscan.io on 2021-01-27
*/

pragma solidity 0.6.2;
// SPDX-License-Identifier: MIT



/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending
and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode
- see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

contract Ownable is Context {
```

```
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed
newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
```

```
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero
address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with
custom message on
     * overflow (when the result is negative).
     *
```

```
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero,
but the
        // benefit is lost if 'b' is also tested.
        //                                                                  See:
https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
```

```
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with
custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't
hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
```

```
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns
```

```
(bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns
(uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: 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
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
```

```
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256
value);
}

// SPDX-License-Identifier: MIT
/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */

// SPDX-License-Identifier: MIT
/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */

// SPDX-License-Identifier: MIT
/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
```

```
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created
accounts
        //                                                                  and
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470       is
returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32                          accountHash                          =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     *
https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer
-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     *
https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use
-the-checks-effects-interactions-pattern[checks-effects-interactions
pattern].
     */
```

```
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient
balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have
reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     *                                                                    use
https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?h
ighlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns
(bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data, string memory
errorMessage) internal returns (bytes memory) {
        return _functionCallWithValue(target, data, 0, errorMessage);
```

```
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

    function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
```

```
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory
via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token,
abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256
value) internal {
        _callOptionalReturn(token,
abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20 token, address spender, uint256 value) internal
{
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
```

```
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token,
abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value)
internal {
        uint256      newAllowance      =      token.allowance(address(this),
spender).add(value);
        _callOptionalReturn(token,
abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value)
internal {
        uint256      newAllowance      =      token.allowance(address(this),
spender).sub(value, "SafeERC20: decreased allowance below zero");
        _callOptionalReturn(token,
abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call
to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned,
it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return
data size checking mechanism, since
        // we're implementing it ourselves. We use {Address.functionCall} to
perform this call, which verifies that
        // the target address contains contract code and also asserts for success
in the low-level call.

        bytes memory returndata = address(token).functionCall(data, "SafeERC20:
low-level call failed");
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata,    (bool)),    "SafeERC20:    ERC20
operation did not succeed");
```

```
        }
    }
}

/**
 * @title Lendefi TokenVesting
 * @dev A token holder contract that can release its token balance gradually
like a
 * typical vesting scheme, with a cliff and vesting period. Optionally revocable
by the
 * owner.
 */
contract LendefiTokenVesting is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 private tokenAddress;
    uint256 private tokensToVest = 0;
    uint256 private vestingId = 0;

    address[2] public beneficiaryAddress;

    string private constant INSUFFICIENT_BALANCE = "Insufficient balance";
    string private constant INVALID_VESTING_ID = "Invalid vesting id";
    string private constant VESTING_ALREADY_RELEASED = "Vesting already
released";
    string private constant INVALID_BENEFICIARY = "Invalid beneficiary
address";
    string private constant NOT_VESTED = "Tokens have not vested yet";

    struct Vesting {
        uint256 releaseTime;
        uint256 amount;
        uint256 beneficiaryIndex;
        bool released;
    }
    mapping(uint256 => Vesting) public vestings;

    event TokenVestingReleased(uint256 indexed vestingId, address indexed
beneficiary, uint256 amount);
    event TokenVestingAdded(uint256 indexed vestingId, address indexed
beneficiary, uint256 amount);
    event TokenVestingRemoved(uint256 indexed vestingId, address indexed
beneficiary, uint256 amount);
```

```
    constructor(IERC20 _token) public {
        require(address(_token) != address(0x0), "Lendefi token address is not
valid");
        tokenAddress = _token;

        uint256 SCALING_FACTOR = 10 ** 18; // decimals
        uint256 day = 1 days;

        beneficiaryAddress[0] = msg.sender;
        beneficiaryAddress[1] = msg.sender;

        addVesting(0, now + 30 * day, 1200000*SCALING_FACTOR);
        addVesting(1, now + 30 * day,  252083*SCALING_FACTOR); // 1452083

        addVesting(0, now + 60 * day,  200000*SCALING_FACTOR);
        addVesting(1, now + 60 * day,  377083*SCALING_FACTOR); // 577083

        addVesting(0, now + 90 * day,  200000*SCALING_FACTOR);
        addVesting(1, now + 90 * day,  252083*SCALING_FACTOR); // 452083

        addVesting(0, now + 120 * day, 200000*SCALING_FACTOR);
        addVesting(1, now + 120 * day, 252083*SCALING_FACTOR); // 452083

        addVesting(1, now + 150 * day, 377083*SCALING_FACTOR);
        addVesting(1, now + 180 * day, 252083*SCALING_FACTOR);
        addVesting(1, now + 210 * day, 252083*SCALING_FACTOR);
        addVesting(1, now + 240 * day, 377083*SCALING_FACTOR);
        addVesting(1, now + 270 * day, 252083*SCALING_FACTOR);
        addVesting(1, now + 300 * day, 252083*SCALING_FACTOR);
        addVesting(1, now + 330 * day, 377083*SCALING_FACTOR);
        addVesting(1, now + 360 * day, 252083*SCALING_FACTOR);
        addVesting(1, now + 390 * day, 168750*SCALING_FACTOR);
        addVesting(1, now + 420 * day, 293750*SCALING_FACTOR);
        addVesting(1, now + 450 * day, 168750*SCALING_FACTOR);
        addVesting(1, now + 480 * day, 168750*SCALING_FACTOR);
        addVesting(1, now + 510 * day, 293750*SCALING_FACTOR);
        addVesting(1, now + 540 * day, 168750*SCALING_FACTOR);
        addVesting(1, now + 570 * day, 168750*SCALING_FACTOR);
        addVesting(1, now + 600 * day, 293750*SCALING_FACTOR);
        addVesting(1, now + 630 * day, 168750*SCALING_FACTOR);
        addVesting(1, now + 660 * day, 168750*SCALING_FACTOR);
        addVesting(1, now + 690 * day, 293750*SCALING_FACTOR);
        addVesting(1, now + 720 * day, 168750*SCALING_FACTOR);
    }
```

```
    function updateBeneficiaryAddress(uint index, address newAddr) external {
        require(newAddr != address(0x0), INVALID_BENEFICIARY);
        require(beneficiaryAddress[index] == msg.sender, "No Access");
        beneficiaryAddress[index] = newAddr;
    }

    /**
     * @dev Token address for vesting contract
     * @return Token contract address
     */
    function token() public view returns (IERC20) {
        return tokenAddress;
    }

    /**
     * @dev Function to check beneficiary of a vesting
     * @param _vestingId  vesting Id
     * @return Beneficiary's address
     */
    function beneficiary(uint256 _vestingId) public view returns (address) {
        return beneficiaryAddress[vestings[_vestingId].beneficiaryIndex];
    }

    /**
     * @dev Function to check Release Time of a vesting
     * @param _vestingId  vesting Id
     * @return Release Time in unix timestamp
     */
    function releaseTime(uint256 _vestingId) public view returns (uint256) {
        return vestings[_vestingId].releaseTime;
    }

    /**
     * @dev Function to check token amount of a vesting
     * @param _vestingId  vesting Id
     * @return Number of tokens for a vesting
     */
    function vestingAmount(uint256 _vestingId) public view returns (uint256)
{
        return vestings[_vestingId].amount;
    }

    /**
     * @notice Function to remove a vesting by owner of vesting contract
     * @param _vestingId  vesting Id
```

```
    */
    function removeVesting(uint256 _vestingId) public onlyOwner {
        Vesting storage vesting = vestings[_vestingId];
        require(beneficiaryAddress[vesting.beneficiaryIndex] != address(0),
INVALID_VESTING_ID);
        require(!vesting.released , VESTING_ALREADY_RELEASED);
        vesting.released = true;
        tokensToVest = tokensToVest.sub(vesting.amount);
        emit                                    TokenVestingRemoved(_vestingId,
beneficiaryAddress[vesting.beneficiaryIndex], vesting.amount);
    }

    /**
     * @notice Function to add a vesting
     * Since this is onlyOwner protected, tokens are assumed to be transferred
to the vesting contract
     * @param _beneficiaryIndex  Beneficiary's address index
     * @param _releaseTime  Time for release
     * @param _amount       Amount of vesting
     */
    function addVesting(uint _beneficiaryIndex, uint256 _releaseTime, uint256
_amount) public onlyOwner {
        require(beneficiaryAddress[_beneficiaryIndex]    !=    address(0),
INVALID_BENEFICIARY);
        require(_releaseTime > now, "Invalid release time");
        require(_amount != 0, "Amount must be greater then 0");
        tokensToVest = tokensToVest.add(_amount);
        vestingId = vestingId.add(1);
        vestings[vestingId] = Vesting({
            beneficiaryIndex: _beneficiaryIndex,
            releaseTime: _releaseTime,
            amount: _amount,
            released: false
        });
        emit                                        TokenVestingAdded(vestingId,
beneficiaryAddress[_beneficiaryIndex], _amount);
    }

    /**
     * @notice Function to release tokens of a vesting id
     * @param _vestingId  vesting Id
     */
    function release(uint256 _vestingId) public {
        Vesting storage vesting = vestings[_vestingId];
        require(beneficiaryAddress[vesting.beneficiaryIndex] != address(0),
```

```
INVALID_VESTING_ID);
        require(!vesting.released , VESTING_ALREADY_RELEASED);
        // solhint-disable-next-line not-rely-on-time
        require(block.timestamp >= vesting.releaseTime, NOT_VESTED);

        require(tokenAddress.balanceOf(address(this))   >=   vesting.amount,
INSUFFICIENT_BALANCE);
        vesting.released = true;
        tokensToVest = tokensToVest.sub(vesting.amount);

tokenAddress.safeTransfer(beneficiaryAddress[vesting.beneficiaryIndex],
vesting.amount);
        emit                                TokenVestingReleased(_vestingId,
beneficiaryAddress[vesting.beneficiaryIndex], vesting.amount);
    }

    /**
     * @dev Function to remove any extra tokens, i.e cancelation of a vesting
     * @param _amount Amount to retrieve
     */
    function retrieveExcessTokens(uint256 _amount) public onlyOwner {
        require(_amount                                                    <=
tokenAddress.balanceOf(address(this)).sub(tokensToVest),
INSUFFICIENT_BALANCE);
        tokenAddress.safeTransfer(owner(), _amount);
    }
}
```