its__msn itsmsn

ITISH

AUDIT COMPANY

Audit Details

Contract Name
Leonard

Deployer address

0x9E3078B70bb90a46536bCb47157bbDcD885E874f

Client contacts:

Leonard team

Blockchain

Binance

Project website:

Not Provided By contract

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Itish and its affiliates (includingholding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Itish) owe no duty of care towards you or any other person, nor does Itish make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Itish hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Itish hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Itish, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Background

Itish was commissioned PWLC token to perform an audit of smart contracts:

https://bscscan.com/address/0x9E3078B70bb90a46536bCb47157bbDcD885E874f

The purpose of the audit was to achieve the following:

- **Ensure that the smart contract functions as intended.**
- **Identify potential security issues with the smart contract.**

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

Contract Details

Token contract details for 15.12.2022

contract name	**Leonard**
Contract creator	0x9E3078B70bb90a46536bCb47157bbDcD885E874f
Transaction's count	**17**

Contract TopTransactions

Transactions | BEP-20 Token Txns | Contract | Events | Analytics | Comments

Latest 17 from a total of 17 transactions

Txn Hash	Method	Block	Age	From		To	Value	[Txn Fee]
0xa9f04a786b4b9fb6947...	Transfer	23924165	6 hrs 32 mins ago	0x445809cdcc50eeacf83...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0x0c530f8eec9340c8421...	Approve	23892743	1 day 9 hrs ago	0xfe3c22654d545c11622...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0x8cbfb16e0d3a147319...	Transfer	23892641	1 day 9 hrs ago	0x2aad546f684bb37dc2...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0x15bee3fc7890051226...	Transfer	23868910	1 day 12 hrs ago	0x445809cdcc50eeacf83...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0x8157991eb333add87f...	Transfer	23866278	2 days 7 hrs ago	0x445809cdcc50eeacf83...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0xc856d94c813fdbae64...	Transfer	23703115	8 days 2 hrs ago	0xa91e48477a91762315...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0x1434058219b8f5c83fa...	Transfer	23703033	8 days 2 hrs ago	0x445809cdcc50eeacf83...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0xe92a0ca736583c6778...	Transfer	23609758	11 days 10 hrs ago	0x445809cdcc50eeacf83...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0x96c0f98e26f2116b456...	Approve	23468033	16 days 10 hrs ago	0x70597f940ac3d3e223...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0x74f5415e20e1a4d64a...	Transfer	23302536	22 days 6 hrs ago	0xce54d84ae4b24fd38c...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0x63389f25c2f3d5abd4a...	Transfer	23302447	22 days 6 hrs ago	0x70597f940ac3d3e223...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0xceb117946e8fae2657...	Transfer	23302373	22 days 6 hrs ago	0x70597f940ac3d3e223...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0x7103c678e6b93d555...	Transfer	23302156	22 days 6 hrs ago	0x70597f940ac3d3e223...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]
0x62c15e00a8ee926665...	Transfer	23302140	22 days 6 hrs ago	0x70597f940ac3d3e223...	IN	0x9e3078b70bb90a4653...	0 BNB	[illegible]

Token Functions Details

```
totalSupply()
decimals()
symbol()
name()
getOwner()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
_msgData()
renounceOwnership()
```

Contract Interface Details

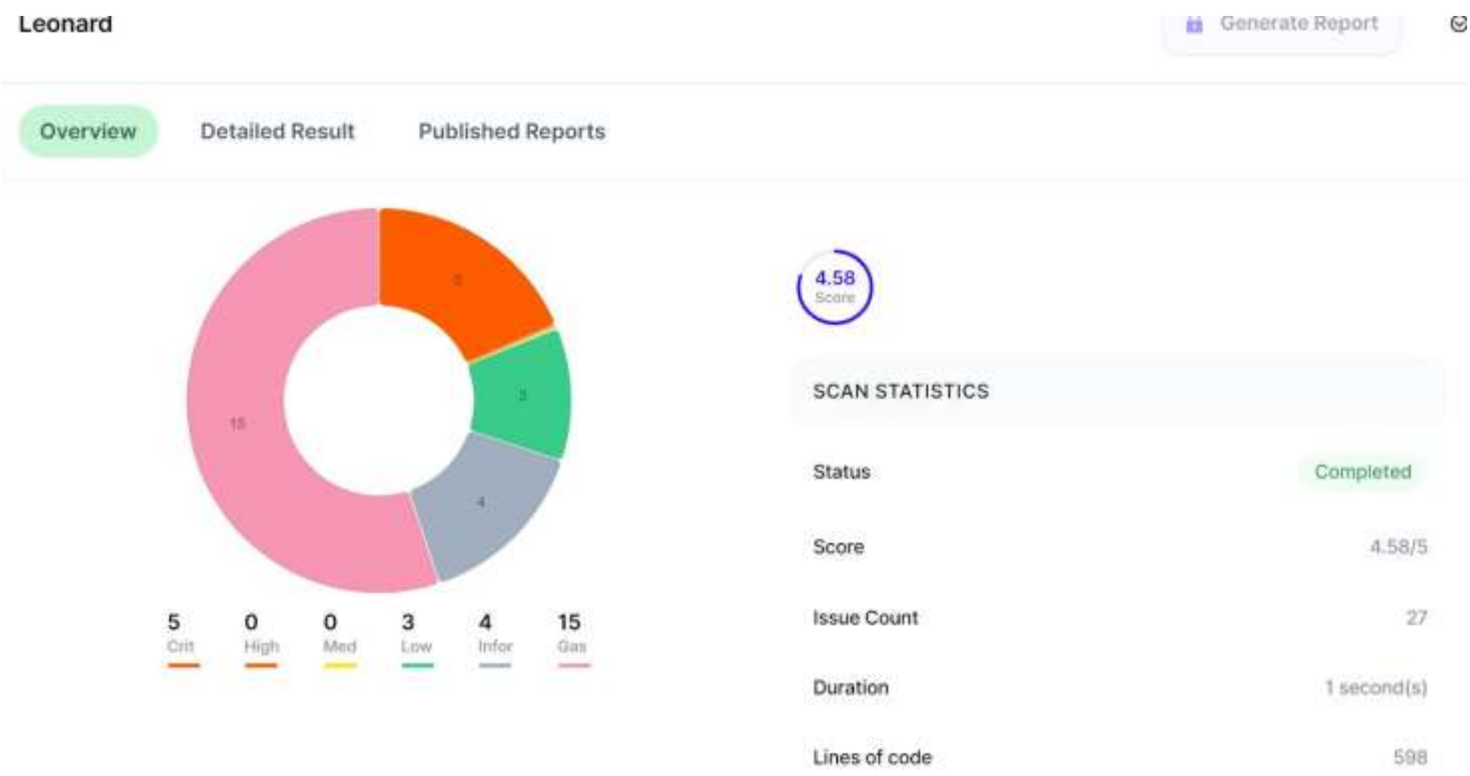
```
interface IERC20
interface IERC20Metadata is IERC20
```

Issues Checking Status

Issue description	checking status
1. Compiler errors.	Passed
2. Compiler Compatibilities	failed
3. Possible delays in data delivery.	Passed
4. Oracle calls.	Moderate
5. Front running.	Passed
6. Timestamp dependence.	Passed
7. Integer Overflow and Underflow.	Passed
8. DoS with Revert.	Severe
9. DoS with block gas limit.	Moderate
10. Methods execution permissions.	Passed
11. Economy model of the contract.	Passed
12. The impact of the exchange rate on the logic.	Severe
13. Private user data leaks.	Passed
14. Malicious Event log.	Passed
15. Scoping and Declarations.	Passed
16. Uninitialized storage pointers.	Passed
17. Arithmetic accuracy.	passed
18. Design Logic.	poor

19.	**Cross-function race conditions.**	**Passed**
20.	Safe Open Zeppelin contracts implementation and usage.	pass
21.	**Fallback function security.**	**Passed**

Security Issues



Critical Security Issues

Issue # 1:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `Leonard` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `transfer` is missing the modifier `onlyOwner`.

```
   * @dev See {BEP20-transfer}.
   *
   * Requirements:
   *
   * - `recipient` cannot be the zero address.
   * - the caller must have a balance of at least `amount`.
   */
  function transfer(address recipient, uint256 amount) exte
    _transfer(_msgSender(), recipient, amount);
    return true;
  }

  /**
   * @dev See {BEP20-allowance}.
   */
```

Remediation # 1:

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Type 2:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `Leonard` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `approve` is missing the modifier `onlyOwner`.

```
/**
 * @dev See {BEP20-approve}.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function approve(address spender, uint256 amount) externa
  _approve(_msgSender(), spender, amount);
  return true;
}

/**
 * @dev See {BEP20-transferFrom}.
 *
```

Remediation # 1:

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Type 3:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `Leonard` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `transferFrom` is missing the modifier `onlyOwner`.

contract.sol

```
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for `sender`'s tokens
     * `amount`.
     */
    function transferFrom(address sender, address recipient,
      _transfer(sender, recipient, amount);
      _approve(sender, _msgSender(), _allowances[sender][_msg
      return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `sp
```

Remediation # 1:

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Type 4:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `Leonard` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `increaseAllowance` is missing the modifier `onlyOwner`.

```
   *
   * Emits an {Approval} event indicating the updated allow
   *
   * Requirements:
   *
   * - `spender` cannot be the zero address.
   */
  function increaseAllowance(address spender, uint256 added
    _approve(_msgSender(), spender, _allowances[_msgSender(
    return true;
  }

  /**
   * @dev Atomically decreases the allowance granted to `sp
   *
```

Remediation # 1:

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Type 5:

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract `PWLC` is importing an access control library `@openzeppelin/contracts/access/Ownable.sol` but the function `decreaseAllowance` is missing the modifier `onlyOwner`.

```
   *
   * Requirements:
   *
   * - `spender` cannot be the zero address.
   * - `spender` must have allowance for the caller of at l
   * `subtractedValue`.
   */
  function decreaseAllowance(address spender, uint256 subtr
    _approve(_msgSender(), spender, _allowances[_msgSender(
    return true;
  }

  /**
   * @dev Creates `amount` tokens and assigns them to `msg.
   * the total supply.
```

Remediation # 1:

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Low Severity Issues

Issue # 1:

LONG NUMBER LITERALS

Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different notation also supported by Solidity.
The value `5400000000000000000000` was detected on line `357`.

```
string private _symbol;
string private _name;

constructor() public {
  _name = "Leonard Token";
  _symbol = "LEONARD";
  _decimals = 18;
  _totalSupply = 54000000000000000000000;
  _balances[msg.sender] = _totalSupply;

  emit Transfer(address(0), msg.sender, _totalSupply);
}

/**
 * @dev Returns the bep token owner.
```

Remediation # 1:

Scientific notation in the form of `2e10` is also supported, where the mantissa can be fractional but the exponent has to be an integer. The literal `MeE` is equivalent to `M * 10**E`. Examples include `2e10`, `2e10`, `2e-10`, `2.5e1`, as suggested in official solidity documentation `https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals`

Issue # 2:

USE OF FLOATING PRAGMA

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version.
The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.
The following affected files were found to be using floating pragma:
`contract.sol - ^0.5.0`

```
// SPDX-License-Identifier: MIT
// Enable optimization
pragma solidity ^0.5.0;

interface IBEP20 {
  /**
   * @dev Returns the amount of tokens in existence.
   */
  function totalSupply() external view returns (uint256);

   /**
    * @dev Returns the token decimals.
    */
   function decimals() external view returns (uint8);

```

Remediation # 2:

It is recommended to use a fixed pragma version, as future compiler versions may handle certain language constructions in a way the developer did not foresee.
Using a floating pragma may introduce several vulnerabilities if compiled with an older version.
The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.
Instead of `^0.5.0` use `pragma solidity 0.8.7`, which is a stable and recommended version right now.

Issue # 3:

OUTDATED COMPILER VERSION

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.
The following outdated versions were detected:
`contract.sol` - `^0.5.0`

```
// SPDX-License-Identifier: MIT
// Enable optimization
pragma solidity ^0.5.0;

interface IBEP20 {
  /**
   * @dev Returns the amount of tokens in existence.
   */
  function totalSupply() external view returns (uint256);

   /**
    * @dev Returns the token decimals.
    */
   function decimals() external view returns (uint8);

```

Remediation # 3:

It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well. Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks. Consider using the solidity version `0.8.7`, which patches most solidity vulnerabilities.

Informative

1)

PRESENCE OF OVERPOWERED ROLE

The overpowered owner (i.e., the person who has too much power) is a project design where the contract is tightly coupled to their owner (or owners); only they can manually invoke critical functions.
Due to the fact that this function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.

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

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

Remediation # 1:

We recommend designing contracts in a trust-less manner. For instance, this functionality can be implemented in the contract's constructor. Another option is to use a MultiSig wallet for this address. For systems that are provisioned for a single user, you can use [Ownable.sol].
For systems that require provisioning users in a group, you can use [@openzeppelin/Roles.sol] or [@hq20/Whitelist.sol].

Type 2

PRESENCE OF OVERPOWERED ROLE

The overpowered owner (i.e., the person who has too much power) is a project design where the contract is tightly coupled to their owner (or owners); only they can manually invoke critical functions.
Due to the fact that this function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.

```
    _owner = address(0);
  }

  /**
   * @dev Transfers ownership of the contract to a new acco
   * Can only be called by the current owner.
   */
  function transferOwnership(address newOwner) public onlyO
    _transferOwnership(newOwner);
  }

  /**
   * @dev Transfers ownership of the contract to a new acco
   */
  function _transferOwnership(address newOwner) internal {
```

Remediation # 2:

We recommend designing contracts in a trust-less manner. For instance, this functionality can be implemented in the contract's constructor. Another option is to use a MultiSig wallet for this address. For systems that are provisioned for a single user, you can use [Ownable.sol].
For systems that require provisioning users in a group, you can use [@openzeppelin/Roles.sol] or [@hq20/Whitelist.sol].

GAS

1)

USE OF SAFEMATH LIBRARY

`SafeMath` library is found to be used in the contract. This increases gas consumption than traditional methods and validations if done manually.
Also, Solidity `0.8.0` includes checked arithmetic operations by default, and this renders `SafeMath` unnecessary.

```
    require(newOwner != address(0), "Ownable: new owner is
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
  }
}

contract Leonard is Context, IBEP20, Ownable {
  using SafeMath for uint256;

  mapping (address => uint256) private _balances;

  mapping (address => mapping (address => uint256)) private

  uint256 private _totalSupply;
  uint8 private _decimals;
```

Remediation # 1:

We do not recommend using `SafeMath` library for all arithmetic operations. It is good practice to use explicit checks where it is really needed and to avoid extra checks where overflow/underflow is impossible.
The compiler should be upgraded to Solidity version `0.8.0+` which automatically checks for overflows and underflows.

2)

CHEAPER INEQUALITIES IN REQUIRE()

The contract was found to be performing comparisons using inequalities inside the `require` statement. When inside the `require` statements, non-strict inequalities `(>=, <=)` are usually costlier than strict equalities `(>, <)`.

```
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");

      return c;
    }

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

Remediation # 2:

It is recommended to go through the code logic, and, if possible, modify the non-strict inequalities with the strict ones to save `~3` gas as long as the logic of the code is not affected.

3)

FUNCTION SHOULD BE EXTERNAL

A function with `public` visibility modifier was detected that is not called internally.
`public` and `external` differs in terms of gas usage. The former use more than the latter when used with large arrays of data. This is due to the fact that Solidity copies arguments to memory on a `public` function while `external` read from calldata which a cheaper than memory allocation.

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

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

Remediation # 3:

If you know the function you create only allows for `external` calls, use the `external` visibility modifier instead of `public`. It provides performance benefits and you will save on gas.

4)

LONG REQUIRE/REVERT STRINGS

The `require()` and `revert()` functions take an input string to show errors if the validation fails. This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

```
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-co
    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 inte
   * division by zero. The result is rounded towards zero.
```

Remediation # 4:

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Conclusion

Smart contracts contain High severity issues! Liquiditypair contract’s security is not checked due to out of scope.

Liquidity locking details NOT provided by the team.

Itish note:

Please check the disclaimer above and note, the audit makes no statements or warranties on business model, investment attractiveness or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by Owner.