Armors Labs

TKL Token

Smart Contract Audit

TKL Token Audit Summary

Project name : TKL Token Contract

Project address: None

Code URL : https://www.bscscan.com/address/0x1e37a6945af939116dd19a49686b259307181074#code

Commit : None

Project target : TKL Token Contract Audit

Blockchain : Binance Smart Chain（BSC）

Test result : PASSED

Audit Info

Audit NO : 0X202112070026

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

TKL Token Audit

The TKL Token team asked us to review and audit their TKL Token contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
TKL Token Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-12-07

Audit results

Notices:

```
1 the administrator can start or stop contract transfer.

2. 10% of each authorized transfer of non white list users will be permanently destroyed,
   of which 5% will be transferred to liquiditypool and 5% will be directly destroyed.
```

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the TKL Token contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
TKL Token.sol	ffec636764d8753ffe4e2139c8f29369

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe

Vulnerability	status
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

```
/**
 *Submitted for verification at BscScan.com on 2021-12-07
*/

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC165 standard, as defined in the
 * https://eips.ethereum.org/EIPS/eip-165[EIP].
 *
 * Implementers can declare support of contract interfaces, which can then be
 * queried by others ({ERC165Checker}).
 *
 * For an implementation, see {ERC165}.
 */
interface IERC165 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * https://eips.ethereum.org/EIPS/eip-165#how-interfaces-are-identified[EIP section]
     * to learn more about how these ids are created.
     *
     * This function call must use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}

// File: @openzeppelin/contracts/utils/introspection/ERC165.sol
// OpenZeppelin Contracts v4.4.0 (utils/introspection/ERC165.sol)

pragma solidity ^0.8.0;


            /**
 * @dev Implementation of          the          {IERC165} interface.
 *
 * Contracts that want to implement ERC165          should          inherit from this contract and overri
 * for          the          additional interface id that          will          be supported. For
 *
 * solidity
 * function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
 *    return interfaceId == type(MyInterface).interfaceId || super.supportsInterface(interfaceId);
 * }
 *
*
* Alternatively, {ERC165Storage} provides          an          easier to use          but
 */
  abstract contract ERC165 is IERC165 {
            /**
  * @dev See {IERC165-supportsInterface}.
   */
      function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
      return interfaceId == type(IERC165).interfaceId;
      }
      }

// File: @openzeppelin/contracts/utils/Strings.sol


// OpenZeppelin Contracts v4.4.0 (utils/Strings.sol)

pragma solidity ^0.8.0;

            /**
* @dev String operations.
 */
  library Strings {
  bytes16 private constant _HEX_SYMBOLS = "0123456789abcdef";

            /**
  * @dev Converts          a          `uint256` to its ASCII `string` decimal representation.
   */
      function toString(uint256 value) internal pure returns (string memory) {
      // Inspired by OraclizeAPI's implementation - MIT licence
      // https://github.com/oraclize/ethereum-api/blob/b42146b063c7d6ee1358846c198246239e9360e8/oracl

      if (value == 0) {
      return "0";
      }
      uint256 temp = value;
      uint256 digits;
      while (temp != 0) {
      digits++;
      temp /= 10;
      }
      bytes memory buffer = new bytes(digits);
      while (value != 0) {
```

```
      digits -= 1;
      buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
      value /= 10;
      }
      return string(buffer);
      }

              /**
  * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation.
   */
      function toHexString(uint256 value) internal pure returns (string memory) {
      if (value == 0) {
      return "0x00";
      }
      uint256 temp = value;
      uint256 length = 0;
      while (temp != 0) {
      length++;
      temp >>= 8;
      }
      return toHexString(value, length);
      }

              /**
  * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation with fixe
   */
      function toHexString(uint256 value, uint256 length) internal pure returns (string memory) {
      bytes memory buffer = new bytes(2 * length + 2);
      buffer[0] = "0";
      buffer[1] = "x";
      for (uint256 i = 2 * length + 1; i > 1; --i) {
      buffer[i] = _HEX_SYMBOLS[value & 0xf];
      value >>= 4;
      }
      require(value == 0, "Strings: hex length insufficient");
      return string(buffer);
      }
      }

// File: @openzeppelin/contracts/access/IAccessControl.sol


// OpenZeppelin Contracts v4.4.0 (access/IAccessControl.sol)

pragma solidity ^0.8.0;

            /**
* @dev External interface of AccessControl declared to support ERC165 detection.
 */
  interface IAccessControl {
              /**
  * @dev Emitted when `newAdminRole` is set as ``role``'s admin role, replacing `previousAdminRole`
  *
  * `DEFAULT_ADMIN_ROLE` is the starting admin for all roles, despite
  * {RoleAdminChanged} not being emitted signaling this.
  *
  * _Available since v3.1._
   */
      event RoleAdminChanged(bytes32 indexed role, bytes32 indexed previousAdminRole, bytes32 indexed
              /**
  * @dev Emitted when `account` is granted `role`.
  *
  * `sender` is the account that originated the contract cal
  * bearer except when using {AccessControl-_setupRole}.
   */
      event RoleGranted(bytes32 indexed role, address indexed account, address indexed sender);

              /**
  * @dev Emitted when `account` is revoked `role`.
  *
  * `sender` is the account that originated the contract cal
  *   - if using `revokeRole`, it is the admin role bearer
  *   - if using `renounceRole`, it is the role bearer (i.e. `account`)
    */
      event RoleRevoked(bytes32 indexed role, address indexed account, address indexed sender);

              /**
  * @dev Returns `true` if `account` has been granted `role`.
   */
      function hasRole(bytes32 role, address account) external view returns (bool);

              /**
  * @dev Returns the admin role that controls `role`. See {grantRole} and
  * {revokeRole}.
  *
  * To change a role's admin, use {AccessControl-_setRoleAdmin}.
   */
      function getRoleAdmin(bytes32 role) external view returns (bytes32);

              /**
  * @dev Grants `role` to `account`.
  *
  * If `account` had not been already granted `role`, emits a {RoleGranted}
  * event.
  *
  * Requirements:
  *
  * - the caller must have ``role``'s admin role.
    */
    function grantRole(bytes32 role, address account) external;

              /**
  * @dev Revokes `role` from `account`.
  *
  * If `account` had been granted `role`, emits a {RoleRevoked} event.
  *
  * Requirements:
  *
  * - the caller must have ``role``'s admin role.
    */
    function revokeRole(bytes32 role, address account) external;
  /**
   * @dev Revokes `role` from the calling account.
   *
   * Roles are often managed via {grantRole} and {revokeRole}: this function's
   * purpose is to provide a mechanism for accounts to lose their privileges
   * if they are compromised ( such
   *
   * If the calling account had been granted `role`, emits a {
   * event.
   *
   * Requirements:
   *
   * - the caller must be `account`.
   */
  function renounceRole(bytes32 role, address account) external;
}

// File: @openzeppelin/contracts/utils/Context.sol


// OpenZeppelin Contracts v4.4.0 (utils/Context.sol)

pragma solidity ^0.8.0;

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

function _msgData() internal view virtual returns (bytes calldata) {
return msg.data;
}
}

// File: @openzeppelin/contracts/access/AccessControl.sol


// OpenZeppelin Contracts v4.4.0 (access/AccessControl.sol)

pragma solidity ^0.8.0;




/**
* @dev Contract module that allows children to implement role-based access
 * control mechanisms. This is a lightweight version that doesn't
 * members except through off-chain means by accessing the contract event logs. Some
 * applications may benefit from on-chain enumerability, for those cases see
 * {AccessControlEnumerable}.
 *
 * Roles are referred to by their `bytes32` identifier. These should
 * in the external API and be unique. The best way to achieve this is by
 * using `public constant` hash digests:
 *
 *
 * bytes32 public constant MY_ROLE = keccak256("MY_ROLE");
 *
 *
 * Roles can be used to represent a set of permissions. To restrict access to
 * function call, use {hasRole}:
 *
 *
 * function foo() public {
 *     require(hasRole(MY_ROLE, msg.sender));
 *     ...
 * }
 *
 *
 * Roles can be granted and revoked dynamically via the {grantRole} and
 * {revokeRole} functions. Each role has an associated admin role, and only
 * accounts that have a role's admin role can call {grantRole} and {revokeRole}.
 *
 * By default, the admin role for all roles is `DEFAULT_ADMIN_ROLE`, which means
 * that only accounts with this role will be able to grant or revoke other
 * roles. More complex role relationships can be created by using
 * {_setRoleAdmin}.
 *
 * WARNING: The `DEFAULT_ADMIN_ROLE` is also its own admin: it has permission to
 * grant and revoke this role. Extra precautions should be taken to secure
 * accounts that have been granted it.
 */
abstract contract AccessControl is Context, IAccessControl, ERC165 {
struct RoleData {
mapping(address => bool) members;
bytes32 adminRole;
}

mapping(bytes32 => RoleData) private _roles;

bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

            /**
 * @dev Modifier that checks that an account has a speci
 * with a standardized message including the required ro
 *
 * The format of the revert reason is given by the followi
 *
```

```
* /^AccessControl: account (0x[0-9a-f]{40}) is missing role (0x[0-9a-f]{64})$/
*
* _Available since v4.1._
 */
    modifier onlyRole(bytes32 role) {
    _checkRole(role, _msgSender());
    _;
    }

            /**
* @dev See {IERC165-supportsInterface}.
 */
    function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
    return interfaceId == type(IAccessControl).interfaceId || super.supportsInterface(interfaceId);
    }

            /**
* @dev Returns `true` if `account` has been granted `role`.
 */
    function hasRole(bytes32 role, address account) public view override returns (bool) {
    return _roles[role].members[account];
    }

            /**
* @dev Revert with a standard message if `account` is missing `role`.
*
* The format of the revert reason is given by the followin
*
* /^AccessControl: account (0x[0-9a-f]{40}) is missing role (0x[0-9a-f]{64})$/
 */
     function _checkRole(bytes32 role, address account) internal view {
     if (!hasRole(role, account)) {
     revert(
     string(
     abi.encodePacked(
     "AccessControl: account ",
     Strings.toHexString(uint160(account), 20),
     " is missing role ",
     Strings.toHexString(uint256(role), 32)
     )
     )
     );
     }
     }

            /**
* @dev Returns the admin role that controls `role`. See {grantRole} and
* {revokeRole}.
*
* To change a role's admin, use {_setRoleAdmin}.
 */
    function getRoleAdmin(bytes32 role) public view override returns (bytes32) {
    return _roles[role].adminRole;
    }

            /**
* @dev Grants `role` to `account`.
*
```

```
* If `account` had not been already granted `role`, emits a {RoleGranted}
* event.
*
* Requirements:
*
* - the caller must have ``role``'s admin role.
  */
      function grantRole(bytes32 role, address account) public virtual override onlyRole(getRoleAdm
      _grantRole(role, account);
      }

            /**
* @dev Revokes `role` from `account`.
*
* If `account` had been granted `role`, emits a {RoleRevoked} event.
*
* Requirements:
*
* - the caller must have ``role``'s admin role.
  */
      function revokeRole(bytes32 role, address account) public virtual override onlyRole(getRoleAd
      _revokeRole(role, account);
      }

            /**
* @dev Revokes `role` from the calling account.
*
* Roles are often managed via {grantRole} and {revokeRole}: this function's
* purpose is to provide a mechanism for accounts to lose their privileges
* if they are compromised ( such
*
* If the calling account had been revoked `role`, emits a
* event.
*
* Requirements:
*
* - the caller must be `account`.
  */
      function renounceRole(bytes32 role, address account) public virtual override {
      require(account == _msgSender(), "AccessControl: can only renounce roles for self");

  _revokeRole(role, account);
  }

            /**
* @dev Grants `role` to `account`.
*
* If `account` had not been already granted `role`, emits a {RoleGranted}
* event. Note that unlike {grantRole}, this function doesn't perform any
* checks on the calling account.
*
* [WARNING]
* ====
 * This function should only be called from the constructo
 * up the initial roles for the system.
 *
 * Using this function in any other way is effectively circumventing the admin
 * system imposed by {AccessControl}.
 * ====
 *
 * NOTE: This function is deprecated in favor of {_grantRole}.
  */
    function _setupRole(bytes32 role, address account) internal virtual {
    _grantRole(role, account);
    }

         /**
 * @dev Sets `adminRole` as ``role``'s admin role.
 *
 * Emits a {RoleAdminChanged} event.
  */
    function _setRoleAdmin(bytes32 role, bytes32 adminRole) internal virtual {
    bytes32 previousAdminRole = getRoleAdmin(role);
    _roles[role].adminRole = adminRole;
    emit RoleAdminChanged(role, previousAdminRole, adminRole);
    }

         /**
 * @dev Grants `role` to `account`.
 *
 * Internal function without access restriction.
  */
    function _grantRole(bytes32 role, address account) internal virtual {
    if (!hasRole(role, account)) {
    _roles[role].members[account] = true;
    emit RoleGranted(role, account, _msgSender());
    }
    }

         /**
 * @dev Revokes `role` from `account`.
 *
 * Internal function without access restriction.
  */
    function _revokeRole(bytes32 role, address account) internal virtual {
    if (hasRole(role, account)) {
    _roles[role].members[account] = false;
    emit RoleRevoked(role, account, _msgSender());
    }
    }
    }
// File: @openzeppelin/contracts/security/Pausable.sol


// OpenZeppelin Contracts v4.4.0 (security/Pausable.sol)

pragma solidity ^0.8.0;


            /**
* @dev Contract module which allows children to implement an emergency stop
* mechanism that can be triggered by an authorized account.
*
* This module is used through inheritance. It will make available the
* modifiers `whenNotPaused` and `whenPaused`, which can be applied to
* the functions of your contract. Note that th
* simply including this module, only once the modifiers
 */
 abstract contract Pausable is Context {
         /**
 * @dev Emitted when the pause is triggered by `account`.
  */
    event Paused(address account);

         /**
 * @dev Emitted when the pause is lifted by `account`.
  */
    event Unpaused(address account);

 bool private _paused;

         /**
 * @dev Initializes the contract in unpaused state.
  */
    constructor() {
    _paused = false;
    }

         /**
 * @dev Returns true if the contract is paused, and false otherwise.
  */
    function paused() public view virtual returns (bool) {
    return _paused;
    }

         /**
 * @dev Modifier to make a function callable only when the
 *
 * Requirements:
 *
 * - The contract must not be paused.
   */
       modifier whenNotPaused() {
       require(!paused(), "Pausable: paused");
       _;
       }

         /**
 * @dev Modifier to make a function callable only when the
 *
 * Requirements:
 *
 * - The contract must be paused.
   */
       modifier whenPaused() {
```

```
        require(paused(), "Pausable: not paused");
        _;
        }

              /**
   * @dev Triggers stopped state.
   *
   * Requirements:
   *
   * - The contract must not be paused.
     */
        function _pause() internal virtual whenNotPaused {
        _paused = true;
        emit Paused(_msgSender());
        }

              /**
   * @dev Returns to normal state.
   *
   * Requirements:
   *
   * - The contract must be paused.
     */
        function _unpause() internal virtual whenPaused {
        _paused = false;
        emit Unpaused(_msgSender());
        }
        }

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol


// OpenZeppelin Contracts v4.4.0 (token/ERC20/IERC20.sol)

pragma solidity ^0.8.0;

            /**
 * @dev Interface of the ERC20 standard as defined in the
  */
  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 operat
   *
   * Emits a {Transfer} event.
    */
      function transfer(address recipient, uint256 amount) external returns (bool);

              /**
   * @dev Returns the remaining number of tokens that `spender` will
   * 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
   *
   * Returns a boolean value indicating whether the operat
   *
   * IMPORTANT: Beware that changing an allowance with this method brings
   * that someone may use both the old and the new allow
   * 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 operat
   *
   * 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
```

```
 *   a call to {approve}. `value` is the new allowance.
  */
    event Approval(address indexed owner, address indexed spender, uint256 value);
    }

// File: @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol


// OpenZeppelin Contracts v4.4.0 (token/ERC20/extensions/IERC20Metadata.sol)

pragma solidity ^0.8.0;


/**
 * @dev Interface for the optional metadata functions from the
 *
 * _Available since v4.1._
 */
  interface IERC20Metadata is IERC20 {
    /**
  * @dev Returns the name of the token.
   */
    function name() external view returns (string memory);

    /**
  * @dev Returns the symbol of the token.
   */
    function symbol() external view returns (string memory);

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

// File: @openzeppelin/contracts/token/ERC20/ERC20.sol


// OpenZeppelin Contracts v4.4.0 (token/ERC20/ERC20.sol)

pragma solidity ^0.8.0;




/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are cr
 * that a supply mechanism has to be added in a derived co
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions revert
* instead returning `false` on failure. This behavior is nonetheless
* conventional and does not conflict with the expectations of ERC20
* applications.
*
* Additionally, an {Approval} event is emitted on calls to {transferFrom}.
* This allows applications to reconstruct the allowance for all accounts j
* by listening to said events. Other implementations of the EIP may not emit
* these events, as it isn't required by the specification.
*
* Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
* functions have been added to mitigate the well-known issues around setting
* allowances. See {IERC20-approve}.
 */
  contract ERC20 is Context, IERC20, IERC20Metadata {
  mapping(address => uint256) private _balances;

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

  uint256 private _totalSupply;

  string private _name;
  string private _symbol;

        /**
  * @dev Sets the values for {name} and {symbol}.
  *
  * The default value of {decimals} is 18. To select a different value for
  * {decimals} you should overload it.
  *
  * All two of these values are immutable: they can only b
  * construction.
   */
      constructor(string memory name_, string memory symbol_) {
      _name = name_;
      _symbol = symbol_;
      }

        /**
  * @dev Returns the name of the token.
   */
      function name() public view virtual override returns (string memory) {
      return _name;
      }

        /**
  * @dev Returns the symbol of the token, usually
  * name.
   */
      function symbol() public view virtual override returns (string memory) {
      return _symbol;
      }

        /**
  * @dev Returns the number of decimals used to get its user representation.
  * For example, if `decimals` equals `2`, a balance of `505` tokens shou
  * be displayed to a user as `5.05` (`505 / 10 ** 2`).
  *
  * Tokens usually opt for a value of 18, imitating the relati
  * Ether and Wei. This is the value {ERC20} uses, unless this function is
  * overridden;
  *
  * NOTE: This information is only used for _display_ purposes: it in
  * no way affects any of the arithmetic of the contract, inc
  * {IERC20-balanceOf} and {IERC20-transfer}.
   */
      function decimals() public view virtual override returns (uint8) {
      return 18;
      }

        /**
  * @dev See {IERC20-totalSupply}.
   */
      function totalSupply() public view virtual override returns (uint256) {
      return _totalSupply;
      }

        /**
  * @dev See {IERC20-balanceOf}.
   */
      function balanceOf(address account) public view virtual override returns (uint256) {
      return _balances[account];
      }

        /**
  * @dev See {IERC20-transfer}.
  *
  * Requirements:
  *
  * - `recipient` cannot be the zero address.
  * - the caller must have a balance of at least `amount`.
    */
       function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
       _transfer(_msgSender(), recipient, amount);
       return true;
       }

        /**
  * @dev See {IERC20-allowance}.
   */
      function allowance(address owner, address spender) public view virtual override returns (uint25
      return _allowances[owner][spender];
      }

        /**
  * @dev See {IERC20-approve}.
  *
  * Requirements:
  *
  * - `spender` cannot be the zero address.
    */
       function approve(address spender, uint256 amount) public virtual override returns (bool) {
       _approve(_msgSender(), spender, amount);
```

```
        return true;
        }

              /**
    * @dev See {IERC20-transferFrom}.
    *
    * Emits an {Approval} event indicating the updated allow
    * required by the EIP. See the note at the
    *
    * 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 of at least
    * `amount`.
     */
      function transferFrom(
      address sender,
      address recipient,
      uint256 amount
      ) public virtual override returns (bool) {
      _transfer(sender, recipient, amount);

      uint256 currentAllowance = _allowances[sender][_msgSender()];
      require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
      unchecked {
      _approve(sender, _msgSender(), currentAllowance - amount);
      }

      return true;
      }

              /**
    * @dev Atomically increases the allowance granted to `spender` by th
    *
    * This is an alternative to {approve} that can be used as a
    * problems described in {IERC20-approve}.
    *
    * Emits an {Approval} event indicating the updated allow
    *
    * Requirements:
    *
    * - `spender` cannot be the zero address.
      */
        function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool)
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender] + addedValue);
        return true;
        }

              /**
    * @dev Atomically decreases the allowance granted to `spender` by t
    *
    * This is an alternative to {approve} that can be used as a
    * problems described in {IERC20-approve}.
    *
    * Emits an {Approval} event indicating the updated allow
    *
    * Requirements:
    *
    * - `spender` cannot be the zero address.
    * - `spender` must have allowance for the caller of at least
    * `subtractedValue`.
     */
       function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bo
       uint256 currentAllowance = _allowances[_msgSender()][spender];
       require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
       unchecked {
       _approve(_msgSender(), spender, currentAllowance - subtractedValue);
       }

       return true;
       }

                /**
    * @dev Moves `amount` of tokens from `sender` to `recipient`.
    *
    * This internal function is equivalent to {transfer}, and can be used to
    * e.g. implement automatic token fees, slashing mechanisms, etc.
    *
    * Emits a {Transfer} event.
    *
    * Requirements:
    *
    * - `sender` cannot be the zero address.
    * - `recipient` cannot be the zero address.
    * - `sender` must have a balance of at least `amount`.
      */
         function _transfer(
         address sender,
         address recipient,
         uint256 amount
         ) internal virtual {
         require(sender != address(0), "ERC20: transfer from the zero address");
         require(recipient != address(0), "ERC20: transfer to the zero address");

     _beforeTokenTransfer(sender, recipient, amount);

     uint256 senderBalance = _balances[sender];
     require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
     unchecked {
     _balances[sender] = senderBalance - amount;
     }
     _balances[recipient] += amount;

     emit Transfer(sender, recipient, amount);

     _afterTokenTransfer(sender, recipient, amount);
     }

                /** @dev Creates `amount` tokens and assigns them to `account`, increasing
    * the total supply.
    *
    * Emits a {Transfer} event with `from` set to the zero add
    *
```

```
* Requirements:
*
* - `account` cannot be the zero address.
  */
     function _mint(address account, uint256 amount) internal virtual {
     require(account != address(0), "ERC20: mint to the zero address");

  _beforeTokenTransfer(address(0), account, amount);

  _totalSupply += amount;
  _balances[account] += amount;
  emit Transfer(address(0), account, amount);

  _afterTokenTransfer(address(0), account, amount);
  }

             /**
* @dev Destroys `amount` tokens from `account`, reducing the
* total supply.
*
* Emits a {Transfer} event with `to` set to the zero addres
*
* Requirements:
*
* - `account` cannot be the zero address.
* - `account` must have at least `amount` tokens.
  */
     function _burn(address account, uint256 amount) internal virtual {
     require(account != address(0), "ERC20: burn from the zero address");

  _beforeTokenTransfer(account, address(0), amount);

  uint256 accountBalance = _balances[account];
  require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
  unchecked {
  _balances[account] = accountBalance - amount;
  }
  _totalSupply -= amount;

  emit Transfer(account, address(0), amount);

  _afterTokenTransfer(account, address(0), amount);
  }

             /**
* @dev Sets `amount` as the allowance of `spender` over the
*
* This internal function is equivalent to `approve`, and can be used to
* e.g. set automatic allowances for certain subsystems, etc.
*
* Emits an {Approval} event.
*
* Requirements:
*
* - `owner` cannot be the zero address.
* - `spender` cannot be the zero address.
  */
```

```
         function _approve(
         address owner,
         address spender,
         uint256 amount
         ) internal virtual {
         require(owner != address(0), "ERC20: approve from the zero address");
         require(spender != address(0), "ERC20: approve to the zero address");

     _allowances[owner][spender] = amount;
     emit Approval(owner, spender, amount);
     }

              /**
  * @dev Hook that is called before any transfer of tokens. This includes
  * minting and burning.
  *
  * Calling conditions:
  *
  * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
  * will be transferred to `to`.
  * - when `from` is zero, `amount` tokens will be minted for `to`.
  * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
  * - `from` and `to` are never both zero.
  *
  * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hoo
   */
       function _beforeTokenTransfer(
       address from,
       address to,
       uint256 amount
       ) internal virtual {}

              /**
  * @dev Hook that is called after any transfer of tokens. This includes
  * minting and burning.
  *
  * Calling conditions:
  *
  * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
  * has been transferred to `to`.
  * - when `from` is zero, `amount` tokens have been minted for `to`.
  * - when `to` is zero, `amount` of ``from``'s tokens have been burned.
  * - `from` and `to` are never both zero.
  *
  * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hoo
   */
       function _afterTokenTransfer(
       address from,
       address to,
       uint256 amount
       ) internal virtual {}
       }

// File: @openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol


// OpenZeppelin Contracts v4.4.0 (token/ERC20/extensions/ERC20Burnable.sol)
```

```
pragma solidity ^0.8.0;



            /**
* @dev Extension of {ERC20} that allows token holders to destroy both their own
* tokens and those that they have an allowance for, in
* recognized off-chain (via event analysis).
 */
  abstract contract ERC20Burnable is Context, ERC20 {
            /**
  * @dev Destroys `amount` tokens from the caller.
  *
  * See {ERC20-_burn}.
   */
      function burn(uint256 amount) public virtual {
      _burn(_msgSender(), amount);
      }

            /**
  * @dev Destroys `amount` tokens from `account`, deducting from the caller's
  * allowance.
  *
  * See {ERC20-_burn} and {ERC20-allowance}.
  *
  * Requirements:
  *
  * - the caller must have allowance for ``accounts``'s tokens of at least
  * `amount`.
   */
      function burnFrom(address account, uint256 amount) public virtual {
      uint256 currentAllowance = allowance(account, _msgSender());
      require(currentAllowance >= amount, "ERC20: burn amount exceeds allowance");
      unchecked {
      _approve(account, _msgSender(), currentAllowance - amount);
      }
      _burn(account, amount);
      }
      }

// File: contracts/TKLRole.sol


pragma solidity ^0.8.2;





contract TKL is ERC20, ERC20Burnable, Pausable, AccessControl {

    address [20] private _whiteAddress = [
    0x60A2aF5F6309840335Dc4896a1D330940Bf95b91,
    0xe65AEEfa511ee4Fd34eA6A4b062a8ED7f3Df747d,
    0x7c861f5fF977b906416Bff7fa4003ce0C77BCb2E,
    0x3CD5E18739991032963D8AaBCF1a96b42930f5b9,
    0xa4Edb9fbBD0fece358d38AE8F70f398486b51ba0,
    0x6916771D1b7856c16CcDd34561Dc23e68D408a34,
0x2f3F1814662344B76E679eC68e96DF1622cbca43,
0x7578e019dEBA6a7a95F037083c13b336a47b1f4E,
0x411B1C0fD58df164E65386556673e70157219df8,
0x7c4b2d955067Bd62233c9582bceE4d60f0fa5D90,
0x6C9437A1CC3f2D33aa7734403f884BB1755fD001,
0xffe1dC4B2F0811a9fEf3c238bd9935B0428F7C40,
0xB4cEf04DF82eD67675Ae5684AE04f662f9D76698,
0xc95cA041ad2aF8D3F353e48B842bA94b45c9Ecdc,
0xa1D156be0f35BD884460158d625cCEca2030323d,
0x2B8c117CC169946A872718aca2Ca24727579CeC2,
0x7690E8Dc6AD19006D161C6dC22De526b60D97C86,
0x8f3AfCB215A70A09583fA5d8839023953F90d2f4,
0xA5Ccf6258cab11105bD688B1DC8c33a5Fe08870a,
0x028f2FFdD938C72e6d0169eB991CA084Eb98648d
];

uint256 public GameLockBalance  = 500000000 * 10 ** decimals();
uint256 public StakeLockBalance = 9000000000 * 10 ** decimals();

address public LiquidityPool = 0x3629D6E1f8013b1f76858E30aaa5612aC9833750;

bytes32 public constant UNLOCKER_ROLE = keccak256("UNLOCKER_ROLE");

constructor() ERC20("TKL", "TKL") {
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(UNLOCKER_ROLE, msg.sender);

    _mint(address(this), GameLockBalance);
    _mint(address(this), StakeLockBalance);
    _mint(0x3E7ba5ce0b25118F5A20CB0Be76b16b86E9c9c92, 100000000 * 10 ** decimals());
    _mint(0x3D8Ec132882FDF2d538E91e2226357c41E19AEc2, 100000000 * 10 ** decimals());
    _mint(0x96824F32b4BfFEEede15bf9dE97E7a43fce7B486, 300000000 * 10 ** decimals());
}

function pause() public onlyRole(DEFAULT_ADMIN_ROLE) {
    _pause();
}

function unpause() public onlyRole(DEFAULT_ADMIN_ROLE) {
    _unpause();
}

function _beforeTokenTransfer(address from, address to, uint256 amount)
internal
whenNotPaused
override
{
    super._beforeTokenTransfer(from, to, amount);
}

function transferFrom(address sender, address recipient, uint256 amount) public override returns
    require(sender != address(0), "ERC20: send not allow the zero address");

    for(uint i = 0; i < _whiteAddress.length; i++) {
        if (sender == _whiteAddress[i]){
            super.transferFrom(sender,recipient,amount);
            return true;
        }

    }
    uint256 _fee = amount*5/100;
    uint256 _rest = amount-_fee*2;

    _transfer(sender, LiquidityPool, _fee);

    _transfer(sender, recipient, _rest);
```

```
        burnFrom(sender,_fee);

        _approve(sender, _msgSender(), allowance(sender,_msgSender()) + _fee - amount);
        return true;
    }

    function unlockGameBalance(address _to, uint256 _amount) public onlyRole(UNLOCKER_ROLE) returns(b

        require(GameLockBalance >= _amount, "ERC20: unlock amount exceeds balance");

    unchecked {
        GameLockBalance = GameLockBalance - _amount;
    }

        (bool success, bytes memory data) = address(this).call(abi.encodeWithSelector(bytes4(keccak25
        require(success && (data.length == 0 || abi.decode(data, (bool))),'ERC20::transfer: transfer

        return true;
    }

    function unlockStakeBalance(address _to, uint256 _amount) public onlyRole(UNLOCKER_ROLE) returns(

        require(StakeLockBalance >= _amount, "ERC20: unlock amount exceeds balance");

    unchecked {
        StakeLockBalance = StakeLockBalance - _amount;
    }

        (bool success, bytes memory data) = address(this).call(abi.encodeWithSelector(bytes4(keccak25
        require(success && (data.length == 0 || abi.decode(data, (bool))),'ERC20::transfer: transfer

        return true;
    }

    function setLiquidityPool(address _newPool) public onlyRole(DEFAULT_ADMIN_ROLE) {
        LiquidityPool = _newPool;
    }

}
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs.

Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulnerabilities in smart contracts as will be discussed in this section.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Entropy Illusion

- **Description:**
All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

External Contract Referencing

- **Description:**
One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Unsolved TODO comments

- **Description:**
Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io