



DPTECH

DPTECH AUDIT REPORT

For BullDogToken (BUDG)

On August 3rd, 2021

Provided By DPTech Team

TABLE OF CONTENTS

OVERVIEW	2
The Target Been Audited	2
VULNERABILITIES ANALYSIS	3
Risk of the vulnerabilities :	3
Audit Result : PASSED	3
AUDIT RESULT ANALYSIS	4
Replay Attack [PASSED]	4
Overflow Audit [PASSED]	4
Access Control of Owner [PASSED]	4
Returned Value Security [PASSED]	4
Pseudo-random Number Generator(PRNG) [PASSED]	5
Transaction-Ordering Dependence [PASSED]	5
DoS [PASSED]	5
Design Logic [PASSED]	6
False top-up [PASSED]	6
Additional Issuance [PASSED]	6
Suspended Account Bypass [PASSED]	6
APPENDIX A: CONTRACT CODE	7
APPENDIX B: VULNERABILITY RISK RATING STANDARDS	24
APPENDIX C: INTRODUCTION TO VULNERABILITY TESTING TOOLS	25
DECLARATION	26

I. OVERVIEW

DPTech Team(The Team) has received the requirement of auditing the **BullDogToken (BUDG)**; hence, the team has run a comprehensive audit for BUDG on **3rd, August 2021** under the DPtech security and tech standards.

Within the audit process, a comprehensive analysis had been conducted for scanning the common vulnerabilities(Included in Chapter 3); in which the code of **BUDG** met the standards of DPtech technologies, and no known common vulnerabilities were found through the audit process. Hence the audit result is **PASSED**.

The audit is carried out in a test environment, meanwhile, the latest code has been audited under the test environment and related regulation, no other effects have been made to the code on the production environment.

The Target Been Audited

Token	BullDogToken (BUDG)
Code Type	Token
Contract Address:	0x446320c9ffa57030ca977a1f90f8049df4004647
Link Of The Code:	https://bscscan.com/address/0x446320c9ffa57030ca977a1f90f8049df4004647#code
Program Language	Solidity 0.6.12

II. VULNERABILITIES ANALYSIS

Risk of the vulnerabilities :

Risk Ladder			
HIGH	MEDIUM	LOW	PASSED
0	0	0	11

Audit Result: PASSED

VULNERABILITIES	RESULT	DESCRIBE
Replay Attack	PASSED	Check use of call.value() functions
Overflow Audit	PASSED	Check the add and sub functions
Access Control of Owner	PASSED	Check access control of each operation
Returned Value Security	PASSED	Check the transfer method with return value
Pseudo-random Number Generator(PRNG)	PASSED	Check whether there is a unified content filter
Transaction-Ordering Dependence	PASSED	Check transaction sequence dependency
DoS	PASSED	Check the uses of resources
Design Logic	PASSED	Check the security issues related to business model
False top-up	PASSED	Check whether there is a false top-up vulnerability
Additional Issuance	PASSED	Check whether there is a function allowed additional issuance
Suspended Account Bypass	PASSED	Check if the suspended account can be bypassed

III. AUDIT RESULT ANALYSIS

1. Replay Attack [**PASSED**]

The replay attack is the most famous Ethereum smart contract vulnerability, which has led to the Ethereum fork (The DAO hack)

The `call.value()` function in Solidity consumes all the gas it receives when it is used to send Ether. There is a risk of replay attacks when the `call.value()` function sending Ether occurs before actually reducing the balance of the sender's account.

Test result: There is no relevant call function in the smart contract code.

Suggestions: None.

2. Overflow Audit [**PASSED**]

The overflow problem in smart contracts refers to integer overflow and integer underflow.

Solidity can handle up to 256 digits ($2^{256}-1$), and increasing the maximum number by 1 will overflow to 0. Similarly, when the number is of unsigned type, 0 minus 1 will underflow to get the maximum number value. Overflow and underflow are not a new type of vulnerability, but they are particularly dangerous in smart contracts. An overflow situation can lead to incorrect results, especially if the possibility is not expected, which may affect the reliability and security of the program.

Test result: The security problem does not exist in the smart contract code.

Suggestions: None.

3. Access Control of Owner [**PASSED**]

Access control deficiencies are possible security risks in all programs. Smart contracts also have similar problems. The famous Parity Wallet smart contract has been affected by this problem.

Test result: The security problem does not exist in the smart contract code.

Suggestions: None.

4. Returned Value Security [**PASSED**]

This problem mostly occurs in smart contracts related to token transfer, so it is also called silent failure transmission or unchecked transmission.

In Solidity, there are transfer methods such as `transfer()`, `send()`, `call.value()`, etc., which can be used to send Ether to an address, the difference is that: `transfer` will throw when the

transfer fails, and the status will be rolled back; Only 2300 gas will be passed for calling to prevent reentry attacks; false will be returned when send fails to send; only 2300 gas will be passed for calling to prevent reentry attacks; false will be returned when call.value fails to be sent; all available gas will be called for Restricted by passing in the gas_value parameter), cannot effectively prevent reentry attacks. If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which may cause unexpected results due to the failure of Ether transmission.

Test result: There are no related vulnerabilities in the smart contract code.

Suggestions: None.

5. Pseudo-random Number Generator(PRNG) [PASSED]

Random numbers may be required in smart contracts. Although the functions and variables provided by Solidity can access obviously unpredictable values, such as block.number and block.timestamp, they are usually either more public than they seem or are affected by miners, these random numbers are predictable to a certain extent, so malicious users can usually copy it and rely on its unpredictability to attack the function.

Test result: The problem does not exist in the smart contract code.

Suggestions: None.

6. Transaction-Ordering Dependence [PASSED]

Since miners always obtain gas fees through code representing externally owned addresses (EOA), users can specify a higher fee for faster transactions. Since the Ethereum blockchain is public, everyone can see the content of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transaction at a higher fee to preempt the original solution.

Test result: There is no risk of transaction order dependence attack in the approval function in the smart contract.

Suggestions: None.

7. DoS [PASSED]

In the Ethereum world, denial of service is fatal, and a smart contract that suffers this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of the smart contract, including malicious behavior when acting as the recipient of the transaction, artificially increasing the gas required for the calculation function leads to the exhaustion of gas, abuse of access control to access the private component of the smart contract, the use of confusion and negligence, etc.

Test result: There are no related vulnerabilities in the smart contract code.

Suggestions: None.

8. Design Logic [PASSED]

Detects security issues related to business design in smart contract code.

Test result: There are no related vulnerabilities in the logic design of the smart contract code.

Suggestions: None

9. False top-up [PASSED]

In the transfer function of the token contract, the balance check of the transfer initiator (msg.sender) uses the if judgment method. When balances[msg.sender]<value, it enters the else logic part and returns false. Finally, no exception is thrown. We believe that only the gentle judgment method of if/else is an imprecise coding method in the context of sensitive functions such as transfer.

Test result: There are no related vulnerabilities in the smart contract code.

Suggestions: None.

10. Additional Issuance [PASSED]

After the initialization of the total amount of tokens, check whether there is a function in the token contract that may increase the total amount of tokens.

Test result: No relevant vulnerabilities have been found.

Suggestions: None.

11. Suspended Account Bypass [PASSED]

Check whether there is any operation of an unverified source account, originating account and target account when transferring tokens in the token contract.

Test result: The problem does not exist in the smart contract code (No whiteList).

Suggestions: None.

12. Additional Suggestions:

None

IV. APPENDIX A: CONTRACT CODE

Note: Comments start with `//AUDIT//`

```
//AUDIT// No Overflow or Race Condition vulnerabilities been found

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

// SPDX-License-Identifier: MIT
pragma solidity 0.6.12;

/*
 * @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.
 */
contract Context { //AUDIT// Standard context related function
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor() internal {}

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

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

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context { //AUDIT// Standard owner related function
    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 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 onlyOwner {
    _transferOwnership(newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new account ('newOwner').
 */
function _transferOwnership(address newOwner) internal {
    require(newOwner != address(0), 'Ownable: new owner is the zero address');
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
}

interface IBEP20 {/**AUDIT// Standard ERC20 framewor which comply with all protocol related interfaces  /**
 * @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);

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

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

    /**
```

```
* @dev Returns the bep token owner.
*/
function getOwner() external view returns (address);

/**
 * @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);
}

/**
 * @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.
 */
library SafeMath {//AUDIT// Standard SafeMath Library, no vulnerabilities have been found yet Solidity version 0.8+
is recommended
    /**
     * @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;
}

function min(uint256 x, uint256 y) internal pure returns (uint256 z) {
    z = x < y ? x : y;
}

// babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_method)
function sqrt(uint256 y) internal pure returns (uint256 z) {
    if (y > 3) {
        z = y;
        uint256 x = y / 2 + 1;
        while (x < z) {
            z = x;
            x = (y / x + x) / 2;
        }
    } else if (y != 0) {
        z = 1;
    }
}

/**
 * @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[c
 * 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?highlight=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);
        }
    }
}

/**
 * @dev Implementation of the {IBEP20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {BEP20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-BEP20-supply-mechanisms/226
 * to implement supply mechanisms.
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of BEP20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * 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 {IBEP20-approve}.
 */
contract BEP20 is Context, IBEP20, Ownable { //AUDIT// Accomplished with all BEP20 functions and no
vulunabliteis have been found; Contract administrators are able to mint new tokens in the admin account
    using SafeMath for uint256;
    using Address for address;

    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;
```



```
/**
 * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
 * a default value of 18.
 *
 * To select a different value for {decimals}, use {_setupDecimals}.
 *
 * All three of these values are immutable: they can only be set once during
 * construction.
 */
constructor(string memory name, string memory symbol) public {
    _name = name;
    _symbol = symbol;
    _decimals = 18;
    _mint(msg.sender, 100000000e18);
}

/**
 * @dev Returns the bep token owner.
 */
function getOwner() external override view returns (address) {
    return owner();
}

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

/**
 * @dev Returns the token decimals.
 */
function decimals() public override view returns (uint8) {
    return _decimals;
}

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

/**
 * @dev See {BEP20-totalSupply}.
 */
function totalSupply() public override view returns (uint256) {
    return _totalSupply;
}

/**
 * @dev See {BEP20-balanceOf}.
 */
function balanceOf(address account) public override view returns (uint256) {
    return _balances[account];
}

/**
 * @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) public override returns (bool) {
    _transfer(_msgSender(), recipient, amount);
    return true;
}

/**
 * @dev See {BEP20-allowance}.
 */
function allowance(address owner, address spender) public override view returns (uint256) {
    return _allowances[owner][spender];
}

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

/**
 * @dev See {BEP20-transferFrom}.
 *
 * Emits an {Approval} event indicating the updated allowance. This is not
 * required by the EIP. See the note at the beginning of {BEP20};
 *
 * 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 override returns (bool) {
    _transfer(sender, recipient, amount);
    _approve(
        sender,
        _msgSender(),
        _allowances[sender][_msgSender()].sub(amount, 'BEP20: transfer amount exceeds allowance')
    );
    return true;
}

/**
 * @dev Atomically increases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for
 * problems described in {BEP20-approve}.
 *
 * Emits an {Approval} event indicating the updated allowance.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
    _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
    return true;
}
```

```
/**
 * @dev Atomically decreases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for
 * problems described in {BEP20-approve}.
 *
 * Emits an {Approval} event indicating the updated allowance.
 *
 * 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 returns (bool) {
    _approve(
        _msgSender(),
        spender,
        _allowances[_msgSender()][spender].sub(subtractedValue, 'BEP20: decreased allowance below zero')
    );
    return true;
}

/**
 * @dev Creates `amount` tokens and assigns them to `msg.sender`, increasing
 * the total supply.
 *
 * Requirements
 *
 * - `msg.sender` must be the token owner
 */
function mint(uint256 amount) public onlyOwner returns (bool) { //AUDIT//Contract administrators are able to mint
new tokens in the admin account

    _mint(_msgSender(), amount);
    return true;
}

/**
 * @dev Moves tokens `amount` from `sender` to `recipient`.
 *
 * This is 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 {
    require(sender != address(0), 'BEP20: transfer from the zero address');
    require(recipient != address(0), 'BEP20: transfer to the zero address');

    _balances[sender] = _balances[sender].sub(amount, 'BEP20: transfer amount exceeds balance');
    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(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 address.
 *
 * Requirements
 *
 * - `to` cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal {
    require(account != address(0), 'BEP20: mint to the zero address');

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}

/**
 * @dev Destroys `amount` tokens from `account`, reducing the
 * total supply.
 *
 * Emits a {Transfer} event with `to` set to the zero address.
 *
 * Requirements
 *
 * - `account` cannot be the zero address.
 * - `account` must have at least `amount` tokens.
 */
function _burn(address account, uint256 amount) internal {//AUDIT// Burn function is not used;
    require(account != address(0), 'BEP20: burn from the zero address');

    _balances[account] = _balances[account].sub(amount, 'BEP20: burn amount exceeds balance');
    _totalSupply = _totalSupply.sub(amount);
    emit Transfer(account, address(0), amount);
}

/**
 * @dev Sets `amount` as the allowance of `spender` over the `owner`'s tokens.
 *
 * This is 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 {
    require(owner != address(0), 'BEP20: approve from the zero address');
    require(spender != address(0), 'BEP20: approve to the zero address');

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

/**
 * @dev Destroys `amount` tokens from `account`. `amount` is then deducted
 * from the caller's allowance.
 *
 * See {_burn} and {_approve}.
 */
```

```
*/
function _burnFrom(address account, uint256 amount) internal {
    _burn(account, amount);
    _approve(
        account,
        _msgSender(),
        _allowances[account][_msgSender()].sub(amount, 'BEP20: burn amount exceeds allowance')
    );
}
}

// BUDGToken with Governance.
contract BullDogToken is BEP20('BullDogToken', 'BUDG') {//AUDIT//According to the account balance, assigned to selected rights. Online and offline signature adjustment is supported. // @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).

    function mint(address _to, uint256 _amount) public onlyOwner //AUDIT//functions allowed contract administrators to mint new tokens in any account
    {
        _mint(_to, _amount);
        _moveDelegates(address(0), _delegates[_to], _amount);
    }

    // @notice A record of each accounts delegate
    mapping (address => address) internal _delegates;

    // @notice A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }

    // @notice A record of votes checkpoints for each account, by index
    mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;

    // @notice The number of checkpoints for each account
    mapping (address => uint32) public numCheckpoints;

    // @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");

    // @notice The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

    // @notice A record of states for signing / validating signatures
    mapping (address => uint) public nonces;

    // @notice An event thats emitted when an account changes its delegate
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

    // @notice An event thats emitted when a delegate account's vote balance changes
    event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

    /**
     * @notice Delegate votes from `msg.sender` to `delegatee`
     * @param delegator The address to get delegatee for
     */
    function delegates(address delegator)
        external
        view
        returns (address)
    {
        return _delegates[delegator];
    }
}
```

```
/**
 * @notice Delegate votes from `msg.sender` to `delegatee`
 * @param delegatee The address to delegate votes to
 */
function delegate(address delegatee) external {
    return _delegate(msg.sender, delegatee);
}

/**
 * @notice Delegates votes from signatory to `delegatee`
 * @param delegatee The address to delegate votes to
 * @param nonce The contract state required to match the signature
 * @param expiry The time at which to expire the signature
 * @param v The recovery byte of the signature
 * @param r Half of the ECDSA signature pair
 * @param s Half of the ECDSA signature pair
 */
function delegateBySig(
    address delegatee,
    uint nonce,
    uint expiry,
    uint8 v,
    bytes32 r,
    bytes32 s
)
    external
{
    bytes32 domainSeparator = keccak256(
        abi.encode(
            DOMAIN_TYPEHASH,
            keccak256(bytes(name())),
            getChainId(),
            address(this)
        )
    );

    bytes32 structHash = keccak256(
        abi.encode(
            DELEGATION_TYPEHASH,
            delegatee,
            nonce,
            expiry
        )
    );

    bytes32 digest = keccak256(
        abi.encodePacked(
            "\x19\x01",
            domainSeparator,
            structHash
        )
    );

    address signatory = ecrecover(digest, v, r, s);
    require(signatory != address(0), "BUDG::delegateBySig: invalid signature");
    require(nonce == nonces[signatory]++, "BUDG::delegateBySig: invalid nonce");
    require(now <= expiry, "BUDG::delegateBySig: signature expired");
    return _delegate(signatory, delegatee);
}

/**
 * @notice Gets the current votes balance for `account`
 * @param account The address to get votes balance
 * @return The number of current votes for `account`
 */
```

```
function getCurrentVotes(address account)
    external
    view
    returns (uint256)
{
    uint32 nCheckpoints = numCheckpoints[account];
    return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
}

/**
 * @notice Determine the prior number of votes for an account as of a block number
 * @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
 * @param account The address of the account to check
 * @param blockNumber The block number to get the vote balance at
 * @return The number of votes the account had as of the given block
 */
function getPriorVotes(address account, uint blockNumber)
    external
    view
    returns (uint256)
{
    require(blockNumber < block.number, "BUDG::getPriorVotes: not yet determined");

    uint32 nCheckpoints = numCheckpoints[account];
    if (nCheckpoints == 0) {
        return 0;
    }

    // First check most recent balance
    if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
        return checkpoints[account][nCheckpoints - 1].votes;
    }

    // Next check implicit zero balance
    if (checkpoints[account][0].fromBlock > blockNumber) {
        return 0;
    }

    uint32 lower = 0;
    uint32 upper = nCheckpoints - 1;
    while (upper > lower) {
        uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
        Checkpoint memory cp = checkpoints[account][center];
        if (cp.fromBlock == blockNumber) {
            return cp.votes;
        } else if (cp.fromBlock < blockNumber) {
            lower = center;
        } else {
            upper = center - 1;
        }
    }
    return checkpoints[account][lower].votes;
}

function _delegate(address delegator, address delegatee)
    internal
{
    address currentDelegate = _delegates[delegator];
    uint256 delegatorBalance = balanceOf(delegator); // balance of underlying BUDGs (not scaled);
    _delegates[delegator] = delegatee;

    emit DelegateChanged(delegator, currentDelegate, delegatee);

    _moveDelegates(currentDelegate, delegatee, delegatorBalance);
}
```

```
function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal {
    if (srcRep != dstRep && amount > 0) {
        if (srcRep != address(0)) {
            // decrease old representative
            uint32 srcRepNum = numCheckpoints[srcRep];
            uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
            uint256 srcRepNew = srcRepOld.sub(amount);
            _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
        }

        if (dstRep != address(0)) {
            // increase new representative
            uint32 dstRepNum = numCheckpoints[dstRep];
            uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
            uint256 dstRepNew = dstRepOld.add(amount);
            _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
        }
    }
}

function _writeCheckpoint(
    address delegatee,
    uint32 nCheckpoints,
    uint256 oldVotes,
    uint256 newVotes
)
internal
{
    uint32 blockNumber = safe32(block.number, "BUDG::_writeCheckpoint: block number exceeds 32 bits");

    if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
        checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
    } else {
        checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
        numCheckpoints[delegatee] = nCheckpoints + 1;
    }

    emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
}

function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
    require(n < 2**32, errorMessage);
    return uint32(n);
}

function getChainId() internal pure returns (uint) {
    uint256 chainId;
    assembly { chainId := chainid() }
    return chainId;
}
```


V. APPENDIX B: VULNERABILITY RISK RATING STANDARDS

Vulnerability	Vulnerability Instructions
High-risk	Vulnerabilities that can directly cause the loss of token or user funds, such as: overflow vulnerabilities that can cause the value of the token to return to zero, false top-up vulnerabilities that can cause the exchanges lose tokens, and replay attack can cause contract accounts to losses ETH or token , etc.; vulnerabilities that can cause the loss of ownership of token contracts, such as: access control of key functions, bypass of key function access control caused by call injection, etc.; vulnerabilities that can cause token contracts to not work properly, such as: Denial of service caused by malicious address sending ETH, denial of service vulnerability due to gas exhaustion.
Medium-risk	Vulnerabilities that require specific addresses to trigger, such as overflow vulnerabilities that can only be triggered by the token contract owner; access control defects of non-critical functions, logical design defects that cannot cause direct capital loss, etc.
Low-risk	Vulnerabilities that are difficult to be triggered, vulnerabilities with limited damage after triggering, such as overflow vulnerabilities that require a large amount of ETH or tokens to trigger, vulnerabilities that the attacker cannot directly profit after the numerical overflow is triggered, and the transaction sequence dependence risk triggered by specifying high gas etc.

VI. APPENDIX C: INTRODUCTION TO VULNERABILITY TESTING TOOLS

Manticore	Manticore is a symbolic execution tool for analyzing binary files and smart contracts. Manticore includes a symbolic Ethereum Virtual Machine (EVM), an EVM disassembler/assembler, and a convenient interface for automatic compilation and analysis of Solidity. It also integrates Ethersplay, a Bit of Traits of Bits visual disassembler for EVM bytecode, for visual analysis. Like binary files, Manticore provides a simple command-line interface and a Python API for analyzing EVM bytecode.
Oyente	Oyente is a smart contract analysis tool. Oyente can be used to detect common bugs in smart contracts, such as reentrancy, transaction ordering dependencies, and so on. What's more convenient is that Oyente's design is modular, so this allows advanced users to implement and insert their own detection logic to check custom attributes in their contracts.
Securify.sh	Securify can verify the common security problems of Ethereum smart contracts, such as out-of-order transactions and lack of input verification. It analyzes all possible execution paths of the program while fully automated. In addition, Securify has a specific language for specifying vulnerabilities, which makes Securify can keep abreast of current security and other reliability issues.
Echidna	Echidna is a Haskell library designed for fuzzing EVM code.
MAIAN	MAIAN is an automated tool for finding Ethereum smart contract vulnerabilities. Maian processes the contract's bytecode and attempts to establish a series of transactions to find and confirm errors.
Ethersplay	Ethersplay is an EVM disassembler, which contains related analysis tools.
Ida-evm	Ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).
Remix-ide	Remix is a browser-based compiler and IDE that allows users to build Ethereum contracts and debug transactions using the Solidity language.

DECLARATION

It is offered subject to acceptance without modification of all of the terms and conditions contained herein and all other operating rules, policies and procedures that may be published from time to time on this Site by **DPTech**(collectively, the Terms of Use).

As a condition of use, you promise not to use the content of the website for any purpose that is unlawful or prohibited by these Terms of Use, or any other purpose not reasonably intended by **DPTech**.

You agree to abide by all applicable local, state, national and international laws. By way of example, and not as a limitation, you agree not to:

- (a) take any action or
- (b) upload, post, submit or otherwise distribute or facilitate distribution of any content (including text, communications, software, images, sounds, data, code, chat, blogs, Podcasts, messages, files, video or other information) using any communications service or other service available on or through the content of the website, that:
 - ❖ is unlawful, untrue, threatening, abusive, harassing, defamatory, libelous, deceptive, fraudulent, invasive of another's privacy, tortious, obscene, profane or which otherwise violates the Terms of Use;
 - ❖ infringes any patent, trademark, trade secret, copyright, right of publicity or other right of any party;
 - ❖ constitutes unauthorized or unsolicited advertising, junk or bulk e-mail ("spamming");
 - ❖ imposes an unreasonable or disproportionately large load on DPTech's computing, storage or communications infrastructure, or attempts to gain unauthorized access to the content of the website, other accounts, computer systems or networks connected to the content of the website, through data mining or otherwise;
 - ❖ contains software viruses or any other computer codes, files, or programs that are designed or intended to disrupt, damage, limit or interfere with the proper function of any software, hardware, or telecommunications equipment or to damage or obtain unauthorized access to any system, data or other information of DPTech or any third party; or harvests or collects any information from the Site; impersonates any person or entity, including any employee or representative of DPTech.
 - ❖ DPTech is unable to and has no obligation to monitor the content of the website or any use thereof. You agree not to provide DPTech with any confidential or proprietary information that you desire or are required to keep secret.

The content of the website AND ALL MATERIALS, INFORMATION, SOFTWARE, PRODUCTS AND SERVICES INCLUDED IN OR AVAILABLE THROUGH THE SITE (THE CONTENT) ARE PROVIDED "AS IS" AND "AS AVAILABLE". the content of the website AND CONTENT ARE PROVIDED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTIES IMPLIED BY ANY COURSE OF PERFORMANCE OR USAGE OF TRADE, ALL OF WHICH ARE EXPRESSLY DISCLAIMED. DPTech DO NOT WARRANT THAT:

(A)FOR THE AUDIT REPORT, THE DPTech ONLY RESPONSIBLE FOR CHECKING THE POSSIBLE SECURITY PROBLEM THAT RELATED TO THE CODE, WITH THE AUDIT REPORT DOESN'T IMPLY AND INDICATED THAT THE CODE IS SECURITY, PLEASE REFER TO THE ACTUAL RESULT OF THE AUDIT REPORT.

(B)DPTech IS NOT PART OF OR RELATED TO ANY PART OF THIRD-PARTY PROJECT, ANY ISSUES OR CASES THAT RELATED TO THE PROJECT ARE NOT RELATED TO DPTech.

DPTech reserves the right to update, modify, maintain,make judgments, and the final interpretation of all content generated by DPTech.No agency, partnership, joint venture, or employment relationship is created as a result of this Agreement and neither party has any authority of any kind to bind the other in any respect. In any action or proceeding to enforce rights under this Agreement, the prevailing party will be entitled to recover costs and attorneys' fees.

DPTech does not represent that the content produced by this website is applicable to the laws of your region, and it is forbidden to access the content of the websites of this website from regions that regard this website as illegal content. If you access the content of the website from other locations, you do so at your own initiative and are responsible for compliance with local laws.

DPTECH AUDIT REPORT

Website: www.dpanquan.com

Contact us at:
service@dpanquan.com

