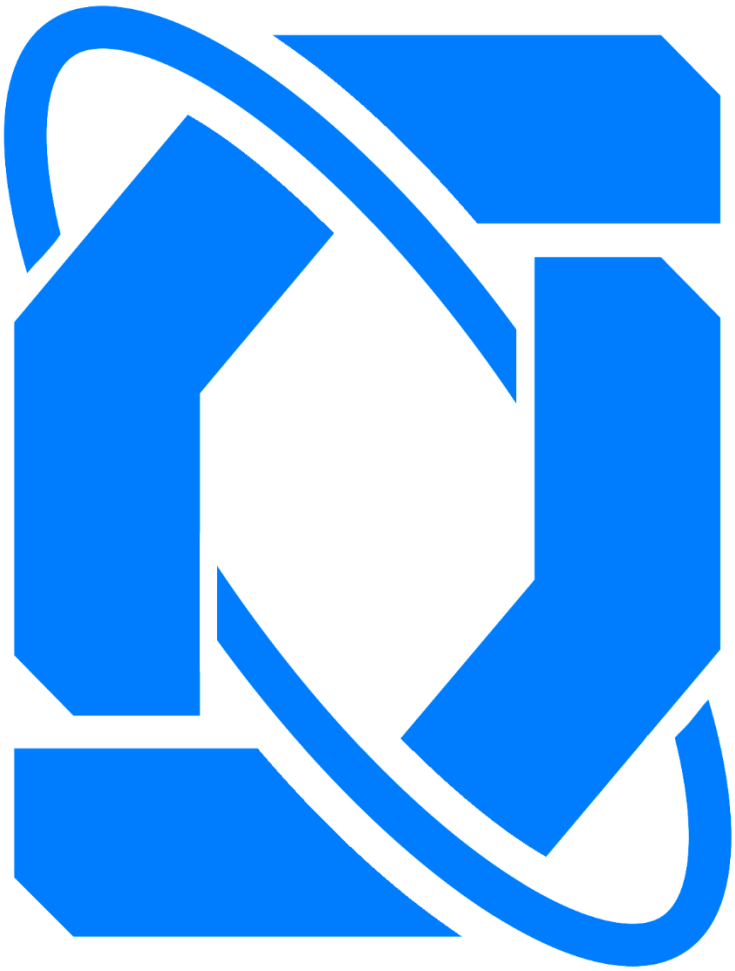


Orbital Station Excelsior Smart Contract Security Audit

Contents

Smart Contract Security Audit Start Point

Smart Contract & EVM Compiler Overview

Blockchain: Binance Smart Chain (BSC)

Address: 0x59FB9a377004CC8974009F72d94b4d801C58499B

Explorer Link: https://bscscan.com/address/0x59FB9a377004CC8974009F72d94b4d801C58499B

Token Name: Baby LTC

Verified: Yes

Status: Deployed

Symbol: (BLTC)

Contract Name: BABYTOKEN

Decimals: 18

Total Supply: 100000000000000000000000000

Deployment Date: May-05-2023 06:07:23 PM +UTC

Compiler Version: v0.8.4+commit.c7e474f2

License Type: MIT license

Optimization: 200 runs

Compiler Settings: default evmVersion, MIT license

Solidity Compiler Bugs for this version:

- FullInlinerNonExpressionSplitArgumentEvaluationOrder (low-severity),
- MissingSideEffectsOnSelectorAccess (low-severity),
- AbiReencodingHeadOverflowWithStaticArrayCleanup (medium-severity),
- DirtyBytesArrayToStorage (low-severity),
- DataLocationChangeInInternalOverride (very low-severity),
- NestedCalldataArrayAbiReencodingSizeValidation (very low-severity),
- SignedImmutables (very low-severity)

Code language: Solidity

Creator: 0xB811D56ed0E71794869A2445cb159B2C235e46c9 at txn 0x7ee12eebc88a9b2dc88f610782ad89aea00647973e62f3b26f9d5a618d81a1be

Date of audit: 2024/02

EVM Compiler Version Bugs Info

Bug Name: FullInlinerNonExpressionSplitArgumentEvaluationOrder
Description: Function call arguments in Yul are evaluated right to left. This order matters when the argument expressions have side-effects, and changing it may change contract behavior. FullInliner is an optimizer step that can replace a function call with the body of that function. The transformation involves assigning argument expressions to temporary variables, which imposes an explicit evaluation order. FullInliner was written with the assumption that this order does not necessarily have to match usual argument evaluation order because the argument expressions have no side-effects. In most circumstances this assumption is true because the default optimization step sequence contains the ExpressionSplitter step. ExpressionSplitter ensures that the code is in *expression-split form*, which means that function calls cannot appear nested inside expressions, and all function call arguments have to be variables. The assumption is, however, not guaranteed to be true in general. Version 0.6.7 introduced a setting allowing users to specify an arbitrary optimization step sequence, making it possible for the FullInliner to actually encounter argument expressions with side-effects, which can result in behavior differences between optimized and unoptimized bytecode. Contracts compiled without optimization or with the default optimization sequence are not affected. To trigger the bug the user has to explicitly choose compiler settings that contain a sequence with FullInliner step not preceded by ExpressionSplitter.

Bug Name: MissingSideEffectsOnSelectorAccess
Description: When accessing the ``.selector`` member on an expression with side-effects, like an assignment, a function call or a conditional, the expression would not be evaluated in the legacy code generation. This would happen in expressions where the functions used in the expression were all known at compilation time, regardless of whether the whole expression could be evaluated at compilation time or not. Note that the code generated by the IR pipeline was unaffected and would behave as expected.

Bug Name: AbiReencodingHeadOverflowWithStaticArrayCleanup
Description: When ABI-encoding a statically-sized calldata array, the compiler always pads the data area to a multiple of 32-bytes and ensures that the padding bytes are zeroed. In some cases, this cleanup used to be performed by always writing exactly 32 bytes, regardless of how many needed to be zeroed. This was done with the assumption that the data that would eventually occupy the area past the end of the array had not yet been written, because the encoder processes tuple components in the order they were given. While this assumption is mostly true, there is an important corner case: dynamically encoded tuple components are stored separately from the statically-sized ones in an area called the *tail* of the encoding and the tail immediately follows the *head*, which is where the statically-sized components are placed. The aforementioned cleanup, if performed for the last component of the head would cross into the tail and overwrite up to 32 bytes of the first component stored there with zeros. The only array type for which the cleanup could actually result in an overwrite were arrays with ``uint256`` or ``bytes32`` as the base element type and in this case the size of the corrupted area was always exactly 32 bytes. The problem affected tuples at any nesting level. This included also structs, which are encoded as tuples in the ABI. Note also that lists of parameters and return values of functions, events and errors are encoded as tuples.

Bug Name: DirtyBytesArrayToStorage
Description: Copying ``bytes`` arrays from memory or calldata to storage is done in chunks of 32 bytes even if the length is not a multiple of 32. Thereby, extra bytes past the end of the array may be copied from calldata or memory to storage. These dirty bytes may then become observable after a ``.push()`` without arguments to the bytes array in storage, i.e. such a push will not result in a zero value at the end of the array as expected. This bug only affects the legacy code generation pipeline, the new code generation pipeline via IR is not affected.

Bug Name: DataLocationChangeInInternalOverride
Description: When calling external functions, it is irrelevant if the data location of the parameters is ``calldata`` or ``memory``, the encoding of the data does not change. Because of that, changing the data location when overriding external functions is allowed. The compiler incorrectly also allowed a change in the data location for overriding public and internal functions. Since public functions can be called internally as well as externally, this causes invalid code to be generated when such an incorrectly overridden function is called internally through the base contract. The caller provides a memory pointer, but the called function interprets it as a calldata pointer or vice-versa.

Bug Name: NestedCalldataArrayAbiReencodingSizeValidation
Description: Calldata validation for nested dynamic types is deferred until the first access to the nested values. Such an access may for example be a copy to memory or an index or member access to the outer type. While in most such accesses calldata validation correctly checks that the data area of the nested array is completely contained in the passed calldata (i.e. in the range [0, calldatasize()]), this check may not be performed, when ABI encoding such nested types again directly from calldata. For instance, this can happen, if a value in calldata with a nested dynamic array is passed to an external call, used in ``abi.encode`` or emitted as event. In such cases, if the data area of the nested array extends beyond ``calldatasize()``, ABI encoding it did not revert, but continued reading values from beyond ``calldatasize()`` (i.e. zero values).

Bug Name: SignedImmutables
Description: When immutable variables of signed integer type shorter than 256 bits are read, their higher order bits were unconditionally set to zero. The correct operation would be to sign-extend the value,
i.e. set the higher order bits to one if the sign bit is one. This sign-extension is performed by Solidity just prior to when it matters, i.e. when a value is stored in memory, when it is compared or when a division is performed. Because of that, to our knowledge, the only way to access the value in its unclean state is by reading it through inline assembly.

Code Overview

```
// Dependency file: @openzeppelin/contracts/token/ERC20/IERC20.sol
// SPDX-License-Identifier: MIT
// pragma solidity ^0.8.0;
/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);
    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);
    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);
    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);
    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);
    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);
    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);
    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
// Dependency file: @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol
// pragma solidity ^0.8.0;
// import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 *
 * _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);
}
// Dependency file: @openzeppelin/contracts/utils/Context.sol
// pragma solidity ^0.8.0;
/**
 * @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 meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }
    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}
// Dependency file: @openzeppelin/contracts/token/ERC20/ERC20.sol
// pragma solidity ^0.8.0;
// import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
// import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
// import "@openzeppelin/contracts/utils/Context.sol";
/**
 * @dev Implementation of the {IERC20} 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 {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 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 {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 be set once during
     * 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 a shorter version of the
     * 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 should
     * be displayed to a user as `5.05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * 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, including
    * {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 (uint256) {
       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 allowance. This is not
    * required by the EIP. See the note at the beginning of {ERC20}.
    *
    * 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 the caller.
    *
    * This is an alternative to {approve} that can be used as a mitigation for
    * problems described in {IERC20-approve}.
    *
    * Emits an {Approval} event indicating the updated allowance.
    *
    * 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 the caller.
    *
    * This is an alternative to {approve} that can be used as a mitigation for
    * problems described in {IERC20-approve}.
    *
    * Emits an {Approval} event indicating the updated allowance.
    *
    * 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 (bool) {
       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 address.
    *
    * 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 address.
    *
    * 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 `owner` s tokens.
    *
    * 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-hooks[Using Hooks].
     */
    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-hooks[Using Hooks].
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}
```

```
    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b > a) return (false, 0);
            return (true, a - b);
        }
    }
    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            // 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 (true, 0);
            uint256 c = a * b;
            if (c / a != b) return (false, 0);
            return (true, c);
```

```
// Dependency file: @openzeppelin/contracts/access/Ownable.sol
// pragma solidity ^0.8.0;
// import "@openzeppelin/contracts/utils/Context.sol";
/**
 * @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.
 */
abstract contract Ownable is Context {
    address private _owner;
    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor() {
        _setOwner(_msgSender());
    }
    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }
    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }
    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
```

```
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        _setOwner(address(0));
    }
    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _setOwner(newOwner);
    }
    function _setOwner(address newOwner) private {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}
// Dependency file: @openzeppelin/contracts/utils/math/SafeMath.sol
// pragma solidity ^0.8.0;
// CAUTION
// This version of SafeMath should only be used with Solidity 0.8 or later,
// because it relies on the compiler's built in overflow checks.
/**
 * @dev Wrappers over Solidity's arithmetic operations.
 *
 * NOTE: `SafeMath` is no longer needed starting with Solidity 0.8. The compiler
 * now has built in overflow checking.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }
```

```
        }
    }
    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a / b);
        }
    }
    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a % b);
        }
    }
    /**
```

```
     * @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) {
        return a + b;
    }
    /**
     * @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 a - b;
    }
    /**
```

```
    * @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) {
       return a * b;
   }
   /**
    * @dev Returns the integer division of two unsigned integers, reverting on
    * division by zero. The result is rounded towards zero.
    *
    * Counterpart to Solidity's `/` operator.
    *
    * Requirements:
    *
    * - The divisor cannot be zero.
    */
   function div(uint256 a, uint256 b) internal pure returns (uint256) {
       return a / b;
   }
   /**
    * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
    * reverting 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 a % b;
   }
   /**
    * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
    * overflow (when the result is negative).
    *
    * CAUTION: This function is deprecated because it requires allocating memory for the error
    * message unnecessarily. For custom revert reasons use {trySub}.
    *
    * Counterpart to Solidity's `-` operator.
    *
    * Requirements:
    *
    * - Subtraction cannot overflow.
    */
   function sub(
       uint256 a,
       uint256 b,
       string memory errorMessage
   ) internal pure returns (uint256) {
       unchecked {
           require(b <= a, errorMessage);
           return a - b;
       }
   }
   /**
    * @dev Returns the integer division of two unsigned integers, reverting 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) {
       unchecked {
           require(b > 0, errorMessage);
           return a / b;
       }
   }
   /**
    * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
    * reverting with custom message when dividing by zero.
    *
    * CAUTION: This function is deprecated because it requires allocating memory for the error
    * message unnecessarily. For custom revert reasons use {tryMod}.
    *
    * 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) {
       unchecked {
           require(b > 0, errorMessage);
           return a % b;
       }
   }
}
// Dependency file: @openzeppelin/contracts/proxy/Clones.sol
// pragma solidity ^0.8.0;
/**
 * @dev https://eips.ethereum.org/EIPS/eip-1167[EIP 1167] is a standard for
 * deploying minimal proxy contracts, also known as "clones".
 *
 * > To simply and cheaply clone contract functionality in an immutable way, this standard specifies
 * > a minimal bytecode implementation that delegates all calls to a known, fixed address.
 *
 * The library includes functions to deploy a proxy using either `create` (traditional deployment) or `create2`
 * (salted deterministic deployment). It also includes functions to predict the addresses of clones deployed using the
 * deterministic method.
 *
 * _Available since v3.4._
 */
library Clones {
   /**
    * @dev Deploys and returns the address of a clone that mimics the behaviour of `implementation`.
    *
    * This function uses the create opcode, which should never revert.
    */
   function clone(address implementation) internal returns (address instance) {
       assembly {
           let ptr := mload(0x40)
           mstore(ptr, 0x3d602d80600a3d3981f3363d3d373d3d3d363d73000000000000000000000000)
           mstore(add(ptr, 0x14), shl(0x60, implementation))
           mstore(add(ptr, 0x28), 0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000)
           instance := create(0, ptr, 0x37)
       }
       require(instance != address(0), "ERC1167: create failed");
   }
   /**
    * @dev Deploys and returns the address of a clone that mimics the behaviour of `implementation`.
    *
    * This function uses the create2 opcode and a `salt` to deterministically deploy
    * the clone. Using the same `implementation` and `salt` multiple time will revert, since
    * the clones cannot be deployed twice at the same address.
    */
   function cloneDeterministic(address implementation, bytes32 salt) internal returns (address instance) {
       assembly {
           let ptr := mload(0x40)
           mstore(ptr, 0x3d602d80600a3d3981f3363d3d373d3d3d363d73000000000000000000000000)
           mstore(add(ptr, 0x14), shl(0x60, implementation))
           mstore(add(ptr, 0x28), 0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000)
           instance := create2(0, ptr, 0x37, salt)
       }
       require(instance != address(0), "ERC1167: create2 failed");
   }
   /**
    * @dev Computes the address of a clone deployed using {Clones-cloneDeterministic}.
    */
   function predictDeterministicAddress(
       address implementation,
       bytes32 salt,
       address deployer
   ) internal pure returns (address predicted) {
       assembly {
           let ptr := mload(0x40)
           mstore(ptr, 0x3d602d80600a3d3981f3363d3d373d3d3d363d73000000000000000000000000)
           mstore(add(ptr, 0x14), shl(0x60, implementation))
           mstore(add(ptr, 0x28), 0x5af43d82803e903d91602b57fd5bf3ff0000000000000000000000000000)
           mstore(add(ptr, 0x38), shl(0x60, deployer))
           mstore(add(ptr, 0x4c), salt)
           mstore(add(ptr, 0x6c), keccak256(ptr, 0x37))
           predicted := keccak256(add(ptr, 0x37), 0x55)
       }
   }
   /**
    * @dev Computes the address of a clone deployed using {Clones-cloneDeterministic}.
    */
   function predictDeterministicAddress(address implementation, bytes32 salt)
       internal
       view
       returns (address predicted)
   {
       return predictDeterministicAddress(implementation, salt, address(this));
   }
}
// Dependency file: @openzeppelin/contracts/utils/Address.sol
// pragma solidity ^0.8.0;
/**
```

```
 * @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) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.
        uint256 size;
        assembly {
            size := extcodesize(account)
        }
        return size > 0;
    }
    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");
        (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");
        require(isContract(target), "Address: call to non-contract");
        (bool success, bytes memory returndata) = target.call{value: value}(data);
        return verifyCallResult(success, returndata, errorMessage);
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");
        (bool success, bytes memory returndata) = target.staticcall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }
    /**
     * @dev Tool to verifies that a low level call was successful, and revert if it wasn't, either by bubbling the
     * revert reason using the provided one.
     *
     * _Available since v4.3._
     */
    function verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) internal pure returns (bytes memory) {
        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
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}
```

```
// Dependency file: contracts/interfaces/IUniswapV2Factory.sol
// pragma solidity >=0.5.0;
interface IUniswapV2Factory {
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint256
    );
    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);
    function getPair(address tokenA, address tokenB)
        external
        view
        returns (address pair);
    function allPairs(uint256) external view returns (address pair);
    function allPairsLength() external view returns (uint256);
    function createPair(address tokenA, address tokenB)
        external
        returns (address pair);
    function setFeeTo(address) external;
    function setFeeToSetter(address) external;
}
// Dependency file: contracts/interfaces/IUniswapV2Router02.sol
// pragma solidity >=0.6.2;
interface IUniswapV2Router01 {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        );
    function addLiquidityETH(
        address token,
        uint256 amountTokenDesired,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        external
        payable
        returns (
            uint256 amountToken,
            uint256 amountETH,
            uint256 liquidity
        );
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountA, uint256 amountB);
    function removeLiquidityETH(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountToken, uint256 amountETH);
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountA, uint256 amountB);
    function removeLiquidityETHWithPermit(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountToken, uint256 amountETH);
    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);
    function swapTokensForExactTokens(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);
    function swapExactETHForTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable returns (uint256[] memory amounts);
    function swapTokensForExactETH(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);
    function swapExactTokensForETH(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);
    function swapETHForExactTokens(
        uint256 amountOut,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable returns (uint256[] memory amounts);
    function quote(
        uint256 amountA,
        uint256 reserveA,
        uint256 reserveB
    ) external pure returns (uint256 amountB);
    function getAmountOut(
        uint256 amountIn,
        uint256 reserveIn,
        uint256 reserveOut
    ) external pure returns (uint256 amountOut);
    function getAmountIn(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut
    ) external pure returns (uint256 amountIn);
    function getAmountsOut(uint256 amountIn, address[] calldata path)
        external
        view
        returns (uint256[] memory amounts);
    function getAmountsIn(uint256 amountOut, address[] calldata path)
        external
        view
        returns (uint256[] memory amounts);
}
interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountETH);
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountETH);
    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable;
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;
}
// Dependency file: @openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol
// pragma solidity ^0.8.0;
/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20Upgradeable {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);
    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);
    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);
    /**
     * @dev Returns the remaining number of tokens that `spender` will be
```

```
    * allowed to spend on behalf of `owner` through {transferFrom}. This is
    * zero by default.
    *
    * This value changes when {approve} or {transferFrom} are called.
    */
   function allowance(address owner, address spender) external view returns (uint256);
   /**
    * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
    *
    * Returns a boolean value indicating whether the operation succeeded.
    *
    * IMPORTANT: Beware that changing an allowance with this method brings the risk
    * that someone may use both the old and the new allowance by unfortunate
    * transaction ordering. One possible solution to mitigate this race
    * condition is to first reduce the spender's allowance to 0 and set the
    * desired value afterwards:
    * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
    *
    * Emits an {Approval} event.
    */
   function approve(address spender, uint256 amount) external returns (bool);
   /**
    * @dev Moves `amount` tokens from `sender` to `recipient` using the
    * allowance mechanism. `amount` is then deducted from the caller's
    * allowance.
    *
    * Returns a boolean value indicating whether the operation succeeded.
    *
    * Emits a {Transfer} event.
    */
   function transferFrom(
       address sender,
       address recipient,
       uint256 amount
   ) external returns (bool);
   /**
    * @dev Emitted when `value` tokens are moved from one account (`from`) to
    * another (`to`).
    *
    * Note that `value` may be zero.
    */
   event Transfer(address indexed from, address indexed to, uint256 value);
   /**
    * @dev Emitted when the allowance of a `spender` for an `owner` is set by
    * a call to {approve}. `value` is the new allowance.
    */
   event Approval(address indexed owner, address indexed spender, uint256 value);
}
// Dependency file: @openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol
// pragma solidity ^0.8.0;
// import "@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";
/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 *
 * _Available since v4.1._
 */
interface IERC20MetadataUpgradeable is IERC20Upgradeable {
   /**
    * @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);
}
// Dependency file: @openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol
// pragma solidity ^0.8.0;
/**
 * @dev This is a base contract to aid in writing upgradeable contracts, or any kind of contract that will be deployed
 * behind a proxy. Since a proxied contract can't have a constructor, it's common to move constructor logic to an
 * external initializer function, usually called `initialize`. It then becomes necessary to protect this initializer
 * function so it can only be called once. The {initializer} modifier provided by this contract will have this effect.
 *
 * TIP: To avoid leaving the proxy in an uninitialized state, the initializer function should be called as early as
 * possible by providing the encoded function call as the `_data` argument to {ERC1967Proxy-constructor}.
 *
 * CAUTION: When used with inheritance, manual care must be taken to not invoke a parent initializer twice, or to ensure
 * that all initializers are idempotent. This is not verified automatically as constructors are by Solidity.
 */
abstract contract Initializable {
   /**
    * @dev Indicates that the contract has been initialized.
    */
   bool private _initialized;
   /**
    * @dev Indicates that the contract is in the process of being initialized.
    */
   bool private _initializing;
   /**
    * @dev Modifier to protect an initializer function from being invoked twice.
    */
   modifier initializer() {
       require(_initializing || !_initialized, "Initializable: contract is already initialized");
       bool isTopLevelCall = !_initializing;
       if (isTopLevelCall) {
           _initializing = true;
           initialized = true;
       }
       _;
       if (isTopLevelCall) {
           _initializing = false;
       }
   }
}
// Dependency file: @openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol
// pragma solidity ^0.8.0;
// import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
/**
 * @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 meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract ContextUpgradeable is Initializable {
   function __Context_init() internal initializer {
       __Context_init_unchained();
   }
   function __Context_init_unchained() internal initializer {
   }
   function _msgSender() internal view virtual returns (address) {
       return msg.sender;
   }
   function _msgData() internal view virtual returns (bytes calldata) {
       return msg.data;
   }
   uint256[50] private __gap;
}
// Dependency file: @openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol
// pragma solidity ^0.8.0;
// import "@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";
// import "@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol";
// import "@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol";
// import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
/**
 * @dev Implementation of the {IERC20} 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 {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 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 {IERC20-approve}.
 */
contract ERC20Upgradeable is Initializable, ContextUpgradeable, IERC20Upgradeable, IERC20MetadataUpgradeable {
   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 be set once during
```

```
    * construction.
    */
    function __ERC20_init(string memory name_, string memory symbol_) internal initializer {
        __Context_init_unchained();
        __ERC20_init_unchained(name_, symbol_);
    }
    function __ERC20_init_unchained(string memory name_, string memory symbol_) internal initializer {
        _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 a shorter version of the
     * 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 should
     * be displayed to a user as `5.05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * 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, including
     * {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 (uint256) {
        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 allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * 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 the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * 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 the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * 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 (bool) {
        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 address.
     *
     * 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 address.
     *
     * 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 `owner` s tokens.
    *
    * 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-hooks[Using Hooks].
    */
   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-hooks[Using Hooks].
    */
   function _afterTokenTransfer(
       address from,
       address to,
       uint256 amount
   ) internal virtual {}
   uint256[45] private __gap;
}
// Dependency file: @openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol
// pragma solidity ^0.8.0;
// import "@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol";
// import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
/**
 * @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.
 */
abstract contract OwnableUpgradeable is Initializable, ContextUpgradeable {
   address private _owner;
   event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
   /**
    * @dev Initializes the contract setting the deployer as the initial owner.
    */
   function __Ownable_init() internal initializer {
       __Context_init_unchained();
       __Ownable_init_unchained();
   }
   function __Ownable_init_unchained() internal initializer {
       _setOwner(_msgSender());
   }
   /**
    * @dev Returns the address of the current owner.
    */
   function owner() public view virtual returns (address) {
       return _owner;
   }
   /**
    * @dev Throws if called by any account other than the owner.
    */
   modifier onlyOwner() {
       require(owner() == _msgSender(), "Ownable: caller is not the owner");
       _;
   }
   /**
    * @dev Leaves the contract without owner. It will not be possible to call
    * `onlyOwner` functions anymore. Can only be called by the current owner.
    *
    * NOTE: Renouncing ownership will leave the contract without an owner,
    * thereby removing any functionality that is only available to the owner.
    */
   function renounceOwnership() public virtual onlyOwner {
       _setOwner(address(0));
   }
   /**
    * @dev Transfers ownership of the contract to a new account (`newOwner`).
    * Can only be called by the current owner.
    */
   function transferOwnership(address newOwner) public virtual onlyOwner {
       require(newOwner != address(0), "Ownable: new owner is the zero address");
       _setOwner(newOwner);
   }
   function _setOwner(address newOwner) private {
       address oldOwner = _owner;
       _owner = newOwner;
       emit OwnershipTransferred(oldOwner, newOwner);
   }
   uint256[49] private __gap;
}
// Dependency file: contracts/interfaces/IUniswapV2Pair.sol
// pragma solidity >=0.5.0;
interface IUniswapV2Pair {
   event Approval(address indexed owner, address indexed spender, uint value);
   event Transfer(address indexed from, address indexed to, uint value);
   function name() external pure returns (string memory);
   function symbol() external pure returns (string memory);
   function decimals() external pure returns (uint8);
   function totalSupply() external view returns (uint);
   function balanceOf(address owner) external view returns (uint);
   function allowance(address owner, address spender) external view returns (uint);
   function approve(address spender, uint value) external returns (bool);
   function transfer(address to, uint value) external returns (bool);
   function transferFrom(address from, address to, uint value) external returns (bool);
   function DOMAIN_SEPARATOR() external view returns (bytes32);
   function PERMIT_TYPEHASH() external pure returns (bytes32);
   function nonces(address owner) external view returns (uint);
   function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;
   event Mint(address indexed sender, uint amount0, uint amount1);
   event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
   event Swap(
       address indexed sender,
       uint amount0In,
       uint amount1In,
       uint amount0Out,
       uint amount1Out,
       address indexed to
   );
   event Sync(uint112 reserve0, uint112 reserve1);
   function MINIMUM_LIQUIDITY() external pure returns (uint);
   function factory() external view returns (address);
   function token0() external view returns (address);
   function token1() external view returns (address);
   function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
   function price0CumulativeLast() external view returns (uint);
   function price1CumulativeLast() external view returns (uint);
   function kLast() external view returns (uint);
   function mint(address to) external returns (uint liquidity);
   function burn(address to) external returns (uint amount0, uint amount1);
```

```
    function swap(uint amount0Out, uint
amount1Out, address to, bytes calldata) ex-
ternal;
    function skim(address to) external;
    function sync() external;
    function initialize(address, address) external;
}
// Dependency file: contracts/libs/Safe-
MathInt.sol
// pragma solidity =0.8.4;
/**
 * @title SafeMathInt
 * @dev Math operations for int256 with over-
flow safety checks.
 */
library SafeMathInt {
    int256 private constant MIN_INT256 =
int256(1) << 255;
    int256 private constant MAX_INT256 =
~(int256(1) << 255);
    /**
     * @dev Multiplies two int256 variables and
fails on overflow.
     */
    function mul(int256 a, int256 b) internal pure
returns (int256) {
        int256 c = a * b;
        // Detect overflow when multiplying
MIN_INT256 with -1
        require(c != MIN_INT256 || (a &
MIN_INT256) != (b & MIN_INT256));
        require((b == 0) || (c / b == a));
        return c;
    }
    /**
     * @dev Division of two int256 variables and
fails on overflow.
     */
    function div(int256 a, int256 b) internal pure
returns (int256) {
        // Prevent overflow when dividing
MIN_INT256 by -1
        require(b != -1 || a != MIN_INT256);
        // Solidity already throws when dividing by
0.
        return a / b;
    }
    /**
     * @dev Subtracts two int256 variables and
fails on overflow.
     */
    function sub(int256 a, int256 b) internal pure
returns (int256) {
        int256 c = a - b;
        require((b >= 0 && c <= a) || (b < 0 && c
> a));
        return c;
    }
    /**
     * @dev Adds two int256 variables and fails
on overflow.
     */
    function add(int256 a, int256 b) internal pure
returns (int256) {
        int256 c = a + b;
        require((b >= 0 && c >= a) || (b < 0 && c
< a));
        return c;
    }
    /**
     * @dev Converts to absolute value, and fails
on overflow.
     */
    function abs(int256 a) internal pure returns
(int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }
    function toUint256Safe(int256 a) internal
pure returns (uint256) {
        require(a >= 0);
        return uint256(a);
    }
}
// Dependency file: contracts/libs/SafeMathU-
int.sol
// pragma solidity =0.8.4;
/**
 * @title SafeMathUint
 * @dev Math operations with safety checks
that revert on error
 */
library SafeMathUint {
    function toInt256Safe(uint256 a) internal pure
returns (int256) {
        int256 b = int256(a);
        require(b >= 0);
        return b;
    }
}
// Dependency file: contracts/baby/Iterable-
Mapping.sol
// pragma solidity =0.8.4;
library IterableMapping {
    // Iterable mapping from address to uint;
    struct Map {
        address[] keys;
        mapping(address => uint256) values;
        mapping(address => uint256) indexOf;
        mapping(address => bool) inserted;
    }
    function get(Map storage map, address key)
public view returns (uint256) {
        return map.values[key];
    }
    function getIndexOfKey(Map storage map,
address key)
        public
        view
        returns (int256)
    {
        if (!map.inserted[key]) {
            return -1;
        }
        return int256(map.indexOf[key]);
    }
    function getKeyAtIndex(Map storage map,
uint256 index)
        public
        view
        returns (address)
    {
        return map.keys[index];
    }
    function size(Map storage map) public view
returns (uint256) {
        return map.keys.length;
    }
    function set(
        Map storage map,
        address key,
        uint256 val
    ) public {
        if (map.inserted[key]) {
            map.values[key] = val;
        } else {
            map.inserted[key] = true;
            map.values[key] = val;
            map.indexOf[key] = map.keys.length;
            map.keys.push(key);
        }
    }
    function remove(Map storage map, address
key) public {
        if (!map.inserted[key]) {
            return;
        }
        delete map.inserted[key];
        delete map.values[key];
        uint256 index = map.indexOf[key];
        uint256 lastIndex = map.keys.length - 1;
        address lastKey = map.keys[lastIndex];
        map.indexOf[lastKey] = index;
        delete map.indexOf[key];
        map.keys[index] = lastKey;
        map.keys.pop();
    }
}
// Dependency file: contracts/baby/BabyToken-
DividendTracker.sol
// pragma solidity =0.8.4;
// import "@openzeppelin/contracts/to-
ken/ERC20/ERC20.sol";
// import "@openzeppelin/contracts-upgradea-
ble/token/ERC20/ERC20Upgradeable.sol";
// import "@openzeppelin/contracts/to-
ken/ERC20/IERC20.sol";
// import "@openzeppelin/contracts/ac-
cess/Ownable.sol";
// import "@openzeppelin/contracts-upgradea-
ble/access/OwnableUpgradeable.sol";
// import "@openzeppelin/con-
tracts/utils/math/SafeMath.sol";
// import "contracts/interfaces/IUniswapV2Fac-
tory.sol";
// import "contracts/interfaces/IUni-
swapV2Router02.sol";
// import "contracts/interfaces/IUni-
swapV2Pair.sol";
// import "contracts/libs/SafeMathInt.sol";
// import "contracts/libs/SafeMathUint.sol";
// import "contracts/baby/IterableMapping.sol";
/// @title Dividend-Paying Token Interface
/// @author Roger Wu (https://github.com/roger-
wu)
/// @dev An interface for a dividend-paying to-
ken contract.
interface DividendPayingTokenInterface {
    /// @notice View the amount of dividend in
wei that an address can withdraw.
    /// @param _owner The address of a token
holder.
    /// @return The amount of dividend in wei
that `_owner` can withdraw.
    function dividendOf(address _owner) exter-
nal view returns (uint256);
    /// @notice Withdraws the ether distributed to
the sender.
    /// @dev SHOULD transfer `divi-
dendOf(msg.sender)` wei to `msg.sender`, and
`dividendOf(msg.sender)` SHOULD be 0 after
the transfer.
    ///  MUST emit a `DividendWithdrawn` event
if the amount of ether transferred is greater than
0.
    function withdrawDividend() external;
    /// @dev This event MUST emit when ether is
distributed to token holders.
    /// @param from The address which sends
ether to this contract.
    /// @param weiAmount The amount of dis-
tributed ether in wei.
    event DividendsDistributed(address indexed
from, uint256 weiAmount);
    /// @dev This event MUST emit when an ad-
dress withdraws their dividend.
    /// @param to The address which withdraws
ether from this contract.
    /// @param weiAmount The amount of with-
drawn ether in wei.
    event DividendWithdrawn(address indexed
to, uint256 weiAmount);
}
/// @title Dividend-Paying Token Optional In-
terface
/// @author Roger Wu (https://github.com/roger-
wu)
/// @dev OPTIONAL functions for a dividend-
paying token contract.
interface DividendPayingTokenOptionalInter-
face {
    /// @notice View the amount of dividend in
wei that an address can withdraw.
    /// @param _owner The address of a token
holder.
    /// @return The amount of dividend in wei
that `  owner` can withdraw.
    function withdrawableDividendOf(address
 owner)
        external
        view
        returns (uint256);
    /// @notice View the amount of dividend in
wei that an address has withdrawn.
    /// @param _owner The address of a token
holder.
    /// @return The amount of dividend in wei
that `_owner` has withdrawn.
    function withdrawnDividendOf(address
_owner)
        external
        view
        returns (uint256);
    /// @notice View the amount of dividend in
wei that an address has earned in total.
    /// @dev accumulativeDividendOf(_owner) =
withdrawableDividendOf(_owner) + with-
drawnDividendOf(_owner)
```

```
    /// @param _owner The address of a token holder.
    /// @return The amount of dividend in wei that `_owner` has earned in total.
    function accumulativeDividendOf(address _owner)
        external
        view
        returns (uint256);
}
/// @title Dividend-Paying Token
/// @author Roger Wu (https://github.com/roger-wu)
/// @dev A mintable ERC20 token that allows anyone to pay and distribute ether
///  to token holders as dividends and allows token holders to withdraw their dividends.
///  Reference: the source code of PoWH3D: https://etherscan.io/address/0xB3775fB83F7D12A36E0475aBdD1FCA35c091efBe#code
contract DividendPayingToken is
    ERC20Upgradeable,
    OwnableUpgradeable,
    DividendPayingTokenInterface,
    DividendPayingTokenOptionalInterface
{
    using SafeMath for uint256;
    using SafeMathUint for uint256;
    using SafeMathInt for int256;
    address public rewardToken;
    // With `magnitude`, we can properly distribute dividends even if the amount of received ether is small.
    // For more discussion about choosing the value of `magnitude`,
    //  see https://github.com/ethereum/EIPs/issues/1726#issuecomment-472352728
    uint256 internal constant magnitude = 2**128;
    uint256 internal magnifiedDividendPerShare;
    // About dividendCorrection:
    // If the token balance of a `_user` is never changed, the dividend of `_user` can be computed with:
    //   `dividendOf(_user) = dividendPerShare * balanceOf(_user)`.
    // When `balanceOf(_user)` is changed (via minting/burning/transferring tokens),
    //   `dividendOf(_user)` should not be changed,
    //   but the computed value of `dividendPerShare * balanceOf(_user)` is changed.
    // To keep the `dividendOf(_user)` unchanged, we add a correction term:
    //   `dividendOf(_user) = dividendPerShare * balanceOf(_user) + dividendCorrectionOf(_user)`,
    //   where `dividendCorrectionOf(_user)` is updated whenever `balanceOf(_user)` is changed:
    //   `dividendCorrectionOf(_user) = dividendPerShare * (old balanceOf(_user)) - (new balanceOf(_user))`.
    // So now `dividendOf(_user)` returns the same value before and after `balanceOf(_user)` is changed.
    mapping(address => int256) internal magnifiedDividendCorrections;
    mapping(address => uint256) internal withdrawnDividends;
    uint256 public totalDividendsDistributed;
    function __DividendPayingToken_init(
        address _rewardToken,
        string memory _name,
        string memory _symbol
    ) internal initializer {
        __Ownable_init();
        __ERC20_init(_name, _symbol);
        rewardToken = _rewardToken;
    }
    function distributeCAKEDividends(uint256 amount) public onlyOwner {
        require(totalSupply() > 0);
        if (amount > 0) {
            magnifiedDividendPerShare = magnifiedDividendPerShare.add(
                (amount).mul(magnitude) / totalSupply()
            );
            emit DividendsDistributed(msg.sender, amount);
            totalDividendsDistributed = totalDividendsDistributed.add(amount);
        }
    }
    /// @notice Withdraws the ether distributed to the sender.
    /// @dev It emits a `DividendWithdrawn` event if the amount of withdrawn ether is greater than 0.
    function withdrawDividend() public virtual override {
        _withdrawDividendOfUser(payable(msg.sender));
    }
    /// @notice Withdraws the ether distributed to the sender.
    /// @dev It emits a `DividendWithdrawn` event if the amount of withdrawn ether is greater than 0.
    function _withdrawDividendOfUser(address payable user)
        internal
        returns (uint256)
    {
        uint256 _withdrawableDividend = withdrawableDividendOf(user);
        if (_withdrawableDividend > 0) {
            withdrawnDividends[user] = withdrawnDividends[user].add(
                _withdrawableDividend
            );
            emit DividendWithdrawn(user, _withdrawableDividend);
            bool success = IERC20(rewardToken).transfer(
                user,
                _withdrawableDividend
            );
            if (!success) {
                withdrawnDividends[user] = withdrawnDividends[user].sub(
                    _withdrawableDividend
                );
                return 0;
            }
            return _withdrawableDividend;
        }
        return 0;
    }
    /// @notice View the amount of dividend in wei that an address can withdraw.
    /// @param _owner The address of a token holder.
    /// @return The amount of dividend in wei that `_owner` can withdraw.
    function dividendOf(address _owner) public view override returns (uint256) {
        return withdrawableDividendOf(_owner);
    }
    /// @notice View the amount of dividend in wei that an address can withdraw.
    /// @param _owner The address of a token holder.
    /// @return The amount of dividend in wei that `_owner` can withdraw.
    function withdrawableDividendOf(address _owner)
        public
        view
        override
        returns (uint256)
    {
        return accumulativeDividendOf(_owner).sub(withdrawnDividends[_owner]);
    }
    /// @notice View the amount of dividend in wei that an address has withdrawn.
    /// @param _owner The address of a token holder.
    /// @return The amount of dividend in wei that `_owner` has withdrawn.
    function withdrawnDividendOf(address _owner)
        public
        view
        override
        returns (uint256)
    {
        return withdrawnDividends[_owner];
    }
    /// @notice View the amount of dividend in wei that an address has earned in total.
    /// @dev accumulativeDividendOf(_owner) = withdrawableDividendOf(_owner) + withdrawnDividendOf(_owner)
    /// = (magnifiedDividendPerShare * balanceOf(_owner) + magnifiedDividendCorrections[_owner]) / magnitude
    /// @param _owner The address of a token holder.
    /// @return The amount of dividend in wei that `_owner` has earned in total.
    function accumulativeDividendOf(address _owner)
        public
        view
        override
        returns (uint256)
    {
        return
            magnifiedDividendPerShare
                .mul(balanceOf(_owner))
                .toInt256Safe()
                .add(magnifiedDividendCorrections[ owner])
                .toUint256Safe() / magnitude;
    }
    /// @dev Internal function that transfer tokens from one address to another.
    /// Update magnifiedDividendCorrections to keep dividends unchanged.
    /// @param from The address to transfer from.
    /// @param to The address to transfer to.
    /// @param value The amount to be transferred.
    function _transfer(
        address from,
        address to,
        uint256 value
    ) internal virtual override {
        require(false);
        int256  magCorrection = magnifiedDividendPerShare
            .mul(value)
            .toInt256Safe();
        magnifiedDividendCorrections[from] = magnifiedDividendCorrections[from]
            .add(_magCorrection);
        magnifiedDividendCorrections[to] = magnifiedDividendCorrections[to].sub(
            _magCorrection
        );
    }
    /// @dev Internal function that mints tokens to an account.
    /// Update magnifiedDividendCorrections to keep dividends unchanged.
    /// @param account The account that will receive the created tokens.
    /// @param value The amount that will be created.
    function _mint(address account, uint256 value) internal override {
        super._mint(account, value);
        magnifiedDividendCorrections[account] = magnifiedDividendCorrections[
            account
        ].sub((magnifiedDividendPerShare.mul(value)).toInt256Safe());
    }
    /// @dev Internal function that burns an amount of the token of a given account.
    /// Update magnifiedDividendCorrections to keep dividends unchanged.
    /// @param account The account whose tokens will be burnt.
    /// @param value The amount that will be burnt.
    function _burn(address account, uint256 value) internal override {
        super._burn(account, value);
        magnifiedDividendCorrections[account] = magnifiedDividendCorrections[
            account
        ].add((magnifiedDividendPerShare.mul(value)).toInt256Safe());
    }
```

```
    function _setBalance(address account,
uint256 newBalance) internal {
        uint256 currentBalance = balanceOf(ac-
count);
        if (newBalance > currentBalance) {
            uint256 mintAmount = new-
Balance.sub(currentBalance);
            _mint(account, mintAmount);
        } else if (newBalance < currentBalance) {
            uint256 burnAmount = currentBal-
ance.sub(newBalance);
            _burn(account, burnAmount);
        }
    }
}
contract BABYTOKENDividendTracker is
OwnableUpgradeable, DividendPayingToken {
    using SafeMath for uint256;
    using SafeMathInt for int256;
    using IterableMapping for Iterable-
Mapping.Map;
    IterableMapping.Map private tokenHolders-
Map;
    uint256 public lastProcessedIndex;
    mapping(address => bool) public exclud-
edFromDividends;
    mapping(address => uint256) public
lastClaimTimes;
    uint256 public claimWait;
    uint256 public minimumTokenBalance-
ForDividends;
    event ExcludeFromDividends(address in-
dexed account);
    event ClaimWaitUpdated(uint256 indexed
newValue, uint256 indexed oldValue);
    event Claim(
        address indexed account,
        uint256 amount,
        bool indexed automatic
    );
    function initialize(
        address rewardToken_,
        uint256 minimumTokenBalanceForDivi-
dends_
    ) external initializer {
        DividendPayingToken.__DividendPay-
ingToken_init(
            rewardToken_,
            "DIVIDEND_TRACKER",
            "DIVIDEND_TRACKER"
        );
        claimWait = 3600;
        minimumTokenBalanceForDividends =
minimumTokenBalanceForDividends_;
    }
    function _transfer(
        address,
        address,
        uint256
    ) internal pure override {
        require(false, "Dividend_Tracker: No
transfers allowed");
    }
    function withdrawDividend() public pure
override {
        require(
            false,
            "Dividend  Tracker: withdrawDividend
disabled. Use the 'claim' function on the main
BABYTOKEN contract."
        );
    }
    function excludeFromDividends(address ac-
count) external onlyOwner {
        require(!excludedFromDividends[ac-
count]);
        excludedFromDividends[account] = true;
        _setBalance(account, 0);
        tokenHoldersMap.remove(account);
        emit ExcludeFromDividends(account);
    }
    function isExcludedFromDividends(address
account)
        public
        view
        returns (bool)
    {
        return excludedFromDividends[account];
    }
    function updateClaimWait(uint256 new-
ClaimWait) external onlyOwner {
        require(
            newClaimWait >= 3600 && newClaim-
Wait <= 86400,
            "Dividend_Tracker: claimWait must be
updated to between 1 and 24 hours"
        );
        require(
            newClaimWait != claimWait,
            "Dividend_Tracker: Cannot update
claimWait to same value"
        );
        emit ClaimWaitUpdated(newClaimWait,
claimWait);
        claimWait = newClaimWait;
    }
    function updateMinimumTokenBalance-
ForDividends(uint256 amount)
        external
        onlyOwner
    {
        minimumTokenBalanceForDividends =
amount;
    }
    function getLastProcessedIndex() external
view returns (uint256) {
        return lastProcessedIndex;
    }
    function getNumberOfTokenHolders() exter-
nal view returns (uint256) {
        return tokenHoldersMap.keys.length;
    }
    function getAccount(address _account)
        public
        view
        returns (
            address account,
            int256 index,
            int256 iterationsUntilProcessed,
            uint256 withdrawableDividends,
            uint256 totalDividends,
            uint256 lastClaimTime,
            uint256 nextClaimTime,
            uint256 secondsUntilAutoClaimAvaila-
ble
        )
    {
        account = _account;
        index = tokenHoldersMap.getIndexOf-
Key(account);
        iterationsUntilProcessed = -1;
        if (index >= 0) {
            if (uint256(index) > lastProcessedIndex)
{
                iterationsUntilProcessed = index.sub(
                    int256(lastProcessedIndex)
                );
            } else {
                uint256 processesUntilEndOfArray =
tokenHoldersMap.keys.length >
                    lastProcessedIndex
                    ? tokenHolders-
Map.keys.length.sub(lastProcessedIndex)
                    : 0;
                iterationsUntilProcessed = index.add(
                    int256(processesUntilEndOfArray)
                );
            }
        }
        withdrawableDividends = withdrawableDi-
videndOf(account);
        totalDividends = accumulative-
DividendOf(account);
        lastClaimTime = lastClaimTimes[account];
        nextClaimTime = lastClaimTime > 0 ?
lastClaimTime.add(claimWait) : 0;
        secondsUntilAutoClaimAvailable =
nextClaimTime > block.timestamp
            ? nextClaimTime.sub(block.timestamp)
            : 0;
    }
    function getAccountAtIndex(uint256 index)
        public
        view
        returns (
            address,
            int256,
            int256,
            uint256,
            uint256,
            uint256,
            uint256,
            uint256
        )
    {
        if (index >= tokenHoldersMap.size()) {
            return (address(0), -1, -1, 0, 0, 0, 0, 0);
        }
        address account = tokenHoldersMap.get-
KeyAtIndex(index);
        return getAccount(account);
    }
    function canAutoClaim(uint256
lastClaimTime) private view returns (bool) {
        if (lastClaimTime > block.timestamp) {
            return false;
        }
        return block.timestamp.sub(lastClaimTime)
>= claimWait;
    }
    function setBalance(address payable account,
uint256 newBalance)
        external
        onlyOwner
    {
        if (excludedFromDividends[account]) {
            return;
        }
        if (newBalance >= minimumTokenBal-
anceForDividends) {
            _setBalance(account, newBalance);
            tokenHoldersMap.set(account, new-
Balance);
        } else {
            _setBalance(account, 0);
            tokenHoldersMap.remove(account);
        }
        processAccount(account, true);
    }
    function process(uint256 gas)
        public
        returns (
            uint256,
            uint256,
            uint256
        )
    {
        uint256 numberOfTokenHolders = to-
kenHoldersMap.keys.length;
        if (numberOfTokenHolders == 0) {
            return (0, 0, lastProcessedIndex);
        }
        uint256 _lastProcessedIndex = lastPro-
cessedIndex;
        uint256 gasUsed = 0;
        uint256 gasLeft = gasleft();
        uint256 iterations = 0;
        uint256 claims = 0;
        while (gasUsed < gas && iterations <
numberOfTokenHolders) {
            lastProcessedIndex++;
            if (_lastProcessedIndex >= tokenHold-
ersMap.keys.length) {
                _lastProcessedIndex = 0;
            }
            address account = tokenHolders-
Map.keys[  lastProcessedIndex];
            if (canAutoClaim(lastClaimTimes[ac-
count])) {
                if (processAccount(payable(account),
true)) {
                    claims++;
                }
            }
            iterations++;
            uint256 newGasLeft = gasleft();
            if (gasLeft > newGasLeft) {
                gasUsed = gasUsed.add(gas-
Left.sub(newGasLeft));
            }
            gasLeft = newGasLeft;
        }
        lastProcessedIndex = _lastProcessedIndex;
        return (iterations, claims, lastProcessedIn-
dex);
    }
    function processAccount(address payable ac-
count, bool automatic)
        public
```

```
        onlyOwner
        returns (bool)
    {
        uint256 amount = _withdrawDividendOfUser(account);
        if (amount > 0) {
            lastClaimTimes[account] = block.timestamp;
            emit Claim(account, amount, automatic);
            return true;
        }
        return false;
    }
}
// Dependency file: contracts/BaseToken.sol
// pragma solidity =0.8.4;
enum TokenType {
    standard,
    antiBotStandard,
    liquidityGenerator,
    antiBotLiquidityGenerator,
    baby,
    antiBotBaby,
    buybackBaby,
    antiBotBuybackBaby
}
abstract contract BaseToken {
    event TokenCreated(
        address indexed owner,
        address indexed token,
        TokenType tokenType,
        uint256 version
    );
}
// Root file: contracts/baby/BabyToken.sol
pragma solidity =0.8.4;
// import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
// import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
// import "@openzeppelin/contracts/access/Ownable.sol";
// import "@openzeppelin/contracts/utils/math/SafeMath.sol";
// import "@openzeppelin/contracts/proxy/Clones.sol";
// import "@openzeppelin/contracts/utils/Address.sol";
// import "contracts/interfaces/IUniswapV2Factory.sol";
// import "contracts/interfaces/IUniswapV2Router02.sol";
// import "contracts/baby/BabyTokenDividendTracker.sol";
// import "contracts/BaseToken.sol";
contract BABYTOKEN is ERC20, Ownable, BaseToken {
    using SafeMath for uint256;
    using Address for address;
    using Address for address payable;
    uint256 public constant VERSION = 3;
    IUniswapV2Router02 public uniswapV2Router;
    address public uniswapV2Pair;
    bool private swapping;
    BABYTOKENDividendTracker public dividendTracker;
    address public rewardToken;
    uint256 public swapTokensAtAmount;
    uint256 public tokenRewardsFee;
    uint256 public liquidityFee;
    uint256 public marketingFee;
    uint256 public totalFees;
    address public _marketingWalletAddress;
    uint256 public gasForProcessing;
    // exlcude from fees and max transaction amount
    mapping(address => bool) private _isExcludedFromFees;
    // store addresses that a automatic market maker pairs. Any transfer *to* these addresses
    // could be subject to a maximum transfer amount
    mapping(address => bool) public automatedMarketMakerPairs;
    event ExcludeFromFees(address indexed account);
    event ExcludeMultipleAccountsFromFees(address[] accounts);
    event SetAutomatedMarketMakerPair(address indexed pair, bool indexed value);
    event GasForProcessingUpdated(
        uint256 indexed newValue,
        uint256 indexed oldValue
    );
    event SwapAndLiquify(
        uint256 tokensSwapped,
        uint256 ethReceived,
        uint256 tokensIntoLiqudity
    );
    event SendDividends(uint256 tokensSwapped, uint256 amount);
    event ProcessedDividendTracker(
        uint256 iterations,
        uint256 claims,
        uint256 lastProcessedIndex,
        bool indexed automatic,
        uint256 gas,
        address indexed processor
    );
    constructor(
        string memory name_,
        string memory symbol_,
        uint256 totalSupply_,
        address[4] memory addrs, // reward, router, marketing wallet, dividendTracker
        uint256[3] memory feeSettings, // rewards, liquidity, marketing
        uint256 minimumTokenBalanceForDividends_,
        address serviceFeeReceiver_,
        uint256 serviceFee_
    ) payable ERC20(name_, symbol_) {
        rewardToken = addrs[0];
        _marketingWalletAddress = addrs[2];
        require(
            msg.sender != _marketingWalletAddress,
            "Owner and marketing wallet cannot be the same"
        );
        require(
            !_marketingWalletAddress.isContract(),
            "Marketing wallet cannot be a contract"
        );
        tokenRewardsFee = feeSettings[0];
        liquidityFee = feeSettings[1];
        marketingFee = feeSettings[2];
        totalFees = tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
        swapTokensAtAmount = totalSupply_.div(1000); // 0.1%
        // use by default 300,000 gas to process auto-claiming dividends
        gasForProcessing = 300000;
        dividendTracker = BABYTOKENDividendTracker(
            payable(Clones.clone(addrs[3]))
        );
        dividendTracker.initialize(
            rewardToken,
            minimumTokenBalanceForDividends_
        );
        IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(addrs[1]);
        // Create a uniswap pair for this new token
        address  uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this), _uniswapV2Router.WETH());
        uniswapV2Router = _uniswapV2Router;
        uniswapV2Pair = _uniswapV2Pair;
        _setAutomatedMarketMakerPair(_uniswapV2Pair, true);
        // exclude from receiving dividends
        dividendTracker.excludeFromDividends(address(dividendTracker));
        dividendTracker.excludeFromDividends(address(this));
        dividendTracker.excludeFromDividends(owner());
        dividendTracker.excludeFromDividends(address(0xdead));
        dividendTracker.excludeFromDividends(address(_uniswapV2Router));
        // exclude from paying fees or having max transaction amount
        _isExcludedFromFees[owner()] = true;
        _isExcludedFromFees[_marketingWalletAddress] = true;
        _isExcludedFromFees[address(this)] = true;
        /*
            _mint is an internal function in ERC20.sol that is only called here,
            and CANNOT be called ever again
        */
        _mint(owner(), totalSupply_);
        emit TokenCreated(owner(), address(this), TokenType.baby, VERSION);
        payable(serviceFeeReceiver_).transfer(serviceFee_);
    }
    receive() external payable {}
    function setSwapTokensAtAmount(uint256 amount) external onlyOwner {
        require(
            amount > totalSupply() / 10**5,
            "BABYTOKEN: Amount must be greater than 0.001% of total supply"
        );
        swapTokensAtAmount = amount;
    }
    function excludeFromFees(address account) external onlyOwner {
        require(
            !_isExcludedFromFees[account],
            "BABYTOKEN: Account is already excluded"
        );
        _isExcludedFromFees[account] = true;
        emit ExcludeFromFees(account);
    }
    function excludeMultipleAccountsFromFees(address[] calldata accounts)
        external
        onlyOwner
    {
        for (uint256 i = 0; i < accounts.length; i++) {
            _isExcludedFromFees[accounts[i]] = true;
        }
        emit ExcludeMultipleAccountsFromFees(accounts);
    }
    function setMarketingWallet(address payable wallet) external onlyOwner {
        require(
            wallet != address(0),
            "BABYTOKEN: The marketing wallet cannot be the value of zero"
        );
        require(!wallet.isContract(), "Marketing wallet cannot be a contract");
        _marketingWalletAddress = wallet;
    }
    function setTokenRewardsFee(uint256 value) external onlyOwner {
        tokenRewardsFee = value;
        totalFees = tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
    }
    function setLiquiditFee(uint256 value) external onlyOwner {
        liquidityFee = value;
        totalFees = tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
    }
    function setMarketingFee(uint256 value) external onlyOwner {
        marketingFee = value;
        totalFees = tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
    }
    function  setAutomatedMarketMakerPair(address pair, bool value) private {
        require(
            automatedMarketMakerPairs[pair] != value,
```

```
        "BABYTOKEN: Automated market maker pair is already set to that value"
        );
        automatedMarketMakerPairs[pair] = value;
        if (value) {
            dividendTracker.excludeFromDividends(pair);
        }
        emit SetAutomatedMarketMakerPair(pair, value);
    }
    function updateGasForProcessing(uint256 newValue) public onlyOwner {
        require(
            newValue >= 200000 && newValue <= 500000,
            "BABYTOKEN: gasForProcessing must be between 200,000 and 500,000"
        );
        require(
            newValue != gasForProcessing,
            "BABYTOKEN: Cannot update gasForProcessing to same value"
        );
        emit GasForProcessingUpdated(newValue, gasForProcessing);
        gasForProcessing = newValue;
    }
    function updateClaimWait(uint256 claimWait) external onlyOwner {
        dividendTracker.updateClaimWait(claimWait);
    }
    function getClaimWait() external view returns (uint256) {
        return dividendTracker.claimWait();
    }
    function updateMinimumTokenBalanceForDividends(uint256 amount)
        external
        onlyOwner
    {
        dividendTracker.updateMinimumTokenBalanceForDividends(amount);
    }
    function getMinimumTokenBalanceForDividends()
        external
        view
        returns (uint256)
    {
        return dividendTracker.minimumTokenBalanceForDividends();
    }
    function getTotalDividendsDistributed() external view returns (uint256) {
        return dividendTracker.totalDividendsDistributed();
    }
    function isExcludedFromFees(address account) public view returns (bool) {
        return  isExcludedFromFees[account];
    }
    function withdrawableDividendOf(address account)
        public
        view
        returns (uint256)
    {
        return dividendTracker.withdrawableDividendOf(account);
    }
    function dividendTokenBalanceOf(address account)
        public
        view
        returns (uint256)
    {
        return dividendTracker.balanceOf(account);
    }
    function excludeFromDividends(address account) external onlyOwner {
        dividendTracker.excludeFromDividends(account);
    }
    function isExcludedFromDividends(address account)
        public
        view
        returns (bool)
    {
        return dividendTracker.isExcludedFromDividends(account);
    }
    function getAccountDividendsInfo(address account)
        external
        view
        returns (
            address,
            int256,
            int256,
            uint256,
            uint256,
            uint256,
            uint256,
            uint256
        )
    {
        return dividendTracker.getAccount(account);
    }
    function getAccountDividendsInfoAtIndex(uint256 index)
        external
        view
        returns (
            address,
            int256,
            int256,
            uint256,
            uint256,
            uint256,
            uint256,
            uint256
        )
    {
        return dividendTracker.getAccountAtIndex(index);
    }
    function processDividendTracker(uint256 gas) external {
        (
            uint256 iterations,
            uint256 claims,
            uint256 lastProcessedIndex
        ) = dividendTracker.process(gas);
        emit ProcessedDividendTracker(
            iterations,
            claims,
            lastProcessedIndex,
            false,
            gas,
            tx.origin
        );
    }
    function claim() external {
        dividendTracker.processAccount(payable(msg.sender), false);
    }
    function getLastProcessedIndex() external view returns (uint256) {
        return dividendTracker.getLastProcessedIndex();
    }
    function getNumberOfDividendTokenHolders() external view returns (uint256) {
        return dividendTracker.getNumberOfTokenHolders();
    }
    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");
        if (amount == 0) {
            super._transfer(from, to, 0);
            return;
        }
        uint256 contractTokenBalance = balanceOf(address(this));
        bool canSwap = contractTokenBalance >= swapTokensAtAmount;
        if (
            canSwap &&
            !swapping &&
            !automatedMarketMakerPairs[from] &&
            from != owner() &&
            to != owner() &&
            totalFees > 0
        ) {
            swapping = true;
            if (marketingFee > 0) {
                uint256 marketingTokens = contractTokenBalance
                    .mul(marketingFee)
                    .div(totalFees);
                swapAndSendToFee(marketingTokens);
            }
            if (liquidityFee > 0) {
                uint256 swapTokens = contractTokenBalance.mul(liquidityFee).div(
                    totalFees
                );
                swapAndLiquify(swapTokens);
            }
            uint256 sellTokens = balanceOf(address(this));
            if (sellTokens > 0) {
                swapAndSendDividends(sellTokens);
            }
            swapping = false;
        }
        bool takeFee = !swapping;
        // if any account belongs to _isExcludedFromFee account then remove the fee
        if (_isExcludedFromFees[from] || _isExcludedFromFees[to]) {
            takeFee = false;
        }
        if (takeFee && totalFees > 0) {
            uint256 fees = amount.mul(totalFees).div(100);
            amount = amount.sub(fees);
            super._transfer(from, address(this), fees);
        }
        super._transfer(from, to, amount);
        try
            dividendTracker.setBalance(payable(from), balanceOf(from))
        {} catch {}
        try dividendTracker.setBalance(payable(to), balanceOf(to)) {} catch {}
        if (!swapping) {
            uint256 gas = gasForProcessing;
            try dividendTracker.process(gas) returns (
                uint256 iterations,
                uint256 claims,
                uint256 lastProcessedIndex
            ) {
                emit ProcessedDividendTracker(
                    iterations,
                    claims,
                    lastProcessedIndex,
                    true,
                    gas,
                    tx.origin
                );
            } catch {}
        }
    }
    function swapAndSendToFee(uint256 tokens) private {
        uint256 initialCAKEBalance = IERC20(rewardToken).balanceOf(
            address(this)
        );
        swapTokensForCake(tokens);
        uint256 newBalance = (IERC20(rewardToken).balanceOf(address(this))).sub(
            initialCAKEBalance
        );
        IERC20(rewardToken).transfer(_marketingWalletAddress, newBalance);
    }
    function swapAndLiquify(uint256 tokens) private {
        // split the contract balance into halves
        uint256 half = tokens.div(2);
        uint256 otherHalf = tokens.sub(half);
        // capture the contract's current ETH balance.
```

```
        // this is so that we can capture exactly the amount of ETH that the
        // swap creates, and not make the liquidity event include any ETH that
        // has been manually sent to the contract
        uint256 initialBalance = address(this).balance;
        // swap tokens for ETH
        swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify is triggered
        // how much ETH did we just swap into?
        uint256 newBalance = address(this).balance.sub(initialBalance);
        // add liquidity to uniswap
        addLiquidity(otherHalf, newBalance);
        emit SwapAndLiquify(half, newBalance, otherHalf);
    }
    function swapTokensForEth(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        _approve(address(this), address(uniswapV2Router), tokenAmount);
        // make the swap
        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );
    }
    function swapTokensForCake(uint256 tokenAmount) private {
        address[] memory path = new address[](3);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        path[2] = rewardToken;
        _approve(address(this), address(uniswapV2Router), tokenAmount);
        // make the swap
        uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
    }
    function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router), tokenAmount);
        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            address(0xdead),
            block.timestamp
        );
    }
    function swapAndSendDividends(uint256 tokens) private {
        swapTokensForCake(tokens);
        uint256 dividends = IERC20(rewardToken).balanceOf(address(this));
        bool success = IERC20(rewardToken).transfer(
            address(dividendTracker),
            dividends
        );
        if (success) {
            dividendTracker.distributeCAKEDividends(dividends);
            emit SendDividends(tokens, dividends);
        }
    }
```

Analysis: Basic Coding Bugs

1. **Semantic Consistency Checks**
- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: Not found
- Severity: Critical

2. **Redundant Fallback Function**
- Description: Whether the contract has a redundant fallback function.
- Result: Not found
- Severity: Critical

3. **Constructor Mismatch**
- Description: Whether the contract name and its constructor are not identical to each other.
- Result: Not found
- Severity: Critical

4. **Ownership Takeover**
- Description: Whether the set owner function is not protected.
- Result: Not found
- Severity: Critical

5. **Overflows & Underflows**
- Description: Whether the contract has general overflow o underflow vulnerabilities.
- Result: Not found
- Severity: Critical

6. **Reentrancy**
- Description: Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
- Result: Not found
- Severity: Critical

7. **Blackhole**
- Description: Whether the contract locks ETH indefinitely: merely in without out.
- Result: Not found
- Severity: High

8. **Money-Giving Bug**
- Description: Whether the contract returns funds to an arbitrary address.
- Result: Not found
- Severity: High

9. **Unauthorized Self-Destruct**
- Description: Whether the contract can be killed by any arbitrary address.
- Result: Not found
- Severity: Medium

10. **Unchecked External Call**
- Description: Whether the contract has any external call without checking the return value.
- Result: Not found
- Severity: Medium

11. Revert DoS
- Description: Whether the contract is vulnerable to DoS attack because of unexpected revert.
- Result: Not found
- Severity: Medium

12. Gasless Send
- Description: Whether the contract is vulnerable to gasless send.
- Result: Not found
- Severity: Medium

13. Send Instead Of Transfer
- Description: Whether the contract uses send instead of transfer.
- Result: Not found
- Severity: Medium

14. Use Of Untrusted Libraries
- Description: Whether the contract use any suspicious libraries.
- Result: Not found
- Severity: Medium

15. Costly Loop
- Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
- Result: Not found
- Severity: Medium

16. Use Of Predictable Variables
- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: Not found
- Severity: Medium

17. Deprecated Uses
- Description: Whether the contract use the deprecated Tx. Origin to perform the authorization.
- Result: Not found
- Severity: Medium

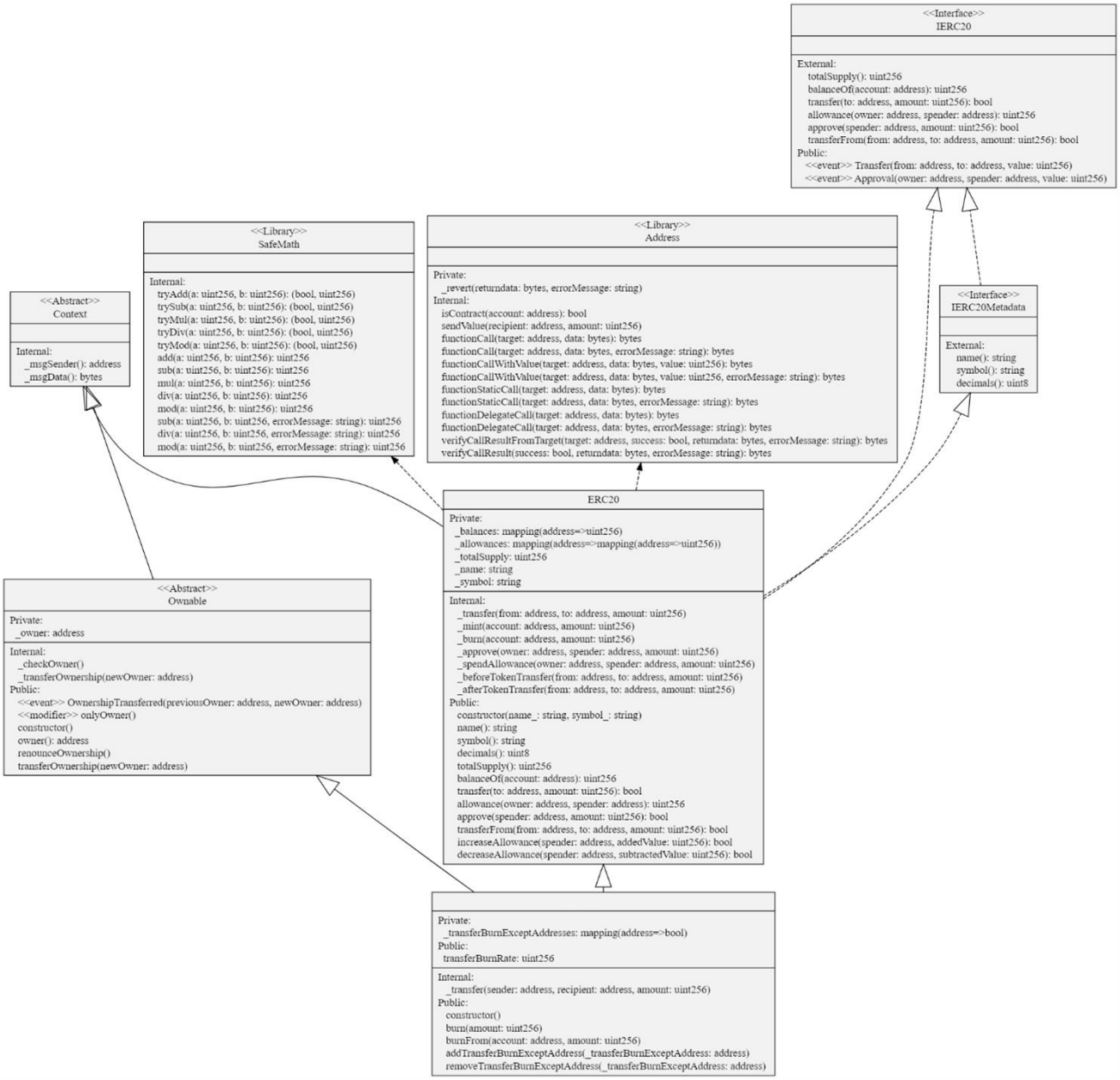
18. Transaction Ordering Dependence
- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Not found
- Severity: Medium

19. Make Type Inference Explicit
- Description: Do not use keyword var to specify the type, i.e., it asks the compiler to deduce the type, which is not safe especially in a loop.
- Result: Not found
- Severity: Low

20. Make Visibility Level Explicit
- Description: Assign explicit visibility specifiers for functions and state variables.
- Result: Not found
- Severity: Low

21. Avoid Use of Variadic Byte Array
- Description: Use fixed-size byte array is better than that of byte, a the latter is a waste of space.
- Result: Not found
- Severity: Low

Analysis: Smart Contract Functions Scheme



Analysis: Smart Contract Vulnerability Check

Fallback function security	Passed
Design logic	Passed
Economy model	Passed
Oracle calls	Passed
Timestamp dependence	Passed
Compiler warnings.	Passed
Methods execution permissions	Passed
Arithmetic accuracy	Passed
Scoping and declarations	Passed
Economy model	Passed
Safe Zeppelin module	Passed
Cross-function race conditions	Passed
Race conditions and reentrancy.	Passed
Cross-function	Passed
Integer overflow and underflow	Passed
Malicious Event log	Passed
Uninitialized storage pointers	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Private user data leaks	Passed
Impact of the exchange rate on the logic	Passed
Front running	Passed
Possible delays in data delivery	Passed

Summary

In this audit, we thoroughly analyzed the Baby LTC design and implementation. The smart-contract presents a unique offering in current BSC ecosystem. We are truly impressed by the design and implementation, especially the dedication to maximized gas optimization. The current code base is well organized and those identified issues are promptly confirmed and fixed.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

Disclaimer

The following disclaimer outlines the terms and conditions that govern the use of our blockchain smart contract security audit services. By engaging our services, you accept and agree to be bound by these terms and conditions.

1. The purpose of this security audit is to assess the security measures implemented in the smart contract on the blockchain network. Our services are provided on an "as is" basis. We do not guarantee that our services will be error-free or uninterrupted, or that any defects will be corrected.

2. The security audit focuses solely on the smart contract code and its implementation within the blockchain network. It does not cover any external systems, applications, or third-party integrations that may interact with the contract. We do not accept any liability for any damages or losses arising from the use of our services, including but not limited to loss of profits, data, or business interruption.

3. Our services do not constitute legal, financial, or investment advice. We do not provide any warranties or representations as to the accuracy, completeness, or reliability of the information provided in our reports.

4. Orbital Station Excelsior strives to maintain independence and objectivity throughout the security audit process. However, it is important to acknowledge that no audit can be completely free from biases or conflicts of interest. Our services are limited to the scope of work agreed upon with the client. We are not responsible for any issues that fall outside of this scope, including but not limited to issues arising from third-party software or hardware.

5. Our services do not guarantee the security of the smart contract or the blockchain network. We provide recommendations based on our expertise, but it is ultimately the responsibility of the client to ensure the security of their smart contract and blockchain network. The security audit report may include recommendations and suggestions for improving the smart contract's security. It is essential to understand that implementing these recommendations does not guarantee absolute security, as new vulnerabilities may emerge over time.

6. Security threats and vulnerabilities evolve over time. Therefore, it is crucial to continuously monitor and update the smart contract's security measures even after the completion of the audit. Regular security assessments are recommended to ensure ongoing protection.

7. We reserve the right to refuse service to anyone at any time for any reason.

First of all, it is important to note that no security audit can guarantee complete protection against all potential vulnerabilities. While we aim to identify and report any security flaws and vulnerabilities in the smart contract as thoroughly as possible, it is inevitable that some risks may remain undisclosed or may develop over time. Therefore, any use or reliance on the security audit report for the smart contract is done at your own risk.

Furthermore, the security audit report is based on the information provided to us at the time of the audit. Any subsequent changes, modifications or updates made to the smart contract without our knowledge may affect the security of the contract and therefore invalidate our report.

In conclusion, our blockchain smart contract security audit services are provided with the understanding that we are not liable for any damages or losses arising from their use. We recommend that clients seek legal and financial advice before engaging in any blockchain-related activities.

If you have any questions or concerns regarding this disclaimer or the security audit process, please contact auditkyc@osexnetwork.com for further clarification.

Smart Contract Security Audit End Point

About Orbital Station Excelsior (OSEX)

The Orbital Station Excelsior (OSEX) is an international community aimed to bring real benefits to the real world with the help of blockchain technologies, providing all possible assistance and giving hope where it no longer exists.

Our slogan - make the world better together! We do not focus on one blockchain, we use them all, using bridges, Web 3 applications, decentralized exchanges, related technologies and a financial system. We are not an official accountable government financial organization; the main mechanism of our smart contracts is the accumulation of funds, with their subsequent use for charitable needs and purposes around the whole world, as well as expanding the influence of our community on social networks and other Internet platforms. The 21st century is not only about discoveries and achievements in the IT field, but it is also the century of widespread use of these innovations. Blockchain technologies allow us to redirect money flows from one place to another without any intermediaries in the form of banks. The future belongs to electronic money, and Blockchain is the future!

OSEX aims to integrate the advantages of multiple chains to create a high-performance compound ecology. The special redistribution to all holders mechanism of smart-contract from every transaction provides participants with maximum profit. The transaction fee for OSEX includes an active burn mechanism that ensures that the token deflates, which means the price of the token increases over time.

All used DEX strives to provide one-stop liquidity services for more high-quality assets and brings users a safe, reliable, diversified and cost-effective transaction experience.

OSEX-project will build a new business ecology with full application scenarios covered and various chains connected. It will continue to expand application scenarios and lower the usage threshold to provide global users with more convenient, high-performance, low-cost and non-differentiated crypto-asset financial services, thereby realizing fair pricing of assets, instant settlement of transactions and free flow of values.

This project allows all holders to obtain benefits, and is also set to interact with the community. The Orbital Station Excelsior will allow participants to choose the direction of the project. Our beginning includes many opportunities for a crypto project to meet the real world. For example, 1% of each transaction will go to a special charity wallet. Cash income received from this wallet will be sent to charitable projects around the world (rescue funds, volunteer organizations, international movements, environmental protection, etc.). We guarantee to provide reports and proofs of our activities.

The Smart contract security audit and KYC department is an independent department in the general structure of the Orbital Station Excelsior project. Department employees report directly to the general director for their work.

The project was founded on November 07, 2021.

Contacts and links

Orbital Station Excelsior (OSEX)

Website: https://osexnetwork.com/

E-mail:
osex.owner@gmail.com (for advertising and commercial questions)
auditkyc@osexnetwork.com (for smart contract security audit or team KYC requests)

Official Twitter profile: https://twitter.com/OSEXNetwork

Telegram chat group: https://t.me/osex_chat
Telegram announcements channel: https://t.me/osex_announcements
Telegram RU chat group: https://t.me/osex_chat_ru
Telegram NG chat group: https://t.me/osex_chat_ng

Telegram contact (for private dialog): https://t.me/osex_support

We are on blockchain:
https://blockscan.com/address/0x42614e5acf9c084a8afdff402ecd89d19f675c00

Smart contract address (BSC): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Avalanche): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Cronos): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Polygon): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Huobi ECO Chain): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Moonbeam): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00
Smart contract address (Fantom): 0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00

Contract code on block explorers:
https://bscscan.com/address/0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00#code
https://snowtrace.io/address/0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00#code
https://cronoscan.com/address/0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00#code
https://polygonscan.com/address/0x42614E5ACf9c084a8aFDfF402eCD89d19F675c00#code
https://moonbeam.moonscan.io/address/0x42614e5acf9c084a8afdff402ecd89d19f675c00#code
https://ftmscan.com/address/0x42614e5acf9c084a8afdff402ecd89d19f675c00#code
https://www.hecoinfo.com/en-us/token/0x42614e5acf9c084a8afdff402ecd89d19f675c00?tab=Transfers

Discord: https://discord.gg/4uPVcyen8Z
GitHub: https://github.com/OrbitalStationExcelsior
Instagram: https://www.instagram.com/osexnetwork/
Youtube: https://www.youtube.com/channel/UCm5QiJSu9rySS15arUXZbpw
GitBook: https://osex.gitbook.io/docs/
Medium: https://medium.com/@OrbitalStationExcelsior
Reddit: https://www.reddit.com/r/OSEX/
Techrate free audit for BSC:
https://drive.google.com/file/d/17lO8y0qELM8yhm7iXKXGPyc0NrjXeJOU
Dev Linkedin: https://www.linkedin.com/in/valdisveiss
Whitepaper: https://osexnetwork.com/OSEX_Whitepaper.pdf

Smart contract security audit

Report

I hereby confirm the full implementation of the order from representatives of the «Baby LTC» team regarding the security audit of this smart contract:
(BABYTOKEN 0x59FB9a377004CC8974009F72d94b4d801C58499B)

OSEX CEO, Valdis Veiss