Smart Contract
Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the BitTorrent on 2022.02.16. The following are the details and results of this smart contract security audit:

Token Name :

BitTorrent

The contract address :

https://etherscan.io/address/0xC669928185DbCE49d2230CC9B0979BE6DC797957

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed

NO.	Audit Items	Result
13	Safety Design Audit	Passed

Audit Result : Passed

Audit Number : 0X002202210002

Audit Date : 2022.02.16 - 2022.02.21

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed. SafeMath security module is used, which is a recommended approach. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found the following issue:

1. The admin role can add a new PREDICATE_ROLE and the PREDICATE_ROLE can mint tokens abitrarily through the mint function and there is no upper limit on the amount of tokens that can be minted. But the admin role is no longer exists in this contract and the PREDICATE_ROLE is owned by the MintableERC20PredicateProxy contract 0x9277a463A508F45115FdEaf22FfeDA1B16352433.
   After communication with the project team, the project team expresses that the PREDICATE ROLE is completely decentralized. The "PREDICATE ROLE" is held by the governance contract of the BTT-Chain (BTTC). BTTC itself is a polygon-like cross-chain protocol. The 7 validators generated based on pledge are jointly governed. The decentralized process is as follows: Ethereum BTT PREDICATE_ROLE permission by MintableERC20PredicateProxy contract. BTTC Validators periodically submit BTTC block headers through RootChainProxy's submitCheckpoint. After the BTTC block header has been committed to the contract, the user calls RootChainManagerProxy's exit method to submit the burn-proof on the BTTC. RootChainManagerProxy validates the burning proof submitted by the user based on the BTTC header submitted by the Validator in RootChainProxy. By the method of MintableERC20PredicateProxy calls the BTT contracts of mint, casting out the amount of BTT.

The RootChainManagerProxy contract address is 0xD06029b23e9d4CD24bAd01d436837Fa02B8f0dd9.

The RootChainProxy contract address is 0x98dfb360cbc65045a8415fa2514f549cd3000f02.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2021-12-10
*/
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
// File: @openzeppelin/contracts/utils/Context.sol

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <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 GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

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

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

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <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);
}

// File: @openzeppelin/contracts/math/SafeMath.sol

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
```

```
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
```

//SlowMist// OpenZeppelin's SafeMath security module is used, which is a recommended approach

```
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) {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }

    /**
     * @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) {
        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) {
        // 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);
    }

    /**
     * @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) {
        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) {
        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) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    /**
```

```
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        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) {
        if (a == 0) return 0;
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
        require(b > 0, "SafeMath: division by zero");
        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) {
        require(b > 0, "SafeMath: modulo by zero");
        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) {
        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.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for
the error
     * message unnecessarily. For custom revert reasons use {tryDiv}.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b > 0, errorMessage);
        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) {
        require(b > 0, errorMessage);
        return a % b;
    }
```

```
}

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

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;




/**
 * @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 guidelines: functions revert instead
 * of 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 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;
```

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

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

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

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public view virtual 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 {_setupDecimals} is
 * called.
 *
```

```
     * 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 returns (uint8) {
        return _decimals;
    }

    /**
     * @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);
        //SlowMist// The return value conforms to the EIP20 specification
        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);
        //SlowMist// The return value conforms to the EIP20 specification
        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);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount,
"ERC20: transfer amount exceeds allowance"));
        //SlowMist// The return value conforms to the EIP20 specification
        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].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {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) {
        _approve(_msgSender(), spender, _allowances[_msgSender()]
[spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
```

```
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal
virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount
exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

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

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

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
```

```
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

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

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount
exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This 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");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to approve errors
        require(spender != address(0), "ERC20: approve to the zero address");

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

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
```

```
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal virtual {
        _decimals = decimals_;
    }

    /**
     * @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 to 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 { }
}

// File: contracts/root/RootToken/IMintableERC20.sol

pragma solidity 0.6.6;

interface IMintableERC20 is IERC20 {
    /**
     * @notice called by predicate contract to mint tokens while withdrawing
     * @dev Should be callable only by MintableERC20Predicate
     * Make sure minting is done only by this function
     * @param user user address for whom token is being minted
     * @param amount amount of token being minted
     */
    function mint(address user, uint256 amount) external;
}

// File: contracts/common/Initializable.sol

pragma solidity 0.6.6;
```

```
contract Initializable {
    bool inited = false;

    modifier initializer() {
        require(!inited, "already inited");
        _;
        inited = true;
    }
}

// File: contracts/common/EIP712Base.sol

pragma solidity 0.6.6;


contract EIP712Base is Initializable {
    struct EIP712Domain {
        string name;
        string version;
        address verifyingContract;
        bytes32 salt;
    }

    string constant public ERC712_VERSION = "1";

    bytes32 internal constant EIP712_DOMAIN_TYPEHASH = keccak256(
        bytes(
            "EIP712Domain(string name,string version,address
verifyingContract,bytes32 salt)"
        )
    );
    bytes32 internal domainSeperator;

    // supposed to be called once while initializing.
    // one of the contractsa that inherits this contract follows proxy pattern
    // so it is not possible to do this in a constructor
    function _initializeEIP712(
        string memory name
    )
        internal
        initializer
    {
        _setDomainSeperator(name);
```

```
    }

    function _setDomainSeperator(string memory name) internal {
        domainSeperator = keccak256(
            abi.encode(
                EIP712_DOMAIN_TYPEHASH,
                keccak256(bytes(name)),
                keccak256(bytes(ERC712_VERSION)),
                address(this),
                bytes32(getChainId())
            )
        );
    }

    function getDomainSeperator() public view returns (bytes32) {
        return domainSeperator;
    }

    function getChainId() public pure returns (uint256) {
        uint256 id;
        assembly {
            id := chainid()
        }
        return id;
    }

    /**
     * Accept message hash and returns hash message in EIP712 compatible form
     * So that it can be used to recover signer from signature signed using EIP712
formatted data
     * https://eips.ethereum.org/EIPS/eip-712
     * "\\x19" makes the encoding deterministic
     * "\\x01" is the version byte to make it compatible to EIP-191
     */
    function toTypedMessageHash(bytes32 messageHash)
        internal
        view
        returns (bytes32)
    {
        return
            keccak256(
                abi.encodePacked("\x19\x01", getDomainSeperator(), messageHash)
            );
    }
```

```
}

// File: contracts/common/NativeMetaTransaction.sol

pragma solidity 0.6.6;



contract NativeMetaTransaction is EIP712Base {
    using SafeMath for uint256;
    bytes32 private constant META_TRANSACTION_TYPEHASH = keccak256(
        bytes(
            "MetaTransaction(uint256 nonce,address from,bytes functionSignature)"
        )
    );
    event MetaTransactionExecuted(
        address userAddress,
        address payable relayerAddress,
        bytes functionSignature
    );
    mapping(address => uint256) nonces;

    /*
     * Meta transaction structure.
     * No point of including value field here as if user is doing value transfer then
he has the funds to pay for gas
     * He should call the desired function directly in that case.
     */
    struct MetaTransaction {
        uint256 nonce;
        address from;
        bytes functionSignature;
    }

    function executeMetaTransaction(
        address userAddress,
        bytes memory functionSignature,
        bytes32 sigR,
        bytes32 sigS,
        uint8 sigV
    ) public payable returns (bytes memory) {
        MetaTransaction memory metaTx = MetaTransaction({
            nonce: nonces[userAddress],
            from: userAddress,
```

```
            functionSignature: functionSignature
        });

        require(
            verify(userAddress, metaTx, sigR, sigS, sigV),
            "Signer and signature do not match"
        );

        // increase nonce for user (to avoid re-use)
        nonces[userAddress] = nonces[userAddress].add(1);

        emit MetaTransactionExecuted(
            userAddress,
            msg.sender,
            functionSignature
        );

        // Append userAddress and relayer address at the end to extract it from
calling context
        (bool success, bytes memory returnData) = address(this).call(
            abi.encodePacked(functionSignature, userAddress)
        );
        require(success, "Function call not successful");

        return returnData;
    }

    function hashMetaTransaction(MetaTransaction memory metaTx)
        internal
        pure
        returns (bytes32)
    {
        return
            keccak256(
                abi.encode(
                    META_TRANSACTION_TYPEHASH,
                    metaTx.nonce,
                    metaTx.from,
                    keccak256(metaTx.functionSignature)
                )
            );
    }

    function getNonce(address user) public view returns (uint256 nonce) {
```

```
        nonce = nonces[user];
    }

    function verify(
        address signer,
        MetaTransaction memory metaTx,
        bytes32 sigR,
        bytes32 sigS,
        uint8 sigV
    ) internal view returns (bool) {
        require(signer != address(0), "NativeMetaTransaction: INVALID_SIGNER");
        return
            signer ==
            ecrecover(
                toTypedMessageHash(hashMetaTransaction(metaTx)),
                sigV,
                sigR,
                sigS
            );
    }
}

// File: contracts/common/ContextMixin.sol

pragma solidity 0.6.6;

abstract contract ContextMixin {
    function msgSender()
        internal
        view
        returns (address payable sender)
    {
        if (msg.sender == address(this)) {
            bytes memory array = msg.data;
            uint256 index = msg.data.length;
            assembly {
                // Load the 32 bytes word from memory with the address on the lower
20 bytes, and mask those.
                sender := and(
                    mload(add(array, index)),
                    0xffffffffffffffffffffffffffffffffffffffff
                )
            }
        } else {
```

```
            sender = msg.sender;
        }
        return sender;
    }
}

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

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 * ```
 * contract Example {
 *     // Add the library methods
 *     using EnumerableSet for EnumerableSet.AddressSet;
 *
 *     // Declare a set state variable
 *     EnumerableSet.AddressSet private mySet;
 * }
 * ```
 *
 * As of v3.3.0, sets of type `bytes32` (`Bytes32Set`), `address` (`AddressSet`)
 * and `uint256` (`UintSet`) are supported.
 */
library EnumerableSet {
    // To implement this library for multiple types with as little code
    // repetition as possible, we write it in terms of a generic Set type with
    // bytes32 values.
    // The Set implementation uses private functions, and user-facing
    // implementations (such as AddressSet) are just wrappers around the
    // underlying Set.
    // This means that we can only create new EnumerableSets for types that fit
```

```
    // in bytes32.

    struct Set {
        // Storage of set values
        bytes32[] _values;

        // Position of the value in the `values` array, plus 1 because index 0
        // means a value is not in the set.
        mapping (bytes32 => uint256) _indexes;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function _add(Set storage set, bytes32 value) private returns (bool) {
        if (!_contains(set, value)) {
            set._values.push(value);
            // The value is stored at length-1, but we add 1 to all indexes
            // and use 0 as a sentinel value
            set._indexes[value] = set._values.length;
            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from the
same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) { // Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the
element to delete with the last one in
            // the array, and then remove the last element (sometimes called as 'swap
```

```
and pop').
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            // When the value to delete is the last one, the swap operation is
unnecessary. However, since this occurs
            // so rarely, we still do the swap anyway to avoid the gas cost of adding
an 'if' statement.

            bytes32 lastvalue = set._values[lastIndex];

            // Move the last value to the index where the value to delete is
            set._values[toDeleteIndex] = lastvalue;
            // Update the index for the moved value
            set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

            // Delete the slot where the moved value was stored
            set._values.pop();

            // Delete the index for the deleted slot
            delete set._indexes[value];

            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function _contains(Set storage set, bytes32 value) private view returns (bool) {
        return set._indexes[value] != 0;
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function _length(Set storage set) private view returns (uint256) {
        return set._values.length;
    }
```

```
/**
 * @dev Returns the value stored at position `index` in the set. O(1).
 *
 * Note that there are no guarantees on the ordering of values inside the
 * array, and it may change when more values are added or removed.
 *
 * Requirements:
 *
 * - `index` must be strictly less than {length}.
 */
function _at(Set storage set, uint256 index) private view returns (bytes32) {
    require(set._values.length > index, "EnumerableSet: index out of bounds");
    return set._values[index];
}

// Bytes32Set

struct Bytes32Set {
    Set _inner;
}

/**
 * @dev Add a value to a set. O(1).
 *
 * Returns true if the value was added to the set, that is if it was not
 * already present.
 */
function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
    return _add(set._inner, value);
}

/**
 * @dev Removes a value from a set. O(1).
 *
 * Returns true if the value was removed from the set, that is if it was
 * present.
 */
function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
    return _remove(set._inner, value);
}

/**
 * @dev Returns true if the value is in the set. O(1).
 */
```

```
    function contains(Bytes32Set storage set, bytes32 value) internal view returns
(bool) {
        return _contains(set._inner, value);
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(Bytes32Set storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function at(Bytes32Set storage set, uint256 index) internal view returns
(bytes32) {
        return _at(set._inner, index);
    }

    // AddressSet

    struct AddressSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
```

```
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns
(bool) {
        return _contains(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function at(AddressSet storage set, uint256 index) internal view returns
(address) {
        return address(uint160(uint256(_at(set._inner, index))));
    }


    // UintSet

    struct UintSet {
        Set _inner;
```

```
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns
(bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
```

```
     *
     * - `index` must be strictly less than {length}.
     */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }
}

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

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.2 <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;
        // solhint-disable-next-line no-inline-assembly
        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");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?
highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
```

```
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes
memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

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

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

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}
[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
```

```
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value,
string memory errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for
call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (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");

        // solhint-disable-next-line avoid-low-level-calls
        (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");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(bool success, bytes memory returndata, string memory
errorMessage) private 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

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

```
}

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

// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;




/**
 * @dev Contract module that allows children to implement role-based access
 * control mechanisms.
 *
 * Roles are referred to by their `bytes32` identifier. These should be exposed
 * in the external API and be unique. The best way to achieve this is by
 * using `public constant` hash digests:
 *
 * ```
 * bytes32 public constant MY_ROLE = keccak256("MY_ROLE");
 * ```
 *
 * Roles can be used to represent a set of permissions. To restrict access to a
 * function call, use {hasRole}:
 *
 * ```
 * function foo() public {
 *     require(hasRole(MY_ROLE, msg.sender));
 *     ...
 * }
 * ```
 *
 * Roles can be granted and revoked dynamically via the {grantRole} and
 * {revokeRole} functions. Each role has an associated admin role, and only
 * accounts that have a role's admin role can call {grantRole} and {revokeRole}.
 *
 * By default, the admin role for all roles is `DEFAULT_ADMIN_ROLE`, which means
 * that only accounts with this role will be able to grant or revoke other
 * roles. More complex role relationships can be created by using
 * {_setRoleAdmin}.
 *
 * WARNING: The `DEFAULT_ADMIN_ROLE` is also its own admin: it has permission to
 * grant and revoke this role. Extra precautions should be taken to secure
```

```
 * accounts that have been granted it.
 */
abstract contract AccessControl is Context {
    using EnumerableSet for EnumerableSet.AddressSet;
    using Address for address;

    struct RoleData {
        EnumerableSet.AddressSet members;
        bytes32 adminRole;
    }

    mapping (bytes32 => RoleData) private _roles;

    bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

    /**
     * @dev Emitted when `newAdminRole` is set as ``role``'s admin role, replacing
`previousAdminRole`
     *
     * `DEFAULT_ADMIN_ROLE` is the starting admin for all roles, despite
     * {RoleAdminChanged} not being emitted signaling this.
     *
     * _Available since v3.1._
     */
    event RoleAdminChanged(bytes32 indexed role, bytes32 indexed previousAdminRole,
bytes32 indexed newAdminRole);

    /**
     * @dev Emitted when `account` is granted `role`.
     *
     * `sender` is the account that originated the contract call, an admin role
     * bearer except when using {_setupRole}.
     */
    event RoleGranted(bytes32 indexed role, address indexed account, address indexed
sender);

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

```
sender);

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

    /**
     * @dev Returns the number of accounts that have `role`. Can be used
     * together with {getRoleMember} to enumerate all bearers of a role.
     */
    function getRoleMemberCount(bytes32 role) public view returns (uint256) {
        return _roles[role].members.length();
    }

    /**
     * @dev Returns one of the accounts that have `role`. `index` must be a
     * value between 0 and {getRoleMemberCount}, non-inclusive.
     *
     * Role bearers are not sorted in any particular way, and their ordering may
     * change at any point.
     *
     * WARNING: When using {getRoleMember} and {getRoleMemberCount}, make sure
     * you perform all queries on the same block. See the following
     * https://forum.openzeppelin.com/t/iterating-over-elements-on-enumerableset-in-
openzeppelin-contracts/2296[forum post]
     * for more information.
     */
    function getRoleMember(bytes32 role, uint256 index) public view returns (address)
{
        return _roles[role].members.at(index);
    }

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

```
    /**
     * @dev Grants `role` to `account`.
     *
     * If `account` had not been already granted `role`, emits a {RoleGranted}
     * event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function grantRole(bytes32 role, address account) public virtual {
        require(hasRole(_roles[role].adminRole, _msgSender()), "AccessControl: sender
must be an admin to grant");

        _grantRole(role, account);
    }

    /**
     * @dev Revokes `role` from `account`.
     *
     * If `account` had been granted `role`, emits a {RoleRevoked} event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function revokeRole(bytes32 role, address account) public virtual {
        require(hasRole(_roles[role].adminRole, _msgSender()), "AccessControl: sender
must be an admin to revoke");

        _revokeRole(role, account);
    }

    /**
     * @dev Revokes `role` from the calling account.
     *
     * Roles are often managed via {grantRole} and {revokeRole}: this function's
     * purpose is to provide a mechanism for accounts to lose their privileges
     * if they are compromised (such as when a trusted device is misplaced).
     *
     * If the calling account had been granted `role`, emits a {RoleRevoked}
     * event.
     *
```

```
     * Requirements:
     *
     * - the caller must be `account`.
     */
    function renounceRole(bytes32 role, address account) public virtual {
        require(account == _msgSender(), "AccessControl: can only renounce roles for
self");

        _revokeRole(role, account);
    }

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

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

    function _grantRole(bytes32 role, address account) private {
        if (_roles[role].members.add(account)) {
            emit RoleGranted(role, account, _msgSender());
```

```
        }
    }

    function _revokeRole(bytes32 role, address account) private {
        if (_roles[role].members.remove(account)) {
            emit RoleRevoked(role, account, _msgSender());
        }
    }
}

// File: contracts/common/AccessControlMixin.sol

pragma solidity 0.6.6;


contract AccessControlMixin is AccessControl {
    string private _revertMsg;
    function _setupContractId(string memory contractId) internal {
        _revertMsg = string(abi.encodePacked(contractId, ":
INSUFFICIENT_PERMISSIONS"));
    }

    modifier only(bytes32 role) {
        require(
            hasRole(role, _msgSender()),
            _revertMsg
        );
        _;
    }
}

// File: contracts/root/RootToken/DummyMintableERC20.sol

// This contract is not supposed to be used in production
// It's strictly for testing purpose

pragma solidity 0.6.6;




contract BTT is
```

```
    ERC20,
    AccessControlMixin,
    NativeMetaTransaction,
    ContextMixin,
    IMintableERC20
{
    bytes32 public constant PREDICATE_ROLE = keccak256("PREDICATE_ROLE");

    constructor(string memory name_, string memory symbol_)
        public
        ERC20(name_, symbol_)
    {
        _setupContractId("BTT");
        _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
        _setupRole(PREDICATE_ROLE, _msgSender());
        _initializeEIP712(name_);
    }

    /**
     * @dev See {IMintableERC20-mint}.
     */
    //SlowMist// The PREDICATE_ROLE can mint tokens abitrarily through the mint
function and there is no upper limit on the amount of tokens that can be minted
    function mint(address user, uint256 amount) external override
only(PREDICATE_ROLE) {
        _mint(user, amount);
    }

    function _msgSender()
        internal
        override
        view
        returns (address payable sender)
    {
        return ContextMixin.msgSender();
    }
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist