Smart Contract Security Audit Report

The SlowMist Security Team received the TOROCUS team's application for smart contract security audit of the TOROCUS on Jan. 14, 2021. The following are the details and results of this smart contract security audit:

Token name :

TOROCUS

The Contract address :

0x406ae253Fb0aa898F9912fB192c1e6dEb9623A07

Link address :

https://etherscan.io/address/0x406ae253Fb0aa898F9912fB192c1e6dEb9623A07

The audit items and results :

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

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive authority audit	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False top-up” vulnerability Audit	-	Passed

9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : Passed

Audit Number : 0X002101140001

Audit Date : Jan. 14, 2021

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed, users can burn their tokens through the burn function. OpenZeppelin's SafeMath security module is used, which is a recommend approach. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found the following issues:

1. The Signer role can transfer any user's tokens through the transferDelegatedWithSign function, which will lead to the risk of excessive Signer authority. After communicating with the project party, the project party decided to set the Signer role to 0 address to avoid the risk of excessive signer role authority.

Reference transaction hash for removal operation:

0x471f46f1cf050e8cb1fbf41fa360e78c12b1d937b747645fb09982db1956fa5b

0x1974633b68c1d084f07370b4bb439b020462a13853ac76629fc9b602d78f7384

2. The Minter role can mint unlimited tokens via the mint function, which will also lead to the risk of excessive minter authority. After communicating with the project party, the project party decided to set the Minter role to 0 address to avoid the risk of excessive minter role authority.

Reference transaction hash for removal operation：

0x8d4f809a5ca7c4b32b7bb4622906c68643924687af55fcae7832d226e871d1cd

0xb60a4e0ee171a2f4d7c0dc5d76ebec386f3df2cad6fa7718db7348276c9539ac

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2019-07-08
*/

// File: openzeppelin-solidity/contracts/token/ERC20/IERC20.sol

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.5.2;

/**
 * @title ERC20 interface
 * @dev see https://eips.ethereum.org/EIPS/eip-20
 */
interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns (uint256);
```

```
    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}

// File: openzeppelin-solidity/contracts/math/SafeMath.sol

pragma solidity ^0.5.2;

/**
 * @title SafeMath
 * @dev Unsigned math operations with safety checks that revert on error
 */
```

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

```
library SafeMath {
    /**
     * @dev Multiplies two unsigned integers, reverts on overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
     * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
```

```
        return c;
    }

    /**
     * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than minuend).
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Adds two unsigned integers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two unsigned integers and returns the remainder (unsigned integer modulo),
     * reverts when dividing by zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20.sol

pragma solidity ^0.5.2;



/**
 * @title Standard ERC20 token
```

```
 *
 * @dev Implementation of the basic standard token.
 * https://eips.ethereum.org/EIPS/eip-20
 * Originally based on code by FirstBlood:
 * https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 *
 * This implementation emits additional Approval events, allowing applications to reconstruct the allowance status for
 * all accounts just by listening to said events. Note that this isn't required by the specification, and other
 * compliant implementations may not do it.
 */
contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    /**
     * @dev Total number of tokens in existence
     */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev Gets the balance of the specified address.
     * @param owner The address to query the balance of.
     * @return A uint256 representing the amount owned by the passed address.
     */
    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }

    /**
     * @dev Function to check the amount of tokens that an owner allowed to a spender.
     * @param owner address The address which owns the funds.
     * @param spender address The address which will spend the funds.
     * @return A uint256 specifying the amount of tokens still available for the spender.
     */
    function allowance(address owner, address spender) public view returns (uint256) {
```

```
        return _allowed[owner][spender];
    }

    /**
     * @dev Transfer token to a specified address
     * @param to The address to transfer to.
     * @param value The amount to be transferred.
     */
    function transfer(address to, uint256 value) public returns (bool) {
        _transfer(msg.sender, to, value);
        return true; //SlowMist// The return value conforms to the EIP20 specification
    }

    /**
     * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
     * 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
     * @param spender The address which will spend the funds.
     * @param value The amount of tokens to be spent.
     */
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value);
        return true; //SlowMist// The return value conforms to the EIP20 specification
    }

    /**
     * @dev Transfer tokens from one address to another.
     * Note that while this function emits an Approval event, this is not required as per the specification,
     * and other compliant implementations may not emit the event.
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
     * @param value uint256 the amount of tokens to be transferred
     */
    function transferFrom(address from, address to, uint256 value) public returns (bool) {
        _transfer(from, to, value);
        _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
        return true; //SlowMist// The return value conforms to the EIP20 specification
```

```
}

/**
 * @dev Increase the amount of tokens that an owner allowed to a spender.
 * approve should be called when _allowed[msg.sender][spender] == 0. To increment
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * Emits an Approval event.
 * @param spender The address which will spend the funds.
 * @param addedValue The amount of tokens to increase the allowance by.
 */
function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
    _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
    return true;
}

/**
 * @dev Decrease the amount of tokens that an owner allowed to a spender.
 * approve should be called when _allowed[msg.sender][spender] == 0. To decrement
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * Emits an Approval event.
 * @param spender The address which will spend the funds.
 * @param subtractedValue The amount of tokens to decrease the allowance by.
 */
function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
    _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue));
    return true;
}

/**
 * @dev Transfer token for a specified addresses
 * @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 {
```

```
        require(to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer

        _balances[from] = _balances[from].sub(value);
        _balances[to] = _balances[to].add(value);
        emit Transfer(from, to, value);
    }

    /**
     * @dev Internal function that mints an amount of the token and assigns it to
     * an account. This encapsulates the modification of balances such that the
     * proper events are emitted.
     * @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 {
        require(account != address(0));

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

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account.
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    function _burn(address account, uint256 value) internal {
        require(account != address(0));

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

    /**
     * @dev Approve an address to spend another addresses' tokens.
     * @param owner The address that owns the tokens.
```

```
     * @param spender The address that will spend the tokens.
     * @param value The number of tokens that can be spent.
     */
    function _approve(address owner, address spender, uint256 value) internal {
        require(spender != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to approve errors
        require(owner != address(0));

        _allowed[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account, deducting from the sender's allowance for said account. Uses the
     * internal burn function.
     * Emits an Approval event (reflecting the reduced allowance).
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    //SlowMist// Because burnFrom() and transferFrom() share the _allowed amount of _approve(), if the agent be evil, there is the possibility of malicious burn
    function _burnFrom(address account, uint256 value) internal {
        _burn(account, value);
        _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
    }
}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20Detailed.sol

pragma solidity ^0.5.2;


/**
 * @title ERC20Detailed token
 * @dev The decimals are only for visualization purposes.
 * All the operations are done using the smallest and indivisible token unit,
 * just as on Ethereum all the operations are done in wei.
```

```
 */
contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }

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

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

    /**
     * @return the number of decimals of the token.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

// File: openzeppelin-solidity/contracts/access/Roles.sol

pragma solidity ^0.5.2;

/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
```

```
library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    /**
     * @dev give an account access to this role
     */
    function add(Role storage role, address account) internal {
        require(account != address(0));
        require(!has(role, account));

        role.bearer[account] = true;
    }

    /**
     * @dev remove an account's access to this role
     */
    function remove(Role storage role, address account) internal {
        require(account != address(0));
        require(has(role, account));

        role.bearer[account] = false;
    }

    /**
     * @dev check if an account has this role
     * @return bool
     */
    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0));
        return role.bearer[account];
    }
}

// File: openzeppelin-solidity/contracts/access/roles/PauserRole.sol

pragma solidity ^0.5.2;


contract PauserRole {
    using Roles for Roles.Role;
```

```
    event PauserAdded(address indexed account);
    event PauserRemoved(address indexed account);

    Roles.Role private _pausers;

    constructor () internal {
        _addPauser(msg.sender);
    }

    modifier onlyPauser() {
        require(isPauser(msg.sender));
        _;
    }

    function isPauser(address account) public view returns (bool) {
        return _pausers.has(account);
    }

    function addPauser(address account) public onlyPauser {
        _addPauser(account);
    }

    function renouncePauser() public {
        _removePauser(msg.sender);
    }

    function _addPauser(address account) internal {
        _pausers.add(account);
        emit PauserAdded(account);
    }

    function _removePauser(address account) internal {
        _pausers.remove(account);
        emit PauserRemoved(account);
    }
}

// File: openzeppelin-solidity/contracts/lifecycle/Pausable.sol

pragma solidity ^0.5.2;
```

```
/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
contract Pausable is PauserRole {
    event Paused(address account);
    event Unpaused(address account);

    bool private _paused;

    constructor () internal {
        _paused = false;
    }

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

    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     */
    modifier whenNotPaused() {
        require(!_paused);
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     */
    modifier whenPaused() {
        require(_paused);
        _;
    }

    /**
     * @dev called by the owner to pause, triggers stopped state
     */
```

```
    //SlowMist// Suspending all transactions upon major abnormalities is a recommended approach
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }

    /**
     * @dev called by the owner to unpause, returns to normal state
     */
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20Pausable.sol

pragma solidity ^0.5.2;



/**
 * @title Pausable token
 * @dev ERC20 modified with pausable transfers.
 */
contract ERC20Pausable is ERC20, Pausable {
    function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transfer(to, value);
    }

    function transferFrom(address from, address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }

    function approve(address spender, uint256 value) public whenNotPaused returns (bool) {
        return super.approve(spender, value);
    }

    function increaseAllowance(address spender, uint addedValue) public whenNotPaused returns (bool success) {
        return super.increaseAllowance(spender, addedValue);
    }
```

```
    function decreaseAllowance(address spender, uint subtractedValue) public whenNotPaused returns (bool success) {
        return super.decreaseAllowance(spender, subtractedValue);
    }
}

// File: openzeppelin-solidity/contracts/access/roles/MinterRole.sol

pragma solidity ^0.5.2;


contract MinterRole {
    using Roles for Roles.Role;

    event MinterAdded(address indexed account);
    event MinterRemoved(address indexed account);

    Roles.Role private _minters;

    constructor () internal {
        _addMinter(msg.sender);
    }

    modifier onlyMinter() {
        require(isMinter(msg.sender));
        _;
    }

    function isMinter(address account) public view returns (bool) {
        return _minters.has(account);
    }

    function addMinter(address account) public onlyMinter {
        _addMinter(account);
    }

    function renounceMinter() public {
        _removeMinter(msg.sender);
    }

    function _addMinter(address account) internal {
        _minters.add(account);
```

```
        emit MinterAdded(account);
    }

    function _removeMinter(address account) internal {
        _minters.remove(account);
        emit MinterRemoved(account);
    }
}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20Mintable.sol

pragma solidity ^0.5.2;



/**
 * @title ERC20Mintable
 * @dev ERC20 minting logic
 */
contract ERC20Mintable is ERC20, MinterRole {
    /**
     * @dev Function to mint tokens
     * @param to The address that will receive the minted tokens.
     * @param value The amount of tokens to mint.
     * @return A boolean that indicates if the operation was successful.
     */
    //SlowMist// The Minter role can mint unlimited tokens via the mint function
    function mint(address to, uint256 value) public onlyMinter returns (bool) {
        _mint(to, value);
        return true;
    }
}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20Burnable.sol

pragma solidity ^0.5.2;



/**
 * @title Burnable Token
 * @dev Token that can be irreversibly burned (destroyed).
```

```
 */
contract ERC20Burnable is ERC20 {
    /**
     * @dev Burns a specific amount of tokens.
     * @param value The amount of token to be burned.
     */
    function burn(uint256 value) public {
        _burn(msg.sender, value);
    }

    /**
     * @dev Burns a specific amount of tokens from the target address and decrements allowance
     * @param from address The account whose tokens will be burned.
     * @param value uint256 The amount of token to be burned.
     */
    function burnFrom(address from, uint256 value) public {
        _burnFrom(from, value);
    }
}

// File: openzeppelin-solidity/contracts/ownership/Ownable.sol

pragma solidity ^0.5.2;

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }
```

```
/**
 * @return the address of the owner.
 */
function owner() public view returns (address) {
    return _owner;
}

/**
 * @dev Throws if called by any account other than the owner.
 */
modifier onlyOwner() {
    require(isOwner());
    _;
}

/**
 * @return true if `msg.sender` is the owner of the contract.
 */
function isOwner() public view returns (bool) {
    return msg.sender == _owner;
}

/**
 * @dev Allows the current owner to relinquish control of the contract.
 * It will not be possible to call the functions with the `onlyOwner`
 * modifier anymore.
 * @notice Renouncing ownership will leave the contract without an owner,
 * thereby removing any functionality that is only available to the owner.
 */
function renounceOwnership() public onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
}

/**
 * @dev Allows the current owner to transfer control of the contract to a newOwner.
 * @param newOwner The address to transfer ownership to.
 */
function transferOwnership(address newOwner) public onlyOwner {
    _transferOwnership(newOwner);
}
```

```
    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0)); //SlowMist// This check is quite good in avoiding losing control of the contract caused by user mistakes
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: openzeppelin-solidity/contracts/access/roles/SignerRole.sol

pragma solidity ^0.5.2;


contract SignerRole {
    using Roles for Roles.Role;

    event SignerAdded(address indexed account);
    event SignerRemoved(address indexed account);

    Roles.Role private _signers;

    constructor () internal {
        _addSigner(msg.sender);
    }

    modifier onlySigner() {
        require(isSigner(msg.sender));
        _;
    }

    function isSigner(address account) public view returns (bool) {
        return _signers.has(account);
    }

    function addSigner(address account) public onlySigner {
```

```
        _addSigner(account);
    }

    function renounceSigner() public {
        _removeSigner(msg.sender);
    }

    function _addSigner(address account) internal {
        _signers.add(account);
        emit SignerAdded(account);
    }

    function _removeSigner(address account) internal {
        _signers.remove(account);
        emit SignerRemoved(account);
    }
}

// File: openzeppelin-solidity/contracts/cryptography/ECDSA.sol

pragma solidity ^0.5.2;

/**
 * @title Elliptic curve signature operations
 * @dev Based on https://gist.github.com/axic/5b33912c6f61ae6fd96d6c4a47afde6d
 * TODO Remove this library once solidity supports passing a signature to ecrecover.
 * See https://github.com/ethereum/solidity/issues/864
 */

library ECDSA {
    /**
     * @dev Recover signer address from a message by using their signature
     * @param hash bytes32 message, the hash is the signed message. What is recovered is the signer address.
     * @param signature bytes signature, the signature is generated using web3.eth.sign()
     */
    function recover(bytes32 hash, bytes memory signature) internal pure returns (address) {
        // Check the signature length
        if (signature.length != 65) {
            return (address(0));
        }

        // Divide the signature in r, s and v variables
```

```
        bytes32 r;
        bytes32 s;
        uint8 v;

        // ecrecover takes the signature parameters, and the only way to get them
        // currently is to use assembly.
        // solhint-disable-next-line no-inline-assembly
        assembly {
            r := mload(add(signature, 0x20))
            s := mload(add(signature, 0x40))
            v := byte(0, mload(add(signature, 0x60)))
        }

        // EIP-2 still allows signature malleability for ecrecover(). Remove this possibility and make the signature
        // unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.io/yellowpaper/paper.pdf), defines
        // the valid range for s in (281): 0 < s < secp256k1n ÷ 2 + 1, and for v in (282): v ∈ {27, 28}. Most
        // signatures from current libraries generate a unique signature with an s-value in the lower half order.
        //
        // If your library generates malleable signatures, such as s-values in the upper range, calculate a new s-value
        // with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 - s1 and flip v
from 27 to 28 or
        // vice versa. If your library also generates signatures with 0/1 for v instead 27/28, add 27 to v to accept
        // these malleable signatures as well.
        if (uint256(s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
            return address(0);
        }

        if (v != 27 && v != 28) {
            return address(0);
        }

        // If the signature is valid (and not malleable), return the signer address
        return ecrecover(hash, v, r, s);
    }

    /**
     * toEthSignedMessageHash
     * @dev prefix a bytes32 value with "\x19Ethereum Signed Message:"
     * and hash the result
     */
    function toEthSignedMessageHash(bytes32 hash) internal pure returns (bytes32) {
        // 32 is the length in bytes of hash,
```

```
        // enforced by the type signature above
        return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", hash));
    }
}

// File: openzeppelin-solidity/contracts/drafts/SignatureBouncer.sol

pragma solidity ^0.5.2;



/**
 * @title SignatureBouncer
 * @author PhABC, Shrugs and aflesher
 * @dev SignatureBouncer allows users to submit a signature as a permission to
 * do an action.
 * If the signature is from one of the authorized signer addresses, the
 * signature is valid.
 * Note that SignatureBouncer offers no protection against replay attacks, users
 * must add this themselves!
 *
 * Signer addresses can be individual servers signing grants or different
 * users within a decentralized club that have permission to invite other
 * members. This technique is useful for whitelists and airdrops; instead of
 * putting all valid addresses on-chain, simply sign a grant of the form
 * keccak256(abi.encodePacked(`:contractAddress` + `:granteeAddress`)) using a
 * valid signer address.
 * Then restrict access to your crowdsale/whitelist/airdrop using the
 * `onlyValidSignature` modifier (or implement your own using _isValidSignature).
 * In addition to `onlyValidSignature`, `onlyValidSignatureAndMethod` and
 * `onlyValidSignatureAndData` can be used to restrict access to only a given
 * method or a given method with given parameters respectively.
 * See the tests in SignatureBouncer.test.js for specific usage examples.
 *
 * @notice A method that uses the `onlyValidSignatureAndData` modifier must make
 * the _signature parameter the "last" parameter. You cannot sign a message that
 * has its own signature in it so the last 128 bytes of msg.data (which
 * represents the length of the _signature data and the _signature data itself)
 * is ignored when validating. Also non fixed sized parameters make constructing
 * the data in the signature much more complex.
 * See https://ethereum.stackexchange.com/a/50616 for more details.
 */
```

```
contract SignatureBouncer is SignerRole {
    using ECDSA for bytes32;

    // Function selectors are 4 bytes long, as documented in
    // https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector
    uint256 private constant _METHOD_ID_SIZE = 4;
    // Signature size is 65 bytes (tightly packed v + r + s), but gets padded to 96 bytes
    uint256 private constant _SIGNATURE_SIZE = 96;

    constructor () internal {
        // solhint-disable-previous-line no-empty-blocks
    }

    /**
     * @dev requires that a valid signature of a signer was provided
     */
    modifier onlyValidSignature(bytes memory signature) {
        require(_isValidSignature(msg.sender, signature));
        _;
    }

    /**
     * @dev requires that a valid signature with a specified method of a signer was provided
     */
    modifier onlyValidSignatureAndMethod(bytes memory signature) {
        require(_isValidSignatureAndMethod(msg.sender, signature));
        _;
    }

    /**
     * @dev requires that a valid signature with a specified method and params of a signer was provided
     */
    modifier onlyValidSignatureAndData(bytes memory signature) {
        require(_isValidSignatureAndData(msg.sender, signature));
        _;
    }

    /**
     * @dev is the signature of `this + account` from a signer?
     * @return bool
     */
    function _isValidSignature(address account, bytes memory signature) internal view returns (bool) {
```

```
        return _isValidDataHash(keccak256(abi.encodePacked(address(this), account)), signature);
    }

    /**
     * @dev is the signature of `this + account + methodId` from a signer?
     * @return bool
     */
    function _isValidSignatureAndMethod(address account, bytes memory signature) internal view returns (bool) {
        bytes memory data = new bytes(_METHOD_ID_SIZE);
        for (uint i = 0; i < data.length; i++) {
            data[i] = msg.data[i];
        }
        return _isValidDataHash(keccak256(abi.encodePacked(address(this), account, data)), signature);
    }

    /**
     * @dev is the signature of `this + account + methodId + params(s)` from a signer?
     * @notice the signature parameter of the method being validated must be the "last" parameter
     * @return bool
     */
    function _isValidSignatureAndData(address account, bytes memory signature) internal view returns (bool) {
        require(msg.data.length > _SIGNATURE_SIZE);

        bytes memory data = new bytes(msg.data.length - _SIGNATURE_SIZE);
        for (uint i = 0; i < data.length; i++) {
            data[i] = msg.data[i];
        }

        return _isValidDataHash(keccak256(abi.encodePacked(address(this), account, data)), signature);
    }

    /**
     * @dev internal function to convert a hash to an eth signed message
     * and then recover the signature and check it against the signer role
     * @return bool
     */
    function _isValidDataHash(bytes32 hash, bytes memory signature) internal view returns (bool) {
        address signer = hash.toEthSignedMessageHash().recover(signature);

        return signer != address(0) && isSigner(signer);
    }
}
```

```
// File: openzeppelin-solidity/contracts/introspection/ERC165Checker.sol

pragma solidity ^0.5.2;

/**
 * @title ERC165Checker
 * @dev Use `using ERC165Checker for address`; to include this library
 * https://eips.ethereum.org/EIPS/eip-165
 */
```

//SlowMist// This library is not used in the contract, it is suggested to remove

```
library ERC165Checker {
    // As per the EIP-165 spec, no interface should ever match 0xffffffff
    bytes4 private constant _INTERFACE_ID_INVALID = 0xffffffff;

    bytes4 private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7;
    /*
     * 0x01ffc9a7 ===
     *     bytes4(keccak256('supportsInterface(bytes4)'))
     */

    /**
     * @notice Query if a contract supports ERC165
     * @param account The address of the contract to query for support of ERC165
     * @return true if the contract at account implements ERC165
     */
    function _supportsERC165(address account) internal view returns (bool) {
        // Any contract that implements ERC165 must explicitly indicate support of
        // InterfaceId_ERC165 and explicitly indicate non-support of InterfaceId_Invalid
        return _supportsERC165Interface(account, _INTERFACE_ID_ERC165) &&
            !_supportsERC165Interface(account, _INTERFACE_ID_INVALID);
    }

    /**
     * @notice Query if a contract implements an interface, also checks support of ERC165
     * @param account The address of the contract to query for support of an interface
     * @param interfaceId The interface identifier, as specified in ERC-165
     * @return true if the contract at account indicates support of the interface with
     * identifier interfaceId, false otherwise
     * @dev Interface identification is specified in ERC-165.
     */
```

```
function _supportsInterface(address account, bytes4 interfaceId) internal view returns (bool) {
    // query support of both ERC165 as per the spec and support of _interfaceId
    return _supportsERC165(account) &&
        _supportsERC165Interface(account, interfaceId);
}

/**
 * @notice Query if a contract implements interfaces, also checks support of ERC165
 * @param account The address of the contract to query for support of an interface
 * @param interfaceIds A list of interface identifiers, as specified in ERC-165
 * @return true if the contract at account indicates support all interfaces in the
 * interfaceIds list, false otherwise
 * @dev Interface identification is specified in ERC-165.
 */
function _supportsAllInterfaces(address account, bytes4[] memory interfaceIds) internal view returns (bool) {
    // query support of ERC165 itself
    if (!_supportsERC165(account)) {
        return false;
    }

    // query support of each interface in _interfaceIds
    for (uint256 i = 0; i < interfaceIds.length; i++) {
        if (!_supportsERC165Interface(account, interfaceIds[i])) {
            return false;
        }
    }

    // all interfaces supported
    return true;
}

/**
 * @notice Query if a contract implements an interface, does not check ERC165 support
 * @param account The address of the contract to query for support of an interface
 * @param interfaceId The interface identifier, as specified in ERC-165
 * @return true if the contract at account indicates support of the interface with
 * identifier interfaceId, false otherwise
 * @dev Assumes that account contains a contract that supports ERC165, otherwise
 * the behavior of this method is undefined. This precondition can be checked
 * with the `supportsERC165` method in this library.
 * Interface identification is specified in ERC-165.
 */
```

```
function _supportsERC165Interface(address account, bytes4 interfaceId) private view returns (bool) {
    // success determines whether the staticcall succeeded and result determines
    // whether the contract at account indicates support of _interfaceId
    (bool success, bool result) = _callERC165SupportsInterface(account, interfaceId);

    return (success && result);
}

/**
 * @notice Calls the function with selector 0x01ffc9a7 (ERC165) and suppresses throw
 * @param account The address of the contract to query for support of an interface
 * @param interfaceId The interface identifier, as specified in ERC-165
 * @return success true if the STATICCALL succeeded, false otherwise
 * @return result true if the STATICCALL succeeded and the contract at account
 * indicates support of the interface with identifier interfaceId, false otherwise
 */
function _callERC165SupportsInterface(address account, bytes4 interfaceId)
    private
    view
    returns (bool success, bool result)
{
    bytes memory encodedParams = abi.encodeWithSelector(_INTERFACE_ID_ERC165, interfaceId);

    // solhint-disable-next-line no-inline-assembly
    assembly {
        let encodedParams_data := add(0x20, encodedParams)
        let encodedParams_size := mload(encodedParams)

        let output := mload(0x40)      // Find empty storage location using "free memory pointer"
        mstore(output, 0x0)

        success := staticcall(
            30000,                     // 30k gas
            account,                   // To addr
            encodedParams_data,
            encodedParams_size,
            output,
            0x20                       // Outputs are 32 bytes long
        )

        result := mload(output)        // Load the result
    }
```

```
    }
}

// File: contracts/TorocusToken.sol

pragma solidity ^0.5.2;









contract TorocusToken is   ERC20Detailed, ERC20Mintable, ERC20Burnable, ERC20Pausable, SignatureBouncer {
    using SafeMath for uint256;
    mapping (address => mapping (uint256 => bool)) public _usedNonce;

    constructor(
        string memory name,
        string memory symbol,
        uint8 decimals,
        uint256 initialSupply,
        address initialHolder,
        address minter,
        address signer,
        address pauser
    )
        ERC20Detailed(name, symbol, decimals)
        SignatureBouncer()
        ERC20Mintable()
        ERC20Pausable()
        public
    {
        _mint(initialHolder, initialSupply);
        _addMinter(minter);
        _addPauser(pauser);
        _addSigner(signer);
    }
```

```
    modifier isNotUsedNonce(address from, uint256 nonce) {
        require(!_usedNonce[from][nonce]);
        _;
    }

    //SlowMist// The Signer role can transfer any user's tokens through the
transferDelegatedWithSign function
    function transferDelegatedWithSign(
        address from,
        address to,
        uint256 amount,
        uint256 fee,
        uint256 nonce,
        string memory message,
        bytes memory signature
    ) public
        whenNotPaused
        isNotUsedNonce(msg.sender, nonce)
        onlyValidSignatureAndData(signature)
        returns (bool success)
    {
        require(from != address(0));
        require(to != address(0));
        require(from != to);
        require(msg.sender != to);
        require(msg.sender != from);
        require(balanceOf(from) >= amount.add(fee), "not enough balance");

        if(fee > 0) {
            _transfer(from, msg.sender, fee);
        }
        _transfer(from, to, amount);

        _usedNonce[msg.sender][nonce] = true;
        return true;
    }
}
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist