# Smart Contract Security Audit Report

The SlowMist Security Team received the OpenOcean team's application for smart contract security audit of the OpenOceanExchange on Feb. 18, 2021. The following are the details and results of this smart contract security audit:

**Project name :**

OpenOceanExchange

**The Contract address :**

0x26d26b1a0243566d1cd38ff9afd5fd3f0fb6cbb4

**Link address :**

https://etherscan.io/address/0x26d26b1a0243566d1cd38ff9afd5fd3f0fb6cbb4

**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 Deposit” 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 : 0X002102240003

Audit Date : Feb. 24, 2021

Audit Team : SlowMist Security Team



**Summary: This is an exchange smart contract, The contract does not have the Overflow and the Race Conditions issue. OpenZeppelin's SafeMath security module is used, which is a recommend approach. The comprehensive evaluation contract is no risk.**

The source code:

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

```
/**
  *Submitted for verification at Etherscan.io on 2021-02-26
*/

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

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

/**
 * @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
```

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

```
pragma solidity ^0.6.0;

/**
 * @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.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
```

```
        return c;
    }

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

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }
```

```
    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

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



pragma solidity ^0.6.2;
```

```
/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
```

```
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     *
https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use
https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
```

```
    }

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

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

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal
returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }
```

```
    function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private
returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

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

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol



pragma solidity ^0.6.0;




/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
```

```
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
```

```
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased allowance below zero");
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which verifies that
        // the target address contains contract code and also asserts for success in the low-level call.

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed");
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

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



pragma solidity ^0.6.0;

/*
 * @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/access/Ownable.sol



pragma solidity ^0.6.0;

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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
```

```
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
```

```
}
```

**//SlowMist// Suspending all transactions upon major abnormalities is a recommended approach.**

```
// File: contracts/lib/Shutdownable.sol


pragma solidity ^0.6.0;


contract Shutdownable is Ownable {
    bool public isShutdown;

    event Shutdown();

    modifier notShutdown {
        require(!isShutdown, "Smart contract is shut down.");
        _;
    }

    function shutdown() public onlyOwner {
        isShutdown = true;
        emit Shutdown();
    }
}

// File: contracts/lib/UniversalERC20.sol


pragma solidity ^0.6.0;




/**
 * @dev See https://github.com/CryptoManiacsZone/1inchProtocol/blob/master/contracts/UniversalERC20.sol
 */
library UniversalERC20 {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
```

```
IERC20 private constant ZERO_ADDRESS = IERC20(0x0000000000000000000000000000000000000000);
IERC20 private constant ETH_ADDRESS = IERC20(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);

function universalTransfer(
    IERC20 token,
    address to,
    uint256 amount
) internal returns (bool) {
    if (amount == 0) {
        return true;
    }

    if (isETH(token)) {
        payable(to).transfer(amount);
        return true;
    } else {
        token.safeTransfer(to, amount);
        return true;
    }
}

function universalTransferFrom(
    IERC20 token,
    address from,
    address to,
    uint256 amount
) internal {
    if (amount == 0) {
        return;
    }

    if (isETH(token)) {
        require(from == msg.sender && msg.value >= amount, "Wrong useage of ETH.universalTransferFrom()");
        if (to != address(this)) {
            address(uint160(to)).transfer(amount);
        }
        if (msg.value > amount) {
            // return the remainder
            msg.sender.transfer(msg.value.sub(amount));
        }
```

```
        } else {
            token.safeTransferFrom(from, to, amount);
        }
    }

    function universalTransferFromSenderToThis(IERC20 token, uint256 amount) internal {
        if (amount == 0) {
            return;
        }

        if (isETH(token)) {
            if (msg.value > amount) {
                // return the remainder
                msg.sender.transfer(msg.value.sub(amount));
            }
        } else {
            token.safeTransferFrom(msg.sender, address(this), amount);
        }
    }

    function universalApprove(
        IERC20 token,
        address to,
        uint256 amount
    ) internal {
        if (!isETH(token)) {
            if (amount == 0) {
                token.safeApprove(to, 0);
                return;
            }

            uint256 allowance = token.allowance(address(this), to);
            if (allowance < amount) {
                if (allowance > 0) {
                    token.safeApprove(to, 0);
                }
                token.safeApprove(to, amount);
            }
        }
    }
```

```
    function universalBalanceOf(IERC20 token, address who) internal view returns (uint256) {
        if (isETH(token)) {
            return who.balance;
        } else {
            return token.balanceOf(who);
        }
    }

    function universalDecimals(IERC20 token) internal view returns (uint256) {
        if (isETH(token)) {
            return 18;
        }

        (bool success, bytes memory data) = address(token).staticcall{gas: 10000}(abi.encodeWithSignature("decimals()"));
        if (!success || data.length == 0) {
            (success, data) = address(token).staticcall{gas: 10000}(abi.encodeWithSignature("DECIMALS()"));
        }

        return (success && data.length > 0) ? abi.decode(data, (uint256)) : 18;
    }

    function isETH(IERC20 token) internal pure returns (bool) {
        return (address(token) == address(ZERO_ADDRESS) || address(token) == address(ETH_ADDRESS));
    }

    function notExist(IERC20 token) internal pure returns (bool) {
        return (address(token) == address(-1));
    }
}

// File: contracts/lib/ExternalCall.sol


pragma solidity ^0.6.0;

library ExternalCall {
    // Source: https://github.com/gnosis/MultiSigWallet/blob/master/contracts/MultiSigWallet.sol
    // call has been separated into its own function in order to take advantage
    // of the Solidity's code generator to produce a loop that copies tx.data into memory.
```

```
    function externalCall(
        address target,
        uint256 value,
        bytes memory data,
        uint256 dataOffset,
        uint256 dataLength,
        uint256 gasLimit
    ) internal returns (bool result) {
        if (gasLimit == 0) {
            gasLimit = gasleft() - 40000;
        }
        assembly {
            let x := mload(0x40) // "Allocate" memory for output (0x40 is where "free memory" pointer is stored by convention)
            let d := add(data, 32) // First 32 bytes are the padded length of data, so exclude that
            result := call(
                gasLimit,
                target,
                value,
                add(d, dataOffset),
                dataLength, // Size of the input (in bytes) - this is what fixes the padding problem
                x,
                0 // Output is ignored, therefore the output size is zero
            )
        }
    }
}

// File: contracts/OpenOceanExchange.sol


pragma solidity ^0.6.0;









contract TokenSpender {
    using SafeERC20 for IERC20;
```

```
    address public owner;

    constructor() public {
        owner = msg.sender;
    }

    function claimToken(
        IERC20 token,
        address who,
        address dest,
        uint256 amount
    ) external {
        require(msg.sender == owner, "Access restricted");
        token.safeTransferFrom(who, dest, amount);
    }
}

contract OpenOceanExchange is Shutdownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    using UniversalERC20 for IERC20;
    using ExternalCall for address;

    TokenSpender public spender;

    event Order(address indexed sender, IERC20 indexed inToken, IERC20 indexed outToken, uint256 inAmount, uint256
outAmount);

    event Swapped(
        IERC20 indexed inToken,
        IERC20 indexed outToken,
        address indexed referrer,
        uint256 inAmount,
        uint256 outAmount,
        uint256 fee,
        uint256 referrerFee
    );

    constructor(address _owner) public {
        spender = new TokenSpender();
```

```
        transferOwnership(_owner);
    }

    receive() external payable notShutdown {
        require(msg.sender != tx.origin);
    }

    function swap(
        IERC20 inToken,
        IERC20 outToken,
        uint256 inAmount,
        uint256 minOutAmount,
        uint256, /*guaranteedAmount*/
        address payable referrer,
        address[] memory addressesToCall,
        bytes memory dataToCall,
        uint256[] memory offsets,
        uint256[] memory gasLimitsAndValues
    ) public payable notShutdown returns (uint256 outAmount) {
        require(minOutAmount > 0, "Min out amount should be greater than zero");
        require(addressesToCall.length > 0, "Call data should exists");
        require((msg.value != 0) == inToken.isETH(), "OpenOcean: msg.value should be used only for ETH swap");

        if (!inToken.isETH()) {
            spender.claimToken(inToken, msg.sender, address(this), inAmount);
        }

        for (uint256 i = 0; i < addressesToCall.length; i++) {
            require(addressesToCall[i] != address(spender), "Access denied");
            require(
                addressesToCall[i].externalCall(
                    gasLimitsAndValues[i] & ((1 << 128) - 1),
                    dataToCall,
                    offsets[i],
                    offsets[i + 1] - offsets[i],
                    gasLimitsAndValues[i] >> 128
                )
            );
        }
```

```
        inToken.universalTransfer(msg.sender, inToken.universalBalanceOf(address(this)));
        outAmount = outToken.universalBalanceOf(address(this));

        require(outAmount >= minOutAmount, "Return amount less than the minimum required amount");
        outToken.universalTransfer(msg.sender, outAmount);

        emit Order(msg.sender, inToken, outToken, inAmount, outAmount);
        emit Swapped(inToken, outToken, referrer, inAmount, outAmount, 0, 0);
    }
}
```

**Official Website**
www.slowmist.com

**E-mail**
team@slowmist.com

**Twitter**
@SlowMist_Team

**Github**
https://github.com/slowmist