KNOWNSEC

# Smart Contract Audit Report

Safe State

Security

★ ★ ★ ★ ★

# Version description

| Revised man | Revised content | Revised | version number | Reviewer |
|---|---|---|---|---|
| Yifeng Luo | Document creation and editing | **2020/9/22** | V1.0 | Haojie Xu |

## Document information

| Name | Document version number | Number | Privacy level |
|---|---|---|---|
| **InitializableAdminUpgradeabilityProxy Smart Contract Audit Report** | V1.0 | 【InitializableAdminUpgradeabilityProxy-DMSJ-20200922】 | Open project team |

## Copyright statement

# 目录

# 1. Review

The effective testing time of this report is from **September 19,2020** to **September 22,2020**. During this period, the Knownsec engineers audited the safety and regulatory aspects of **InitializableAdminUpgradeabilityProxy** smart contract code.

In this test, engineers comprehensively analyzed common vulnerabilities of smart contracts (Chapter 3) and It was discovered that denial of service attack detection low risk; so it's evaluated as **Security.**

**The result of the safety auditing:** Pass

Since the test process is carried out in a non-production environment, all the codes are the latest backups. We communicates with the relevant interface personnel, and the relevant test operations are performed under the controllable operation risk to avoid the risks during the test..

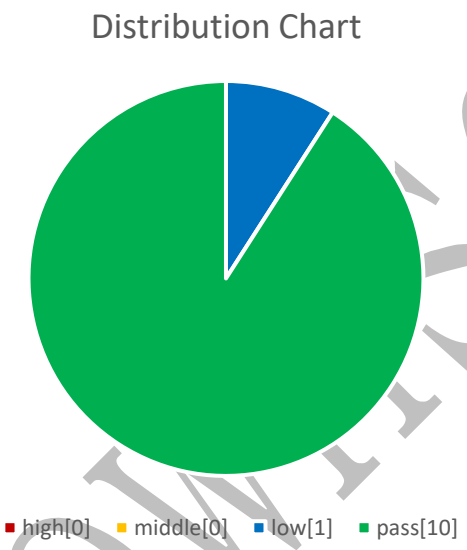
Target information for this test:

| Module name | content |
| --- | --- |
| Token name | InitializableAdminUpgradeabilityProxy |
| Code type | Token code |
| Code language | solidity |

# 2. Analysis of code vulnerability

## 2.1. Distribution of vulnerability Levels

| Vulnerability statistics | | | |
|---|---|---|---|
| high | Middle | low | pass |
| 0 | 0 | 1 | 10 |



## 2.2. **Audit result summary**

| Result | | | |
|---|---|---|---|
| Test project | Test content | status | description |
| Smart Contract | Reentrancy | Pass | Check the call.value() function for security |
| | Arithmetic Issues | Pass | Check add and sub functions |
| | Access Control | Pass | Check the operation access control |
| | Unchecked Return Values For Low Level Calls | Pass | Check the currency conversion method. |
| | Bad Randomness | Pass | Check the unified content filter |
| | Transaction ordering dependence | Pass | Check the transaction ordering dependence |
| | Denial of service attack detection | Low | Check whether the code has a resource abuse problem when using a resource |
| | Logic design Flaw | Pass | Examine the security issues associated with business design in intelligent contract codes. |
| | USDT Fake Deposit Issue | Pass | Check for the existence of USDT Fake Deposit Issue |
| | Adding tokens | Pass | It is detected whether there is a function in the token contract that may increase the total amounts of tokens |
| | Freezing accounts bypassed | Pass | It is detected whether there is an unverified token source account, an originating account, and whether the target account is frozen. |

# 3. Result analysis

## 3.1. Reentrancy【Pass】

The Reentrancy attack, probably the most famous Ethereum vulnerability， led to a hard fork of Ethereum.

When the low level call() function sends ether to the msg.sender address, it becomes vulnerable; if the address is a smart contract, the payment will trigger its fallback function with what's left of the transaction gas.

**Test results**: No related vulnerabilities in smart contract code

**Safety advice**： None

## 3.2. Arithmetic Issues【Pass】

Also known as integer overflow and integer underflow. Solidity can handle up to 256 digits (2^256-1), The largest number increases by 1 will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum numeric value.

Integer overflows and underflows are not a new class of vulnerability, but they are especially dangerous in smart contracts. Overflow can lead to incorrect results, especially if the probability is not expected, which may affect the reliability and security of the program.

**Test results**: No related vulnerabilities in smart contract code

**Safety advice**： None

## 3.3. Access Control【Pass】

Access Control issues are common in all programs,Also smart contracts. The famous Parity Wallet smart contract has been affected by this issue.

**Test results**: No related vulnerabilities in smart contract code

**Safety advice**： None。

## 3.4. Unchecked Return Values For Low Level Calls

【Pass】

Also known as or related to silent failing sends, unchecked-send. There are transfer methods such as transfer(), send(), and call.value() in Solidity and can be used to send Ether to an address. The difference is: transfer will be thrown when failed to send, and rollback; only 2300gas will be passed for call to prevent reentry attacks; send will return false if send fails; only 2300gas will be passed for call to prevent reentry attacks; If .value fails to send, it will return false; passing all available gas calls (which can be restricted by passing in the gas_value parameter) cannot effectively prevent reentry attacks.

If the return value of the send and call.value switch functions is not been checked in the code, the contract will continue to execute the following code,and it may have caused unexpected results due to Ether sending failure.

**Test results**: No related vulnerabilities in smart contract code

**Safety advice**：None。

## 3.5. Bad Randomness【Pass】

Smart Contract May Need to Use Random Numbers. While Solidity offers functions and variables that can access apparently hard-to-predict values just as block.number and block.timestamp. they are generally either more public than they seem or subject to miners' influence. Because these sources of randomness are to an extent predictable, malicious users can generally replicate it and attack the function relying on its unpredictablility.

**Test results**: No related vulnerabilities in smart contract code

**Safety advice**：None。

## 3.6. **Transaction ordering dependence【Pass】**

Since miners always get rewarded via gas fees for running code on behalf of externally owned addresses (EOA), users can specify higher fees to have their transactions mined more quickly. Since the Ethereum blockchain is public, everyone can see the contents of others' pending transactions.

This means if a given user is revealing the solution to a puzzle or other valuable secret, a malicious user can steal the solution and copy their transaction with higher fees to preempt the original solution.

**Test results**: No related vulnerabilities in smart contract code

**Safety advice**：None。

## 3.7. **Denial of service attack detection【Low】**

In the ethernet world, denial of service is deadly, and smart contracts under attack of this type may never be able to return to normal. There may be a number of reasons for a denial of service in smart contracts, including malicious behavior as a recipient of transactions, gas depletion caused by artificially increased computing gas, and abuse of access control to access the private components of the intelligent contract. Take advantage of confusion and neglect, etc.

**Test results**: After testing, there is an error in the smart contract code because of the user's owner access control strategy, which will cause the user to permanently lose control.

```
function _setAdmin(address newAdmin) internal {
  bytes32 slot = ADMIN_SLOT;

  assembly {
    sstore(slot, newAdmin)
  }
}
```

**Safety advice**: For the conversion of control authority, attention should be paid to the determination of user ownership to avoid permanent loss of control.

## 3.8. Logical design Flaw【Pass】

Detect the security problems related to business design in the contract code.

**Test results**: No related vulnerabilities in smart contract code.

**Safety advice**: None.

## 3.9. USDT Fake Deposit Issue【Pass】

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

**Detection results**: No related vulnerabilities in smart contract code..

**Safety advice:** none

## 3.10. Adding tokens【Pass】

It is detected whether there is a function in the token contract that may increase the total amount of tokens after the total amount of tokens is initialized.

**Detection results**: No related vulnerabilities in smart contract code..

**Safety advice:** none

## 3.11. Freezing accounts bypassed【Pass】

In the token contract, when transferring the token, it is detected whether there is an unverified token source account, an originating account, and whether the target account is frozen.

**Detection results:** No related vulnerabilities in smart contract code.

**Safety advice:** none.

# 4. Appendix A：Contract code

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-19
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;


/**
 * @title Proxy
 * @dev Implements delegation of calls to other contracts, with proper
 * forwarding of return values and bubbling of failures.
 * It defines a fallback function that delegates all calls to the address
 * returned by the abstract _implementation() internal function.
 */
abstract contract Proxy {
  /**
   * @dev Fallback function.
   * Implemented entirely in `_fallback`.
   */
  fallback () payable external {
    _fallback();
  }

  receive () payable external {
    _fallback();
  }

  /**
   * @return The Address of the implementation.
   */
  function _implementation() virtual internal view returns (address);

  /**
   * @dev Delegates execution to an implementation contract.
   * This is a low level function that doesn't return to its internal call site.
   * It will return to the external caller whatever the implementation returns.
   * @param implementation Address to delegate.
   */
  function _delegate(address implementation) internal {
    assembly {
      // Copy msg.data. We take full control of memory in this inline assembly
      // block because it will not return to Solidity code. We overwrite the
      // Solidity scratch pad at memory position 0.
      calldatacopy(0, 0, calldatasize())

      // Call the implementation.
      // out and outsize are 0 because we don't know the size yet.
      let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

      // Copy the returned data.
      returndatacopy(0, 0, returndatasize())

      switch result
      // delegatecall returns 0 on error.
      case 0 { revert(0, returndatasize()) }
      default { return(0, returndatasize()) }
    }
  }

  /**
   * @dev Function that is run as the first thing in the fallback function.
   * Can be redefined in derived contracts to add functionality.
   * Redefinitions must call super._willFallback().
   */
  function _willFallback() virtual internal;

  /**
   * @dev fallback implementation.
   * Extracted to enable manual triggering.
   */
```

```
  function _fallback() internal {
    if(OpenZeppelinUpgradesAddress.isContract(msg.sender) && msg.data.length == 0 &&
gasleft() <= 2300)         // for receive ETH only from other contract
        return;
    _willFallback();
    _delegate(_implementation());
  }
}


/**
 * @title BaseUpgradeabilityProxy
 * @dev This contract implements a proxy that allows to change the
 * implementation address to which it will delegate.
 * Such a change is called an implementation upgrade.
 */
abstract contract BaseUpgradeabilityProxy is Proxy {
  /**
   * @dev Emitted when the implementation is upgraded.
   * @param implementation Address of the new implementation.
   */
  event Upgraded(address indexed implementation);

  /**
   * @dev Storage slot with the address of the current implementation.
   * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, and
is
   * validated in the constructor.
   */
  bytes32 internal constant IMPLEMENTATION_SLOT =
0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc;

  /**
   * @dev Returns the current implementation.
   * @return impl Address of the current implementation
   */
  function _implementation() override internal view returns (address impl) {
    bytes32 slot = IMPLEMENTATION_SLOT;
    assembly {
      impl := sload(slot)
    }
  }

  /**
   * @dev Upgrades the proxy to a new implementation.
   * @param newImplementation Address of the new implementation.
   */
  function _upgradeTo(address newImplementation) internal {
    _setImplementation(newImplementation);
    emit Upgraded(newImplementation);
  }

  /**
   * @dev Sets the implementation address of the proxy.
   * @param newImplementation Address of the new implementation.
   */
  function _setImplementation(address newImplementation) internal {
    require(OpenZeppelinUpgradesAddress.isContract(newImplementation), "Cannot set a
proxy implementation to a non-contract address");

    bytes32 slot = IMPLEMENTATION_SLOT;

    assembly {
      sstore(slot, newImplementation)
    }
  }
}


/**
 * @title BaseAdminUpgradeabilityProxy
 * @dev This contract combines an upgradeability proxy with an authorization
 * mechanism for administrative tasks.
 * All external functions in this contract must be guarded by the
 * `ifAdmin` modifier. See ethereum/solidity#3864 for a Solidity
 * feature proposal that would enable this to be done automatically.
 */
```

```
contract BaseAdminUpgradeabilityProxy is BaseUpgradeabilityProxy {
  /**
   * @dev Emitted when the administration has been transferred.
   * @param previousAdmin Address of the previous admin.
   * @param newAdmin Address of the new admin.
   */
  event AdminChanged(address previousAdmin, address newAdmin);

  /**
   * @dev Storage slot with the admin of the contract.
   * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
   * validated in the constructor.
   */

  bytes32 internal constant ADMIN_SLOT =
0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;

  /**
   * @dev Modifier to check whether the `msg.sender` is the admin.
   * If it is, it will run the function. Otherwise, it will delegate the call
   * to the implementation.
   */
  modifier ifAdmin() {
    if (msg.sender == _admin()) {
      _;
    } else {
      _fallback();
    }
  }

  /**
   * @return The address of the proxy admin.
   */
  function admin() external ifAdmin returns (address) {
    return _admin();
  }

  /**
   * @return The address of the implementation.
   */
  function implementation() external ifAdmin returns (address) {
    return _implementation();
  }

  /**
   * @dev Changes the admin of the proxy.
   * Only the current admin can call this function.
   * @param newAdmin Address to transfer proxy administration to.
   */
  function changeAdmin(address newAdmin) external ifAdmin {
    require(newAdmin != address(0), "Cannot change the admin of a proxy to the zero
address");
    emit AdminChanged(_admin(), newAdmin);
    _setAdmin(newAdmin);
  }

  /**
   * @dev Upgrade the backing implementation of the proxy.
   * Only the admin can call this function.
   * @param newImplementation Address of the new implementation.
   */
  function upgradeTo(address newImplementation) external ifAdmin {
    _upgradeTo(newImplementation);
  }

  /**
   * @dev Upgrade the backing implementation of the proxy and call a function
   * on the new implementation.
   * This is useful to initialize the proxied contract.
   * @param newImplementation Address of the new implementation.
   * @param data Data to send as msg.data in the low level call.
   * It should include the signature and the parameters of the function to be called,
as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-
argument-encoding.
   */
```

```
  function upgradeToAndCall(address newImplementation, bytes calldata data) payable
external ifAdmin {
    _upgradeTo(newImplementation);
    (bool success,) = newImplementation.delegatecall(data);
    require(success);
  }

  /**
   * @return adm The admin slot.
   */
  function _admin() internal view returns (address adm) {
    bytes32 slot = ADMIN_SLOT;
    assembly {
      adm := sload(slot)
    }
  }

  /**
   * @dev Sets the address of the proxy admin.
   * @param newAdmin Address of the new proxy admin.
   */
  function _setAdmin(address newAdmin) internal {
    bytes32 slot = ADMIN_SLOT;

    assembly {
      sstore(slot, newAdmin)
    }
  }

  /**
   * @dev Only fall back when the sender is not the admin.
   */
  function _willFallback() virtual override internal {
    require(msg.sender != _admin(), "Cannot call fallback function from the proxy
admin");
    //super._willFallback();
  }
}

interface IAdminUpgradeabilityProxyView {
  function admin() external view returns (address);
  function implementation() external view returns (address);
}


/**
 * @title UpgradeabilityProxy
 * @dev Extends BaseUpgradeabilityProxy with a constructor for initializing
 * implementation and init data.
 */
abstract contract UpgradeabilityProxy is BaseUpgradeabilityProxy {
  /**
   * @dev Contract constructor.
   * @param _logic Address of the initial implementation.
   * @param _data Data to send as msg.data to the implementation to initialize the
proxied contract.
   * It should include the signature and the parameters of the function to be called,
as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-
argument-encoding.
   * This parameter is optional, if no data is given the initialization call to proxied
contract will be skipped.
   */
  constructor(address _logic, bytes memory _data) public payable {
    assert(IMPLEMENTATION_SLOT ==
bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1));
    _setImplementation(_logic);
    if(_data.length > 0) {
      (bool success,) = _logic.delegatecall(_data);
      require(success);
    }
  }

  //function _willFallback() virtual override internal {
    //super._willFallback();
  //}
}
```

```
/**
 * @title AdminUpgradeabilityProxy
 * @dev Extends from BaseAdminUpgradeabilityProxy with a constructor for
 * initializing the implementation, admin, and init data.
 */
contract AdminUpgradeabilityProxy is BaseAdminUpgradeabilityProxy, UpgradeabilityProxy
{
  /**
   * Contract constructor.
   * @param _logic address of the initial implementation.
   * @param _admin Address of the proxy administrator.
   * @param _data Data to send as msg.data to the implementation to initialize the
proxied contract.
   * It should include the signature and the parameters of the function to be called,
as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-
argument-encoding.
   * This parameter is optional, if no data is given the initialization call to proxied
contract will be skipped.
   */
  constructor(address _admin, address _logic, bytes memory _data)
UpgradeabilityProxy(_logic, _data) public payable {
    assert(ADMIN_SLOT == bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1));
    _setAdmin(_admin);
  }

  function _willFallback() override(Proxy, BaseAdminUpgradeabilityProxy) internal {
    super._willFallback();
  }
}


/**
 * @title InitializableUpgradeabilityProxy
 * @dev Extends BaseUpgradeabilityProxy with an initializer for initializing
 * implementation and init data.
 */
abstract contract InitializableUpgradeabilityProxy is BaseUpgradeabilityProxy {
  /**
   * @dev Contract initializer.
   * @param _logic Address of the initial implementation.
   * @param _data Data to send as msg.data to the implementation to initialize the
proxied contract.
   * It should include the signature and the parameters of the function to be called,
as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-
argument-encoding.
   * This parameter is optional, if no data is given the initialization call to proxied
contract will be skipped.
   */
  function initialize(address _logic, bytes memory _data) public payable {
    require(_implementation() == address(0));
    assert(IMPLEMENTATION_SLOT ==
bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1));
    _setImplementation(_logic);
    if(_data.length > 0) {
      (bool success,) = _logic.delegatecall(_data);
      require(success);
    }
  }
}


/**
 * @title InitializableAdminUpgradeabilityProxy
 * @dev Extends from BaseAdminUpgradeabilityProxy with an initializer for
 * initializing the implementation, admin, and init data.
 */
contract InitializableAdminUpgradeabilityProxy is BaseAdminUpgradeabilityProxy,
InitializableUpgradeabilityProxy {
  /**
   * Contract initializer.
   * @param _logic address of the initial implementation.
   * @param _admin Address of the proxy administrator.
```

```
   * @param _data Data to send as msg.data to the implementation to initialize the
proxied contract.
   * It should include the signature and the parameters of the function to be called,
as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-
argument-encoding.
   * This parameter is optional, if no data is given the initialization call to proxied
contract will be skipped.
   */
  function initialize(address _admin, address _logic, bytes memory _data) public
payable {
    require(_implementation() == address(0));
    InitializableUpgradeabilityProxy.initialize(_logic, _data);
    assert(ADMIN_SLOT == bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1));
    _setAdmin(_admin);
  }
}


/**
 * Utility library of inline functions on addresses
 *
 * Source https://raw.githubusercontent.com/OpenZeppelin/openzeppelin-
solidity/v2.1.3/contracts/utils/Address.sol
 * This contract is copied here and renamed from the original to avoid clashes in the
compiled artifacts
 * when the user imports a zos-lib contract (that transitively causes this contract to
be compiled and added to the
 * build/artifacts folder) as well as the vanilla Address implementation from an
openzeppelin version.
 */
library OpenZeppelinUpgradesAddress {
    /**
     * Returns whether the target address is a contract
     * @dev This function will return false if invoked during the constructor of a
contract,
     * as the code is not actually created until after the constructor finishes.
     * @param account address of the account to check
     * @return whether the target address is a contract
     */
    function isContract(address account) internal view returns (bool) {
        uint256 size;
        // XXX Currently there is no better way to check if there is a contract in an
address
        // than to check the size of the code at that address.
        // See https://ethereum.stackexchange.com/a/14016/36603
        // for more details about how this works.
        // TODO Check this again before the Serenity release, because all addresses
will be
        // contracts then.
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }
}}
```

# 5. Appendix B: vulnerability risk rating criteria

| Smart contract vulnerability rating standard | |
|---|---|
| **Vulnerability rating** | Vulnerability rating description |
| **High risk vulnerability** | The loophole which can directly cause the contract or the user's fund loss, such as the value overflow loophole which can cause the value of the substitute currency to zero, the false recharge loophole that can cause the exchange to lose the substitute coin, can cause the contract account to lose the ETH or the reentry loophole of the substitute currency, and so on; It can cause the loss of ownership rights of token contract, such as: the key function access control defect or call injection leads to the key function access control bypassing, and the loophole that the token contract can not work properly. Such as: a denial-of-service vulnerability due to sending ETHs to a malicious address, and a denial-of-service vulnerability due to gas depletion. |
| **Middle risk vulnerability** | High risk vulnerabilities that need specific addresses to trigger, such as numerical overflow vulnerabilities that can be triggered by the owner of a token contract, access control defects of non-critical functions, and logical design defects that do not result in direct capital losses, etc. |
| **Low risk vulnerability** | A vulnerability that is difficult to trigger, or that will harm a limited number after triggering, such as a numerical overflow that requires a large number of ETH or tokens to trigger, and a vulnerability that the attacker cannot directly profit from after triggering a numerical overflow. Rely on risks by specifying the order of transactions triggered by a high gas. |

# 6. Appendix C： Introduction of test tool

## 6.1. Manticore

Manticore is a symbolic execution tool for analysis of binaries and smart contracts. It discovers inputs that crash programs via memory safety violations. Manticore records an instruction-level

trace of execution for each generated input and exposes programmatic access to its analysis engine via a Python API.

## 6.2. Oyente

Oyente is a smart contract analysis tool that Oyente can use to detect common bugs in smart contracts, such as reentrancy, transaction ordering dependencies, and more. More conveniently, Oyente's design is modular, so this allows advanced users to implement and insert their own detection logic to check for custom attributes in their contracts.

## 6.3. securify.sh

Securify can verify common security issues with Ethereum smart contracts, such as transactional out-of-order and lack of input validation. It analyzes all possible execution paths of the program while fully automated. In addition, Securify has a specific language for specifying vulnerabilities. Securify can keep an eye on current security and other reliability issues.

## 6.4. Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

## 6.5. MAIAN

MAIAN is an automated tool for finding Ethereum smart contract vulnerabilities. Maian deals with the contract's bytecode and tries to establish a series of transactions to find and confirm errors.

## 6.6. ethersplay

Ethersplay is an EVM disassembler that contains related analysis tools.

## 6.7. ida-evm

Ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

## 6.8. Remix-ide

Remix is a browser-based compiler and IDE that allows users to build Ethereum contracts and debug transactions using the Solidity language.

## 6.9. Knownsec Penetration Tester Special Toolkit

Knownsec penetration tester special tool kit, developed and collected by Knownsec penetration testing engineers, includes batch automatic testing tools dedicated to testers, self-developed tools, scripts, or utility tools.