知道创宇

密级 ★	项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

版本说明

修订人	修订内容	修订时间	版本号	审阅人
罗逸锋	**编写文档**	**2020/9/22**	**V1.0**	**徐昊杰**

文档信息

文档名称	文档版本号	文档编号	保密级别
InitializableAdminUpgradeabilityProxy 合约审计报告	V1.0	【InitializableAdminUpgradeabilityProxy-DMSJ-20200922】	项目组公开

版权声明

目录

1. 综述

本次报告有效测试时间是从 2020 年 9 月 19 日开始到 2020 年 9 月 22 日结束，在此期间针对 InitializableAdminUpgradeabilityProxy 智能合约代码的安全性和规范性进行审计并以此作为报告统计依据。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，发现了拒绝服务攻击风险，故综合评定为安全。

本次智能合约安全审计结果：通过

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

本次测试的目标信息：

模块名称	内容
Token 名称	InitializableAdminUpgradeabilityProxy
代码类型	代币代码
代码语言	solidity

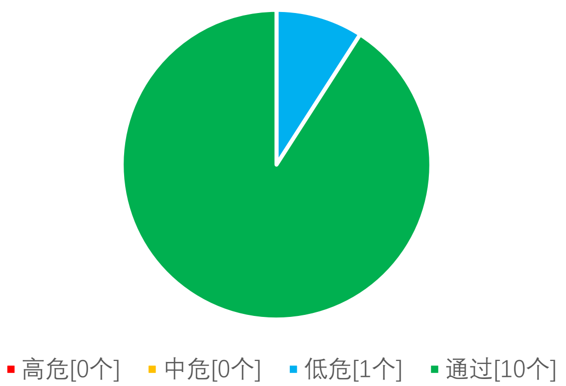
2. 代码漏洞分析

2.1. 漏洞等级分布

本次漏洞风险按等级统计:

漏洞风险等级个数统计表			
高危	中危	低危	通过
0	0	1	10

风险等级分布图



2.2. 审计结果汇总

审计结果			
测试项目	测试内容	状态	描述
智能合约	重入攻击检测	通过	检查 call.value() 函数使用安全
	数值溢出检测	通过	检查 add 和 sub 函数使用安全
	访问控制缺陷检测	通过	检查各操作访问权限控制
	未验证返回值的调用	通过	检查转币方法看是否验证返回值
	错误使用随机数检测	通过	检查是否具备统一的内容过滤器
	事务顺序依赖检测	通过	检查是否存在事务顺序依赖风险
	拒绝服务攻击检测	低危	检查代码在使用资源时是否存在资源滥用问题
	逻辑设计缺陷检测	通过	检查智能合约代码中与业务设计相关的安全问题
	假充值漏洞检测	通过	检查智能合约代码中是否存在假充值漏洞
	增发代币漏洞检测	通过	检查智能合约中是否存在增发代币的功能
	冻结账户绕过检测	通过	检查转移代币中是否存在未校验冻结账户的问题

3. 代码审计结果分析

3.1. 重入攻击检测【通过】

重入漏洞是最著名的以太坊智能合约漏洞，曾导致了以太坊的分叉（The DAO hack）。

Solidity 中的 call.value()函数在被用来发送 Ether 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 Ether 的操作发生在实际减少发送者账户的余额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在相关 call 外部合约调用。

安全建议：无

3.2. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。

Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.3. 访问控制检测【通过】

访问控制缺陷是所有程序中都可能存在的安全风险，智能合约也同样会存在类似问题，著名的 Parity Wallet 智能合约就受到过该问题的影响。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.4. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 Ether，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继续执行后面的代码，可能由于 Ether 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在相关漏洞。

安全建议：无。

3.5. 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问

明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数载一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该问题。

安全建议：无。

3.6. 事务顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 gas 费用，因此用户可以指定更高的费用以便更快地开展交易。由于以太坊区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，智能合约代码中不存在相关漏洞。

安全建议：无

3.7. 拒绝服务攻击【低危】

在以太坊的世界中，拒绝服务是致命的，遭受该类型攻击的智能合约可能永远无法恢复正常工作状态。导致智能合约拒绝服务的原因可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中存在因为对于用户 owner 访问控制策略出错，这里就会导致用户永久失去控制权。

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

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

安全建议： 对于控制权限的转换需要注意对于用户所有权的确定，避免造成控制权的永久丢失。

3.8. 逻辑设计缺陷【通过】

检测智能合约代码中与业务设计相关的安全问题。

检测结果： 经检测，智能合约代码中不存在相关漏洞。

安全建议： 无。

3.9. 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(ABBT.sender)的余额检查用的是 if 判断方式，当 balances[ABBT.sender] < value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

检测结果： 经检测，智能合约代码中不存在相关漏洞。

安全建议： 无。

3.10. 增发代币漏洞【通过】

检测在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中不存在相关漏洞。

安全建议：无

3.11. 冻结账户绕过【通过】

检测代币合约中在转移代币时，是否存在未校验代币来源账户、发起账户、目标账户是否被冻结的操作。

检测结果：经检测，智能合约代码中不存在该问题。

安全建议：无。

4. 附录 A：合约代码

```
/**
 *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. 附录 B：漏洞风险评级标准

智能合约漏洞评级标准	
漏洞评级	**漏洞评级说明**
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失 ETH 或代币的重入漏洞等； 能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call 注入导致关键函数访问控制绕过等； 能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送 ETH 导致的拒绝服务漏洞、因 gas 耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等。
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 ETH 或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高 gas 触发的事务顺序依赖风险等。

6. 附录 C：漏洞测试工具简介

6.1. MaABBTicore

MaABBTicore 是一个分析二进制文件和智能合约的符号执行工具，MaABBTicore 包含一个符号以太坊虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。 与二进制文件一样，MaABBTicore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

6.2. OyeABBTe

OyeABBTe 是一个智能合约分析工具，OyeABBTe 可以用来检测智能合约中常见的 bug，比如 reeABBTrancy、事务排序依赖等等。更方便的是，OyeABBTe 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合约中自定义的属性。

6.3. securify.sh

Securify 可以验证以太坊智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

6.4. Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

6.5. MAIAN

MAIAN 是一个用于查找以太坊智能合约漏洞的自动化工具，Maian 处理合约的字节码，并尝试建立一系列交易以找出并确认错误。

6.6. ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

6.7. ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

6.8. Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建以太坊合约并调试交易。

6.9. 知道创宇渗透测试人员专用工具包

知道创宇渗透测试人员专用工具包，由知道创宇渗透测试工程师研发，收集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587
邮　　箱 sec@knownsec.com
官　　网 www.knownsec.com
地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509