知道创宇

密级 ★ | 项目组公开

智能合约审计报告

Gate USD (USDG)

FiatTokenProxy

安全状态

安全

★ ★ ★ ★ ★

主测人：知道创宇区块链安全研究团队

版本说明

修订内容	时间	修订者	版本号
编写文档	20200930	知道创宇区块链安全研究团队	V1.0

文档信息

文档名称	文档版本	文档编号	保密级别
Gate USD (USDG) FiatTokenProxy 智能合约审计报告	V1.0	FIATTOKENPROXY-ZNNY-20200930	项目组公开

声明

目录

1. 综述

本次报告有效测试时间是从 2020 年 9 月 29 日开始到 2020 年 9 月 30 日结束，在此期间针对 **Gate USD (USDG) FiatTokenProxy 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，未发现相关安全问题，综合评定为**通过**。

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：

条目	描述
Token 名称	Gate USD (USDG) FiatTokenProxy
合约地址	0x1c0b9c1995ea61f8fb505e4d94aa32b0c024899e
代码类型	代币代码、DeFi 协议代码、以太坊智能合约代码
代码语言	solidity

合约文件及哈希：

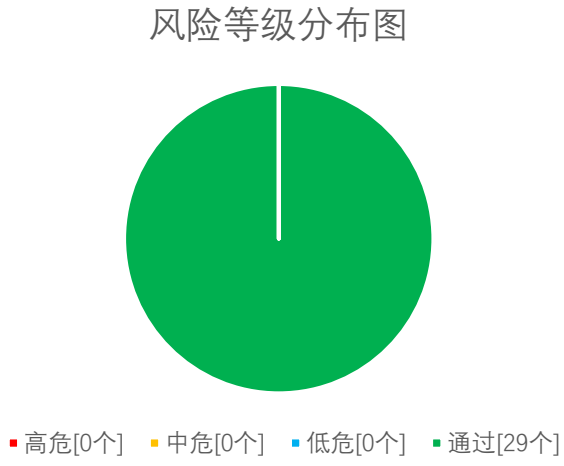
合约文件	MD5
FiatTokenProxy.sol	76EB142D5014A15FADB39B074F042A7A
Address.sol	1A1BA1CF1A033D8ED33674E8447ADCC6
Proxy.sol	EB1B511F8E3F4D42E1EC5DAD26379CE7
UpgradeabilityProxy.sol	2EB84F9E3DFDCA2F6CF9678848002213
AdminUpgradeabilityProxy.sol	EEA51BFA90778FBD73EA572C7E714904

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:

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	29



2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
智能合约	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	tx.progin 身份验证	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	逻辑设计缺陷检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	增发代币漏洞检测	通过	经检测，不存在该安全问题
	冻结账户绕过检测	通过	经检测，不存在该安全问题。
	编译器版本安全	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	require/assert 的使用	通过	经检测，不存在该安全问题。

	gas 消耗检测	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	不安全的接口使用	通过	经检测，不存在该安全问题。

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该安全问题。

安全建议：无。

3.2 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景 00009 在智能合约中也比较普遍。。

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 重排攻击检测【通过】

重排攻击是指矿工或其他方试图通过将自己的信息插入列表(list)或映射(mapping)中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

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。

3.4 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，智能合约代码中未直接使用算数运算。

安全建议：无。

3.5 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、函数、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20，而 5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

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。

3.6 访问控制检测【通过】

合约中不同函数应设置合理的权限

检查合约中各函数是否正确使用了 public、private 等关键词进行可见性修饰，检查合约是否正确定义并使用了 modifier 对关键函数进行访问限制，避免越权导致的问题。

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。

3.7 tx.origin 身份验证【通过】

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

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。

3.8 call 注入攻击【通过】

call 函数调用时，应该做严格的权限控制，或直接写死 call 调用的函数。

检测结果：经检测，智能合约中 call 未做其他操作，不存在此漏洞。

安全建议：无。

3.9 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该安全问题，对 call.value 转币函数返回值做了校验。

安全建议：无。

3.10 未初始化的储存指针【通过】

在 solidity 中允许一个特殊的数据结构为 struct 结构体，而函数内的局部变量默认使用 storage 或 memory 储存。

而存在 storage(存储器)和 memory(内存)是两个不同的概念，solidity 允许指针指向一个未初始化的引用，而未初始化的局部 stroage 会导致变量指向其他储存变量，导致变量覆盖，甚至其他更严重的后果，在开发中应该避免在函数中初始化 struct 变量。

检测结果：经检测，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

3.11 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，即这些随机数在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安全问题。

安全建议：无。

3.12 交易顺序依赖【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该安全问题。

安全建议：无。

3.13 拒绝服务攻击【通过】

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，智能合约代码中不存在该安全问题。

安全建议：无。

3.14 逻辑设计缺陷【通过】

检查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该安全问题。

安全建议：无。

3.15 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.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该安全问题。

安全建议：无。

3.16 增发代币漏洞【通过】

检查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该安全问题。

安全建议：该问题不属于安全问题，但部分交易所会限制增发函数的使用，具体情况需根据交易所的要求而定。

3.17 冻结账户绕过【通过】

检查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安全问题。

安全建议：无。

3.18 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本 0.5.8 以上，不存在该安全问题。

安全建议：无。

3.19 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

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。

3.20 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

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。

3.21 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

检测结果：经检测，智能合约代码中无算数运算操作。

安全建议：无。

3.22 require/assert 的使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

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。

3.23 gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制

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。

3.24 fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

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。

3.25 owner 权限控制【通过】

检查合约代码实现中的 owner 是否具有过高的权限。例如，任意修改其他

账户余额等。

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。

3.26 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞

call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中

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。

3.27 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题

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。

3.28 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有 900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。

检查合约代码实现中是否存在有依赖于时间戳的关键功能

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。

3.29 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

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。

4. 附录 A：合约代码

本次测试代码来源：

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-27
*/

// File: contracts/upgradeability/Proxy.sol


pragma solidity 0.6.12;// knownsec  指定编译器版本

/**
 * @notice 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.
 *                @dev                Forked                from                https://github.com/zeppelinos/zos-lib/blob/8a16ef3ad17ec7430e3a9d2b5e3f39b8204f8c8d/contracts/upgradeability/Proxy.sol
 * Modifications:
 * 1. Reformat and conform to Solidity 0.6 syntax (5/13/20)
 */
abstract contract Proxy {
    /**
     * @dev Fallback function.
     * Implemented entirely in `_fallback`.
     */
    fallback() external payable {
        _fallback();
    }//回退

    /**
     * @return The Address of the implementation.
     */
    function _implementation() internal virtual 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 {//knownsec  授权
        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() internal virtual {}
```

```
    /**
     * @dev fallback implementation.
     * Extracted to enable manual triggering.
     */
    function _fallback() internal {
        _willFallback();
        _delegate(_implementation());
    }
}

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

pragma solidity ^0.6.2;// knownsec 指定编译器版本

/**
 * @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 {// knownsec 转账
        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 }("");// knownsec 利用 call 转账，call 不报错，因此后续 require 校验
        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(000, 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: contracts/upgradeability/UpgradeabilityProxy.sol


pragma solidity 0.6.12;


/**
 * @notice 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.
 *                @dev                Forked                from                https://github.com/zeppelinos/zos-
```

```
lib/blob/8a16ef3ad17ec7430e3a9d2b5e3f39b8204f8c8d/contracts/upgradeability/UpgradeabilityProxy.sol
 * Modifications:
 * 1. Reformat, conform to Solidity 0.6 syntax, and add error messages (5/13/20)
 * 2. Use Address utility library from the latest OpenZeppelin (5/13/20)
 */
contract UpgradeabilityProxy is Proxy {
    /**
     * @dev Emitted when the implementation is upgraded.
     * @param implementation Address of the new implementation.
     */
    event Upgraded(address implementation);

    /**
     * @dev Storage slot with the address of the current implementation.
     * This is the keccak-256 hash of "org.zeppelinos.proxy.implementation", and is
     * validated in the constructor.
     */
    bytes32
        private constant IMPLEMENTATION_SLOT =
0x7050c9e0f4ca769c69bd3a8ef740bc37934f8e2c036e5a723fd8ee048ed3f8c3;

    /**
     * @dev Contract constructor.
     * @param implementationContract Address of the initial implementation.
     */
    constructor(address implementationContract) public {
        assert(
            IMPLEMENTATION_SLOT ==
                keccak256("org.zeppelinos.proxy.implementation")
        );

        _setImplementation(implementationContract);
    }

    /**
     * @dev Returns the current implementation.
     * @return impl Address of the current implementation
     */
    function _implementation() internal override view returns (address impl) {
        bytes32 slot = IMPLEMENTATION_SLOT;
        assembly {
            impl := sload(slot)//knownsec 内联汇编 返回 storage 中 IMPLEMENTATION_SLOT 的 value
        }
    }

    /**
     * @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) private {
        require(
            Address.isContract(newImplementation),
            "Cannot set a proxy implementation to a non-contract address"
        );

        bytes32 slot = IMPLEMENTATION_SLOT;

        assembly {
            sstore(slot, newImplementation)//防止 storage 中 slot 值为 newImplementation
        }
    }
}

// File: contracts/upgradeability/AdminUpgradeabilityProxy.sol

pragma solidity 0.6.12;


/**
 * @notice This contract combines an upgradeability proxy with an authorization
 * mechanism for administrative tasks.
 *                 @dev                 Forked                 from                 https://github.com/zeppelinos/zos-
lib/blob/8a16ef3ad17ec7430e3a9d2b5e3f39b8204f8c8d/contracts/upgradeability/AdminUpgradeabilityProxy.sol
 * Modifications:
 * 1. Reformat, conform to Solidity 0.6 syntax, and add error messages (5/13/20)
 * 2. Remove ifAdmin modifier from admin() and implementation() (5/13/20)
 */
contract AdminUpgradeabilityProxy is UpgradeabilityProxy {
```

```
    /**
     * @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 "org.zeppelinos.proxy.admin", and is
     * validated in the constructor.
     */
    bytes32
        private                 constant                ADMIN_SLOT                  =
0x10d6a54a4754c8869d6886b5f5d7fbfa5b4522237ea5c60d11bc4e7a1ff9390b;// knownsec ADMIN_SLOT 的值恒
定(bytes32)keccak256("org.zeppelinos.proxy.admin")

    /**
     * @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() {//knownsec 判定发送者满足_admin
        if (msg.sender == _admin()) {
            _;
        } else {
            _fallback();//回退
        }
    }

    /**
     * @dev Contract constructor.
     * It sets the `msg.sender` as the proxy administrator.
     * @param implementationContract address of the initial implementation.
     */
    constructor(address implementationContract)
        public
        UpgradeabilityProxy(implementationContract)
    {
        assert(ADMIN_SLOT    ==    keccak256("org.zeppelinos.proxy.admin"));//    knownsec    计    算
org.zeppelinos.proxy.admin 的 hash256 与 ADMIN_SLOT 对比
// knownsec ADMIN_SLOT 的值恒定(bytes32)keccak256("org.zeppelinos.proxy.admin")
        _setAdmin(msg.sender);// knownsec 相等则设置管理员，当 ADMIN_SLOT 不合法则无法部署，将回
退
    }

    /**
     * @return The address of the proxy admin.
     */
    function admin() external view returns (address) {// knownsec 外部获取 admin
        return _admin();
    }

    /**
     * @return The address of the implementation.
     */
    function implementation() external view 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 {// knownsec 管理权转移，外部调用。ifAdmin
校验发送至地址是否是_admin
        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 {// knownsec 更新 implementation 管理
员可用
        _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/develop/abi-spec.html#function-selector-and-argument-encoding.
     */
    function upgradeToAndCall(address newImplementation, bytes calldata data)// knownsec 更新 implementation 并同步数据 管理员可用
        external
        payable
        ifAdmin
    {
        upgradeTo(newImplementation);
        // prettier-ignore
        // solhint-disable-next-line avoid-low-level-calls
        (bool success,) = address(this).call{value: msg.value}(data);
        // solhint-disable-next-line reason-string
        require(success);
    }

    /**
     * @return adm The admin slot.
     */
    function admin() internal view returns (address adm) {//knownsec 返回 sload( keccak256("org.zeppelinos.proxy.admin") )
        bytes32 slot = ADMIN_SLOT;

        assembly {
            adm := sload(slot)//knownsec 内联汇编 返回 storage 中 ADMIN_SLOT 的 value
        }
    }

    /**
     * @dev Sets the address of the proxy admin.
     * @param newAdmin Address of the new proxy admin.
     */
    function _setAdmin(address newAdmin) internal {//knownsec 设置新的管理员 内部使用
        bytes32 slot = ADMIN_SLOT;//knownsec 临时变量，存储 ADMIN_SLOT

        assembly {
            sstore(slot, newAdmin)//knownsec 内联汇编 sstore 存储 newAdmin 的地址
        }
    }

    /**
     * @dev Only fall back when the sender is not the admin.
     */
    function _willFallback() internal override {
        require(
            msg.sender != admin(),
            "Cannot call fallback function from the proxy admin"
        );//knownsec 校验发送人 不能是 admin
        super._willFallback();
    }
}
// File: contracts/v1/FiatTokenProxy.sol

pragma solidity 0.6.12;


/**
 * @title FiatTokenProxy
 * @dev This contract proxies FiatToken calls and enables FiatToken upgrades
 */
contract FiatTokenProxy is AdminUpgradeabilityProxy {// knownsec 继承相关方法和变量
    constructor(address implementationContract)// knownsec 传入控制地址
        public
        AdminUpgradeabilityProxy(implementationContract)
    {}
}
```

5. 附录 B：安全风险评级标准

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：智能合约安全审计工具简介

6.1 Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 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，Manticore 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 Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 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 知道创宇区块链安全审计人员专用工具包

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