

智能合约安全审计报告





慢雾安全团队于 2020-07-05 日,收到 dForce 团队对 GOLDx 项目智能合约安全审计申请。如下为本次智能合约安全审计细节及结果:

Token 名称:

GOLDx

项目链接:

Github: https://github.com/dforce-network/GOLDx/tree/audit

commit: ea276c17c4c119287896f9322e4369ad156c7a0b

本次审计项及结果:

(其他未知安全漏洞不包含在本次审计责任范围)

序号	审计大类	审计子类	审计结果		
1	溢出审计		通过		
2	条件竞争审计		通过		
2	+778+6×41 Φ XI	权限漏洞审计	通过		
3	权限控制审计	权限过大审计	通过		
		Zeppelin 模块使用安全	通过		
		编译器版本安全	通过		
		硬编码地址安全	通过		
4	安全设计审计	Fallback 函数使用安全	通过		
		显现编码安全	通过		
		函数返回值安全	通过		
		call 调用安全	通过		
5	拒绝服务审计		通过		
6	Gas 优化审计		通过		
7	设计逻辑审计		通过		
8	"假充值"漏洞审计		通过		
9	恶意 Event 事件日志审计		通过		



											1.5	//												
											- 1	Jul.												
												~												
			1											4										
	-7			`-	-	Б		┌.	ţ	н	~	17	· 1	+	-	-	⇤	- 1	7	_	1	$^{\sim}$	Ċ.	
	-7				-		×		Lo.	щ	- 4			4	٠.	1	1		Ť	7	~	Ŧ	_	
	100	<∙	-/		Ŀ	Ľ	-	• •	-)	`\	ŀЬ	J.J.	-	با	_	A.	2	١.	⋗	۹.	_	Œ	-	
												16												
												<^>>												

10	变量声明及作用域审计		通过
11	重放攻击审计	ECDSA 签名重放审计	通过
12	未初始化的存储指针		通过
13	算术精度误差		通过

备注: 审计意见及建议见代码注释 //SlowMist// ······

审计结果:通过

审计编号: 0X002007090002

审计日期: 2020年07月09日

审计团队:慢雾安全团队

(**声明:**慢雾仅就本报告出具前已经发生或存在的事实出具本报告,并就此承担相应责任。对于出具以后发生或存在的事实,慢雾无法判断其智能合约安全状况,亦不对此承担责任。本报告所作的安全审计分析及其他内容,仅基于信息提供者截至本报告出具时向慢雾提供的文件和资料(简称"已提供资料")。慢雾假设:已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的,慢雾对由此而导致的损失和不利影响不承担任何责任。慢雾仅对该项目的安全情况进行约定内的安全审计并出具了本报告,慢雾不对该项目背景及其他情况进行负责。)

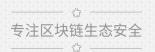
总结:此为代币(token)合约,不包含锁仓(tokenVault)部分。合约代币总量可变。用户可以通过 mint 函数和 burn 函数进行代币兑换。使用了 SafeMath 安全模块,值得称赞的做法。合约不存在溢出问题。

在审计过程中,我们发现以下问题:

- 1. Auth 可以通过 addBlacklist 函数将任意用户添加到黑名单中。
- 2. Auth 可以通过 retrieveBlackAddress 函数将黑名单用户的余额转移到 owner 的帐户中。
- 3. Auth 可以通过 wipeBlackAddress 函数清除黑名单用户的余额。
- 4. Auth 可以通过 setUnit 函数随意更改代币兑换汇率。如果 auth 随意更改汇率,可能会导致用户的资产在代币兑换时受到影响。如果 auth 在用户兑换代币的同时更改了汇率,则可能会导致条件竞争问题。根据项目方反馈,任何代币兑换都固定以克为单位,auth 不会随意更改汇率。

建议将 auth 设置为多签合约,以降低被攻击风险。



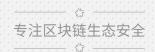


合约源代码如下:

IERC20.sol

//SlowMist// 合约不存在溢出、条件竞争问题 pragma solidity 0.5.16; * @dev Interface of the ERC20 standard as defined in the EIP. Does not include * the optional functions; to access them see {ERC20Detailed}. interface IERC20 { * @dev Returns the amount of tokens in existence. function totalSupply() external view returns (uint); * @dev Returns the amount of tokens owned by 'account'. */ function balanceOf(address account) external view returns (uint); * @dev Returns the decimals of tokens.. function decimals() external view returns (uint); * @dev Moves `amount` tokens from the caller's account to `recipient`. * Emits a {Transfer} event. function transfer(address recipient, uint amount) external; * @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 (uint);





```
* @dev Sets 'amount' as the allowance of 'spender' over the caller's tokens.
     * 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
     * desireo value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     * Emits an {Approval} event.
    function approve(address spender, uint amount) external;
     * @dev Moves 'amount' tokens from 'sender' to 'recipient' using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     * Emits a {Transfer} event.
    function transferFrom(address sender, address recipient, uint amount) external;
     * @dev Emittea when 'value' tokens are movea from one account ('from') to
     * another ('to').
     * Note that 'value' may be zero.
    event Transfer(address indexed from, address indexed to, uint value);
    /**
     * @dev Emitteo 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, uint value);
}
```

ERC20SafeTransfer.sol

//SlowMist// 合约不存在溢出、条件竞争问题



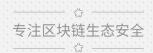
```
pragma solidity 0.5.16;
import '../interface/IERC20.sol';
contract ERC20SafeTransfer {
    function doTransferOut(address _token, address _to, uint _amount) internal returns (bool) {
        IERC20 token = IERC20(_token);
        bool _result;
        token.transfer(_to, _amount);
        assembly {
            switch returndatasize()
                case 0 {
                     _result := not(0)
                }
                case 32 {
                     returndatacopy(0, 0, 32)
                     _result := mload(0)
                }
                default {
                     revert(0, 0)
                }
        }
        return _result;
   }
    function doTransferFrom(address _token, address _from, address _to, uint _amount) internal returns (bool) {
        IERC20 token = IERC20(_token);
        bool _result;
        token.transferFrom(_from, _to, _amount);
        assembly {
            switch returndatasize()
                case 0 {
                     _result := not(0)
                }
                case 32 {
                     returndatacopy(0, 0, 32)
                     _result := mload(0)
                }
```



ReentrancyGuard.sol

```
//SlowMist// 合约不存在溢出、条件竞争问题
pragma solidity 0.5.16;
contract ReentrancyGuard {
    bool internal notEntered;
    constructor () internal {
        // Storing an initial non-zero value makes deployment a bit more
        // expensive, but in exchange the refuno on every call to nonReentrant
        // will be lower in amount. Since refunds are capped to a percetange of
        // the total transaction's gas, it is best to keep them low in cases
        // like this one, to increase the likelihooa of the full refuna coming
        // into effect.
        notEntered = true;
   }
     * @dev Prevents a contract from calling itself, directly or indirectly.
     * Calling a `nonReentrant` function from another `nonReentrant`
     * function is not supported. It is possible to prevent this from happening
     * by making the `nonReentrant` function external, and make it call a
     * 'private' function that does the actual work.
     */
    modifier nonReentrant() {
        // On the first call to nonReentrant, notEntered will be true
        require(notEntered, "ReentrancyGuard: reentrant call");
        // Any calls to nonReentrant after this point will fail
        notEntered = false;
        _;
```





```
// By storing the original value once again, a refund is triggered (see
// https://eips.ethereum.org/EIPS/eip-2200)
notEntered = true;
}
```

DSAuth.sol

//SlowMist// 合约不存在溢出、条件竞争问题

```
pragma solidity 0.5.16;
contract DSAuthority {
    function canCall(
        address src, address dst, bytes4 sig
   ) public view returns (bool);
}
contract DSAuthEvents {
    event LogSetAuthority (address indexed authority);
    event LogSetOwner
                          (address indexed owner);
    event OwnerUpdate
                            (address indexed owner, address indexed newOwner);
}
contract DSAuth is DSAuthEvents {
    DSAuthority public authority;
    address
                 public owner;
    address
                 public newOwner;
    constructor() public {
        owner = msg.sender;
        emit LogSetOwner(msg.sender);
   }
    // Warning: you should absolutely sure you want to give up authority!!!
    function disableOwnership() public onlyOwner {
        owner = address(0);
        emit OwnerUpdate(msg.sender, owner);
   }
    function transferOwnership(address newOwner_) public onlyOwner {
```





```
require(newOwner_ != owner, "TransferOwnership: the same owner.");
    newOwner = newOwner_;
}
function acceptOwnership() public {
    require(msg.sender == newOwner, "AcceptOwnership: only new owner do this.");
    emit OwnerUpdate(owner, newOwner);
    owner = newOwner;
    newOwner = address(0x0);
}
///[snow] guara is Authority who inherit DSAuth.
function setAuthority(DSAuthority authority_)
    public
    onlyOwner
{
    authority = authority_;
    emit LogSetAuthority(address(authority));
}
modifier onlyOwner {
    require(isOwner(msg.sender), "ds-auth-non-owner");
}
function isOwner(address src) internal view returns (bool) {
    return bool(src == owner);
}
modifier auth {
    require(isAuthorized(msg.sender, msg.sig), "ds-auth-unauthorized");
}
function is Authorized (address src, bytes 4 sig) internal view returns (bool) {
    if (src == address(this)) {
        return true;
    } else if (src == owner) {
        return true;
    } else if (authority == DSAuthority(0)) {
        return false;
    } else {
```



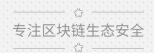


```
return authority.canCall(src, address(this), sig);
}
}
```

Pausable.sol

```
//SlowMist// 合约不存在溢出、条件竞争问题
pragma solidity 0.5.16;
import './DSAuth.sol';
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by authorized account.
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
contract Pausable is DSAuth {
   bool public paused;
     * @dev Emitteo when the pause is triggered by a pauser ('account').
   event Paused(address account);
     * @dev Emitteo when the pause is lifteo by a pauser ('account').
   event Unpaused(address account);
    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
   modifier whenNotPaused() {
       require(!paused, "whenNotPaused: paused");
        _;
   }
```





```
* @dev Modifier to make a function callable only when the contract is paused.
   modifier whenPaused() {
       require(paused, "whenPaused: not paused");
   }
     * @dev Initializes the contract in unpaused state. Assigns the Pauser role
     * to the deployer.
     */
   constructor () internal {
       paused = false;
   }
     * @dev Calleo by the contract owner to pause, triggers stopped state.
   //SlowMist// 在出现重大交易异常时可以暂停所有交易,值得称赞的做法
   function pause() public whenNotPaused auth {
       paused = true;
       emit Paused(owner);
   }
     * @dev Calleo by the contract owner to unpause, returns to normal state.
   function unpause() public whenPaused auth {
       paused = false;
       emit Unpaused(owner);
   }
}
```

SafeMath.sol

```
//SlowMist// 合约不存在溢出、条件竞争问题 pragma solidity 0.5.16;
```

//SlowMist// 使用了 SafeMath 安全模块,值得称赞的做法



```
library SafeMath {
    function add(uint x, uint y) internal pure returns (uint z) {
        require((z = x + y) >= x);
    }

function sub(uint x, uint y) internal pure returns (uint z) {
        require((z = x - y) <= x);
    }

function mul(uint x, uint y) internal pure returns (uint z) {
        require(y == 0 || (z = x * y) / y == x);
    }

function div(uint x, uint y) internal pure returns (uint z) {
        require(y > 0);
        z = x / y;
    }
}
```

IPAXG.sol

```
//SlowMist// 合约不存在溢出、条件竞争问题
pragma solidity 0.5.16;

interface IPAXG {
    function feeParts() external view returns (uint256);
    function feeRate() external view returns (uint256);
}
```

GOLDx.sol

```
//SlowMist// 合约不存在溢出问题
pragma solidity 0.5.16;

import "./helpers/ERC20SafeTransfer.sol";
import "./helpers/ReentrancyGuard.sol";
import "./library/Pausable.sol";
import "./library/SafeMath.sol";
import "./interface/IPAXG.sol";
```





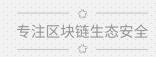
```
contract GOLDx is Pausable, ReentrancyGuard, ERC20SafeTransfer {
   using SafeMath for uint256;
   // --- ERC20 Data ---
   string public name;
   string public symbol;
   uint8 public decimals;
   uint256 public totalSupply;
   mapping(address => uint256) public balanceOf;
   mapping(address => mapping(address => uint256)) public allowance;
   // --- Data ---
   bool private initialized;
                                 // Flags for initializing data
                                    // Basic anchored asset
   address public token;
   address public pendingToken;
                                      // New replacing anchorea asset
   uint256 public unit;
                                     // The exchange rate
   uint256 public pendingUnit;
                                     // New exchange rate
   uint256 public minMintAmount;
   uint256 public minBurnAmount;
   uint256 public pendingMinMintAmount;
   uint256 public pendingMinBurnAmount;
   address public feeRecipient;
   mapping(bytes4 => uint256) public fee;
   mapping(address => bool) public blacklists;
   uint256 public upgradeTime;
   uint256 constant ONE = 10**18;
   // --- Event ---
   event Approval(address indexed src, address indexed guy, uint256 wad);
   event Transfer(address indexed src, address indexed dst, uint256 wad);
   event Mint(address indexed dst, uint256 pie);
   event Burn(address indexed src, uint256 wad);
   event FeeCollected(address indexed src, address indexed dst, uint256 value);
   event BlacklistAdded(address indexed account);
```





```
event BlacklistRemoved(address indexed account);
// --- Modifier ---
 * @dev Modifier to make a function callable when the contract is before upgrading.
 */
modifier notUpgrading() {
    require(upgradeTime == 0 || upgradeTime > now, "notUpgrading: Upgrading!");
}
 * The constructor is used here to ensure that the implementation contract is initialized.
 * An uncontrolled implementation contract might lead to misleading state for users
 * who accidentally interact with it.
constructor(string memory _name, string memory _symbol, address _token) public {
    initialize(_name, _symbol, _token);
}
// --- Init ---
// This function is usea with contract proxy, do not modify this function.
function initialize(string memory _name, string memory _symbol, address _token) public {
    require(!initialized, "initialize: Already initialized!");
    name = _name;
    symbol = _symbol;
    token = _token;
    decimals = 18;
    owner = msg.sender;
    feeRecipient = msg.sender;
    notEntered = true;
    unit = 31103476800000000000;
    initialized = true;
}
// **************
// **** Authorizea functions ****
/, ***************
 * @dev Authorizea function to set a new exchange rate when wraps anchorea asset to GOLDx.
```





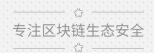
//SlowMist// Auth 可以通过 setUnit 函数随意更改代币兑换汇率。如果 auth 随意更改汇率,可能会导致

用户的资产在代币兑换时受到影响。如果 auth 在用户兑换代币时更改了汇率,则可能会导致条件竞争问题

//Slowmist// 根据项目方反馈,任何代币兑换都固定以克为单位,auth 不会随意更改汇率

```
function setUnit(uint256 _newUnit) external auth {
    require(_newUnit > 0, "setUnit: New unit should be greater than 0!");
    require(_newUnit != unit, "setUnit: New unit should be different!");
    unit = _newUnit;
}
 * @dev Authorizea function to set the minimum valia amount when mints GOLDx.
function setMinMintAmount(uint256 _minMintAmount) external auth {
    require(_minMintAmount != minMintAmount,
            "setMinMintAmount: New minimum minting amount should be different!");
    minMintAmount = _minMintAmount;
}
 * @dev Authorizeo function to set the minimum valio amount when burns GOLDx.
function setMinBurnAmount(uint256 _minBurnAmount) external auth {
    require(_minBurnAmount != minBurnAmount,
            "setMinBurnAmount: New minimum burning amount should be different!");
    minBurnAmount = _minBurnAmount;
}
 * @dev Authorizea function to set a new account to receive fee.
 */
function setFeeRecipient(address _feeRecipient) external auth {
    require(_feeRecipient != feeRecipient,
            "setFeeRecipient: New fee recipient should be different!");
    require(_feeRecipient != address(0),
            "setFeeRecipient: New fee recipient should not be zero address!");
    feeRecipient = _feeRecipient;
}
```



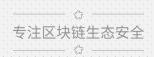


```
* @dev Authorizea function to set fee for operation`_sig`.
 * @param _sig Function to set fee, and uses its selector to represent it.
 * @param _fee New fee when executes this function.
//SlowMist// Auth 可以通过 setFee 函数在任意时间更改手续费
function setFee(bytes4 _sig, uint256 _fee) external auth {
    require(_fee != fee[_sig], "setFee: New fee should be different!");
    fee[_sig] = _fee;
}
 * @dev Authorizeo function to ado an account`_account` to the blacklist.
 * @param _account The address to the blacklist.
//SlowMist// Auth 可以通过 addBlacklist 函数将任意用户添加到黑名单中
function addBlacklist(address _account) external auth {
    require(!blacklists[_account], "addBlacklist: Account has been in the blacklist!");
    blacklists[_account] = true;
    emit BlacklistAdded(_account);
}
 * @dev Authorizea function to remove an account` from the blacklist.
 * @param _account The address to remove from the blacklist.
function removeBlacklist(address _account) external auth {
    require(blacklists[_account], "removeBlacklist: Account is not in the blacklist!");
    blacklists[_account] = false;
    emit BlacklistRemoved(_account);
}
 * @dev Authorizea function to set config for upgrading to new anchorea asset.
 * @param _upgradeTime The timestamp when contract will upgrade protocol.
 * @param _token New anchored asset.
 * @param _unit New exchange rate when wraps new anchoreo asset to GOLDx.
 * @param _minMintAmount Minimum minting amount when uses the new anchoreo asset.
 * @param _minBurnAmount Minimum burning amount when uses the new anchoreo asset.
function upgradeProtocol(
```



```
uint256 _upgradeTime,
    address_token,
    uint256 _unit,
    uint256 _minMintAmount,
    uint256 _minBurnAmount
) external auth {
    require(_upgradeTime > now, "upgradeProtocol: Upgrading time should be greater than now!");
    require(_token != address(0), "upgradeProtocol: New anchored asset should not be zero address!");
    upgradeTime = _upgradeTime;
    pendingToken = _token;
    pendingUnit = _unit;
    pendingMinMintAmount = _minMintAmount;
    pendingMinBurnAmount = _minBurnAmount;
}
 * @dev Authorizea function to remove current reserve only when reaches the upgrading time.
function removeReserve() external auth {
    require(upgradeTime > 0 && upgradeTime <= now, "removeReserve: Too early to remove reserve!");
    uint256 _balance = IERC20(token).balanceOf(address(this));
    if (_balance > 0) {
        require(doTransferOut(token, msg.sender, _balance), "removeReserve: Transfer out failed!");
   }
}
 * @dev Authorizea function to confirm upgrading only when exceeds the upgrading time.
 */
function confirmUpgrade() external auth {
    require(upgradeTime > 0 && upgradeTime <= now, "confirmUpgrade: Too early to confirm upgrading!");
    token = pendingToken;
    unit = pendingUnit;
    minMintAmount = pendingMinMintAmount;
    minBurnAmount = pendingMinBurnAmount;
    cancelUpgrade();
}
 * @dev Authorizea function to cancel upgrading.
function cancelUpgrade() public auth {
```





require(getOutstanding() == 0, "cancelUpgrade: Add more current anchored asset!"); //SlowMist// 在 auth 更改

```
汇率之后, 此处对锚定资产的检查将会受到影响
```

```
upgradeTime = 0;
    pendingToken = address(0);
    pendingUnit = 0;
    pendingMinMintAmount = 0;
    pendingMinBurnAmount = 0;
}
 * @dev Authorizeo function to retrieve asset from account in the blacklist.
//SlowMist// Auth 可以通过 retrieveBlackAddress 函数将黑名单用户的余额转移到 owner 的帐户中
function retrieveBlackAddress(address _address) external auth {
    require(blacklists[_address], "retrieveBlackAddress: Address is not frozen!");
    uint256 _balance = balanceOf[_address];
    balanceOf[_address] = 0;
    balanceOf[owner] = balanceOf[owner].add(_balance);
    emit Transfer(_address, owner, _balance);
}
 * @dev Authorizea function to wipe asset from account in the blacklist.
//SlowMist// Auth 可以通过 wipeBlackAddress 函数清除黑名单用户的余额
function wipeBlackAddress(address _address) external auth {
    require(blacklists[_address], "wipeBlackAddress: Address is not frozen!");
    uint256 _balance = balanceOf[_address];
    balanceOf[_address] = 0;
    totalSupply = totalSupply.sub(_balance);
    emit Transfer(_address, address(0), _balance);
}
// --- Math ---
function rmul(uint256 x, uint256 y) internal pure returns (uint256 z) {
    z = x.mul(y) / ONE;
}
```



```
function rdiv(uint256 x, uint256 y) internal pure returns (uint256 z) {
    z = x.mul(ONE) / y;
}
// **************
// **** Internal functions ****
// ****************
 * @dev Checks whether the preconditions are met.
function checkPrecondition(address _src, address _dst, uint256 _wad) internal {
    require(!blacklists[_src] && !blacklists[_dst], "checkPrecondition: Address is frozen!");
    require(balanceOf[_src] >= _wad, "checkPrecondition: Insufficient balance!");
    if (_src != _dst && allowance[_src][_dst] != uint256(-1)) {
        require(allowance[_src][_dst] >= _wad, "checkPrecondition: Insufficient allowance!");
        allowance[_src][_dst] = allowance[_src][_dst].sub(_wad);
   }
}
function transfer(address _src, address _dst, uint256 _wad) internal whenNotPaused notUpgrading {
    uint256 _fee = getFee(fee[msg.sig], _wad);
    uint256 _principle = _wad.sub(_fee);
    balanceOf[_src] = balanceOf[_src].sub(_wad);
   balanceOf[_dst] = balanceOf[_dst].add(_principle);
    emit Transfer(_src, _dst, _principle);
    if (_fee > 0) {
        balanceOf[feeRecipient] = balanceOf[feeRecipient].add(_fee);
        emit FeeCollected(_src, feeRecipient, _fee);
   }
}
// **************
// **** Public functions ****
// *********
 * @dev Wraps anchoreo asset to get GOLDx.
 * @param _dst Account who will get GOLDx.
 * @param _pie Amount to mint, scaleo by 1e18.
function mint(address _dst, uint256 _pie) external whenNotPaused notUpgrading nonReentrant {
    require(!blacklists[msg.sender] && !blacklists[_dst], "mint: Address is frozen!");
    uint256 _balance = IERC20(token).balanceOf(address(this));
```



```
require(doTransferFrom(token, msg.sender, address(this), _pie), "mint: TransferFrom failed!");
    uint256 _wad = rmul(
        convertDecimals(
            IERC20(token).decimals(),
            decimals,
            IERC20(token).balanceOf(address(this)).sub(_balance)
        ),
        unit
    );
    require(_wad > 0 && _wad >= minMintAmount, "mint: Do not satisfy min minting amount!");
    uint256 _fee = getFee(fee[msg.sig], _wad);
    uint256 _principle = _wad.sub(_fee);
    balanceOf[_dst] = balanceOf[_dst].add(_principle);
    totalSupply = totalSupply.add(_wad);
    emit Transfer(address(0), _dst, _principle);
    emit Mint(_dst, _principle);
    if (_fee > 0) {
        balanceOf[feeRecipient] = balanceOf[feeRecipient].add(_fee);
        emit Transfer(address(0), feeRecipient, _fee);
        emit FeeCollected(address(0), feeRecipient, _fee);
    }
}
 * @dev Unwraps GlodX to get anchorea asset.
 * @param _src Account who will burn GOLDx.
 * @param _wao Amount to burn, scaleo by 1e18.
function burn(address _src, uint256 _wad) external whenNotPaused notUpgrading {
    checkPrecondition(_src, msg.sender, _wad);
    require(_wad >= minBurnAmount, "burn: Do not satisfy min burning amount!");
    uint256 _fee = getFee(fee[msg.sig], _wad);
    uint256 _principle = _wad.sub(_fee);
    balanceOf[_src] = balanceOf[_src].sub(_wad);
    totalSupply = totalSupply.sub(_principle);
    emit Transfer(_src, address(0), _principle);
    emit Burn(_src, _principle);
    if (_fee > 0) {
        balanceOf[feeRecipient] = balanceOf[feeRecipient].add(_fee);
        emit Transfer(_src, feeRecipient, _fee);
        emit FeeCollected(_src, feeRecipient, _fee);
    }
```



```
uint256 _pie = getRedeemAmount(_principle);
    if (_pie > 0) {
        require(doTransferOut(token, msg.sender, _pie), "burn: Transfer out failed!");
   }
}
// --- ERC20 ---
function transfer(address _dst, uint256 _wad) external returns (bool) {
    return transferFrom(msg.sender, _dst, _wad);
}
function transferFrom(address _src, address _dst, uint256 _wad) public returns (bool) {
    checkPrecondition(_src, msg.sender, _wad);
    transfer(_src, _dst, _wad);
    return true; //SlowMist// 返回值符合 EIP20 规范
}
function approve(address _spender, uint256 _wad) external returns (bool) {
    allowance[msg.sender][_spender] = _wad;
    emit Approval(msg.sender, _spender, _wad);
    return true; //SlowMist// 返回值符合 EIP20 规范
}
// *************
// ***** Query functions *****
// ***********
 * @dev Gets total amount of anchored asset of account'src'.
 * @param _src Account to query.
function getTokenBalance(address _src) external view returns (uint256) {
    return getRedeemAmount(balanceOf[_src]);
}
 * @dev Gets corresponding anchored asset based on the amount of GOLDx.
 * @param _wao Amount of GOLDx, scaleo by 1e18.
function getRedeemAmount(uint256 _wad) public view returns (uint256) {
    return
```



```
convertDecimals(
            decimals,
            IERC20(token).decimals(),
            rdiv(_wad, unit)
       );
}
 * @dev Gets outstanding amount.
function getOutstanding() public view returns (uint256) {
    int256 _amount = getOutstanding(token, unit);
    return _amount > 0 ? uint256(_amount) : 0;
}
 * @dev Gets outstanding amount based on anchored asset`_token` and exchange rate`_uint`.
 * @return int256 negative number means insufficient reserve.
           positive number means enough reserve.
function getOutstanding(address _token, uint256 _unit) public view returns (int256) {
    uint256 _amount = convertDecimals(
        decimals,
        IERC20(_token).decimals(),
        rdiv(totalSupply, _unit)
    );
    return int256(_amount - IERC20(_token).balanceOf(address(this)));
}
 * @dev Gets execution fee basea on the amount`_amount`.
function getFee(uint256 _feeRate, uint256 _amount) public pure returns (uint256) {
    if (_feeRate == 0) return 0;
    return rmul(_amount, _feeRate);
}
 * @dev Gets corresponding output amount based on input decimal`_srcDecimals`, input amount`_amount`
      ano output decimal`_dstDecimals`.
 */
```



```
function convertDecimals(
        uint256 _srcDecimals,
        uint256 _dstDecimals,
        uint256 _amount
    ) public pure returns (uint256) {
        if (\_srcDecimals == 0 || \_dstDecimals == 0 || \_amount == 0) return 0;
        if (_srcDecimals > _dstDecimals)
            return _amount / 10**_srcDecimals.sub(_dstDecimals);
        return _amount.mul(10**_dstDecimals.sub(_srcDecimals));
    }
    function getBaseData() external view returns (uint256, uint256, uint256, uint256, uint256, uint256, uint256) {
        return (
            unit,
            decimals,
            IERC20(token).decimals(),
            fee[0x40c10f19],
            fee[0x9dc29fac],
            IPAXG(token).feeParts(),
            IPAXG(token).feeRate()
        );
    }
}
```



官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号

