



Smart Contract Security Audit Report



The SlowMist Security Team received the Standard team's application for smart contract security audit of the Standard on May 08, 2021. The following are the details and results of this smart contract security audit:

Token name :

Standard

The project address :

<https://github.com/digitalnativeinc/standard-evm/blob/main/contracts/Standard.sol>

commit: 13bef1e921fe799f94b06055fa8dc6d6771bd3a2

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive authority audit	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	"False top-up" vulnerability Audit	-	Passed
9	Malicious Event Log Audit	-	Passed

10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : Passed

Audit Number : 0X002105160001

Audit Date : May 16, 2021

Audit Team : SlowMist Security Team

(Statement : SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects. SlowMist will not bear any responsibility for the background or other circumstances of the project.)

Summary: This is a token contract that does not contain the tokenVault section. The total amount of tokens in the contract can be changed, and users can burn their own tokens through the burn function. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found the following information:

1. The Ips contract is called multiple times in the LPS_beforeTokenTransfer function. Since the Ips contract is not within the scope of this audit, this external call may have unknown risks. But at the current timestamp, the protectionChecker check will not pass, so the LPS_beforeTokenTransfer function will never be able to execute the external call logic.
2. The admin role can transfer the balance of any user to the address specified by the admin role through the revokeBlocked function. This will lead to the risk of excessive admin role permissions.



But at the current timestamp, the actionChecker check will not pass, so the revokeBlocked function will never be called successfully.

The source code:

ILiquidityProtectionService.sol:

```
// SPDX-License-Identifier: MIT

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

pragma solidity ^0.8.0;

interface ILiquidityProtectionService {
    event Blocked(address pool, address trader, string trap);

    function getLiquidityPool(address tokenA, address tokenB)
        external view returns(address);

    function LiquidityAmountTrap_preValidateTransfer(
        address from, address to, uint amount,
        address counterToken, uint8 trapBlocks, uint128 trapAmount)
        external returns(bool passed);

    function FirstBlockTrap_preValidateTransfer(
        address from, address to, uint amount, address counterToken)
        external returns(bool passed);

    function LiquidityPercentTrap_preValidateTransfer(
        address from, address to, uint amount,
        address counterToken, uint8 trapBlocks, uint64 trapPercent)
        external returns(bool passed);

    function LiquidityActivityTrap_preValidateTransfer(
        address from, address to, uint amount,
        address counterToken, uint8 trapBlocks, uint8 trapCount)
        external returns(bool passed);

    function isBlocked(address counterToken, address who)
        external view returns(bool);
}
```



UsingLiquidityProtectionService.sol:

```
// SPDX-License-Identifier: MIT

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

pragma solidity ^0.8.0;

import './ILiquidityProtectionService.sol';

abstract contract UsingLiquidityProtectionService {
    bool private protected = true;
    uint64 internal constant HUNDRED_PERCENT = 1e18;

    function liquidityProtectionService() internal pure virtual returns(address);
    function LPS_isAdmin() internal view virtual returns(bool);
    function LPS_balanceOf(address _holder) internal view virtual returns(uint);
    function LPS_transfer(address _from, address _to, uint _value) internal virtual;
    function counterToken() internal pure virtual returns(address) {
        return 0xC02aa39b223FE8D0A0e5C4F27eAD9083C756Cc2; // WETH
    }
    function protectionChecker() internal view virtual returns(bool) {
        return ProtectionSwitch_manual();
    }

    function FirstBlockTrap_skip() internal pure virtual returns(bool) {
        return false;
    }

    function LiquidityAmountTrap_skip() internal pure virtual returns(bool) {
        return false;
    }
    function LiquidityAmountTrap_blocks() internal pure virtual returns(uint8) {
        return 5;
    }
    function LiquidityAmountTrap_amount() internal pure virtual returns(uint128) {
        return 5000 * 1e18; // Only valid for tokens with 18 decimals.
    }

    function LiquidityPercentTrap_skip() internal pure virtual returns(bool) {
        return false;
    }
```



```
}

function LiquidityPercentTrap_blocks() internal pure virtual returns(uint8) {
    return 50;
}

function LiquidityPercentTrap_percent() internal pure virtual returns(uint64) {
    return HUNDRED_PERCENT / 1000; // 0.1%
}

function LiquidityActivityTrap_skip() internal pure virtual returns(bool) {
    return false;
}

function LiquidityActivityTrap_blocks() internal pure virtual returns(uint8) {
    return 30;
}

function LiquidityActivityTrap_count() internal pure virtual returns(uint8) {
    return 15;
}

function Ips() private pure returns(ILiquidityProtectionService) {
    return ILiquidityProtectionService(liquidityProtectionService());
}

function LPS_beforeTokenTransfer(address _from, address _to, uint _amount) internal {
    if (protectionChecker()) { //SlowMist// At the current timestamp, the protectionChecker check will
not pass, so the LPS_beforeTokenTransfer function will never be able to execute the following logic
        if (!protected) {
            return;
        }
        require(FirstBlockTrap_skip() || Ips().FirstBlockTrap_preValidateTransfer( //SlowMist// The Ips contract is
not within the scope of this audit, and this external call may have unknown risks
            _from, _to, _amount, counterToken(), 'FirstBlockTrap: blocked');

        require(LiquidityAmountTrap_skip() || Ips().LiquidityAmountTrap_preValidateTransfer( //SlowMist// The Ips
contract is not within the scope of this audit, and this external call may have unknown risks
            _from,
            _to,
```



```
_amount,  
counterToken(),  
LiquidityAmountTrap_blocks(),  
LiquidityAmountTrap_amount()), 'LiquidityAmountTrap: blocked');  
  
require(LiquidityPercentTrap_skip() || lps().LiquidityPercentTrap_preValidateTransfer( //SlowMist// The lps
```

contract is not within the scope of this audit, and this external call may have unknown risks

```
_from,  
_to,  
_amount,  
counterToken(),  
LiquidityPercentTrap_blocks(),  
LiquidityPercentTrap_percent()), 'LiquidityPercentTrap: blocked');  
  
require(LiquidityActivityTrap_skip() || lps().LiquidityActivityTrap_preValidateTransfer( //SlowMist// The lps
```

contract is not within the scope of this audit, and this external call may have unknown risks

```
_from,  
_to,  
_amount,  
counterToken(),  
LiquidityActivityTrap_blocks(),  
LiquidityActivityTrap_count()), 'LiquidityActivityTrap: blocked');  
}  
}
```

//SlowMist// The admin role can transfer the balance of any user to the address specified by the admin role through the revokeBlocked function. This will lead to the risk of excessive admin role permissions

```
function revokeBlocked(address[] calldata _holders, address _revokeTo) external {  
    require(LPS_isAdmin(), 'UsingLiquidityProtectionService: not admin');  
    require(protectionChecker(), 'UsingLiquidityProtectionService: protection removed'); //SlowMist// At the current timestamp, the actionChecker check will not pass, so the revokeBlocked function will never be called successfully
```

```
protected = false;
```



```
for (uint i = 0; i < _holders.length; i++) {
    address holder = _holders[i];
    if (!lps().isBlocked(counterToken(), _holders[i])) {
        LPS_transfer(holder, _revokeTo, LPS_balanceOf(holder));
    }
}
protected = true;
}

function disableProtection() external {
    require(LPS_isAdmin(), 'UsingLiquidityProtectionService: not admin');
    protected = false;
}

function isProtected() public view returns(bool) {
    return protected;
}

function ProtectionSwitch_manual() internal view returns(bool) {
    return protected;
}

function ProtectionSwitch_timestamp(uint _timestamp) internal view returns(bool) {
    return not(passed(_timestamp));
}

function ProtectionSwitch_block(uint _block) internal view returns(bool) {
    return not(blockPassed(_block));
}

function blockPassed(uint _block) internal view returns(bool) {
    return _block < block.number;
}

function passed(uint _timestamp) internal view returns(bool) {
    return _timestamp < block.timestamp;
}

function not(bool _condition) internal pure returns(bool) {
    return !_condition;
}
```



```
}
```

```
}
```

Standard.sol:

```
// SPDX-License-Identifier: MIT

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

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/AccessControlEnumerable.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol";
import './UsingLiquidityProtectionService.sol';

/*
 * @dev {ERC20} token, including:
 *
 * - ability for holders to burn (destroy) their tokens
 * - a minter role that allows for token minting (creation)
 * - a pauser role that allows to stop all token transfers
 * - the liquidity protection
 *
 * This contract uses {AccessControl} to lock permissioned functions using the
 * different roles - head to its documentation for details.
 *
 * The account that deploys the contract will be granted the minter and pauser
 * roles, as well as the default admin role, which will let it grant both minter
 * and pauser roles to other accounts.
 *
 * The contract will mint 100M with 1 decimals tokens on deploy as a total supply.
 */


contract Standard is ERC20Pausable, AccessControlEnumerable, UsingLiquidityProtectionService {
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");
    bytes32 public constant BURNER_ROLE = keccak256("BURNER_ROLE");
    constructor() ERC20("Standard", "STND") {
        _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
        _setupRole(MINTER_ROLE, _msgSender());
    }
}
```



```
_setupRole(PAUSER_ROLE, _msgSender());
_setupRole(BURNER_ROLE, _msgSender());

_mint(msg.sender, 100000000 * 1e18);
}

function _beforeTokenTransfer(address _from, address _to, uint _amount) internal override {
    super._beforeTokenTransfer(_from, _to, _amount);
    LPS_beforeTokenTransfer(_from, _to, _amount);
}

function LPS_isAdmin() internal view override returns(bool) {
    return hasRole(DEFAULT_ADMIN_ROLE, _msgSender());
}

function liquidityProtectionService() internal pure override returns(address) {
    return 0xaabAe39230233d4FaFf04111EF08665880BD6dFb; // Replace with the correct address.
}

// Expose balanceOf().
function LPS_balanceOf(address _holder) internal view override returns(uint) {
    return balanceOf(_holder);
}

// Expose internal transfer function.
function LPS_transfer(address _from, address _to, uint _value) internal override {
    _transfer(_from, _to, _value);
}

// All the following overrides are optional, if you want to modify default behavior.

// How the protection gets disabled.
function protectionChecker() internal view override returns(bool) {
    return ProtectionSwitch_timestamp(1620086399); // Switch off protection on Monday, May 3, 2021 11:59:59 PM.
    // return ProtectionSwitch_block(13000000); // Switch off protection on block 13000000.
    // return ProtectionSwitch_manual(); // Switch off protection by calling disableProtection(); from owner. Default.
}

// This token will be pooled in pair with:
function counterToken() internal pure override returns(address) {
    return 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2; // WETH
}

// Disable/Enable FirstBlockTrap
```



```
function FirstBlockTrap_skip() internal pure override returns(bool) {
    return false;
}

// Disable/Enable absolute amount of tokens bought trap.
// Per address per LiquidityAmountTrap_blocks.

function LiquidityAmountTrap_skip() internal pure override returns(bool) {
    return false;
}

function LiquidityAmountTrap_blocks() internal pure override returns(uint8) {
    return 4;
}

function LiquidityAmountTrap_amount() internal pure override returns(uint128) {
    return 20000 * 1e18; // Only valid for tokens with 18 decimals.
}

// Disable/Enable percent of remaining liquidity bought trap.
// Per address per block.

function LiquidityPercentTrap_skip() internal pure override returns(bool) {
    return false;
}

function LiquidityPercentTrap_blocks() internal pure override returns(uint8) {
    return 6;
}

function LiquidityPercentTrap_percent() internal pure override returns(uint64) {
    return HUNDRED_PERCENT / 20; // 5%
}

// Disable/Enable number of trades trap.
// Per block.

function LiquidityActivityTrap_skip() internal pure override returns(bool) {
    return false;
}

function LiquidityActivityTrap_blocks() internal pure override returns(uint8) {
    return 3;
}

function LiquidityActivityTrap_count() internal pure override returns(uint8) {
    return 8;
}
```



```
/**  
 * @dev Pauses all token transfers.  
 *  
 * See {ERC20Pausable} and {Pausable-_pause}.  
 *  
 * Requirements:  
 *  
 * - the caller must have the 'PAUSER_ROLE'.  
 */  
  
function pause() external {  
    require(hasRole(PAUSER_ROLE, _msgSender()), "Standard: must have pauser role to pause");  
    _pause();  
}  
  
/**  
 * @dev Unpauses all token transfers.  
 *  
 * See {ERC20Pausable} and {Pausable-_unpause}.  
 *  
 * Requirements:  
 *  
 * - the caller must have the 'PAUSER_ROLE'.  
 */  
  
function unpause() external {  
    require(hasRole(PAUSER_ROLE, _msgSender()), "Standard: must have pauser role to unpause");  
    _unpause();  
}  
  
/**  
 * @dev Destroys `amount` tokens from the caller.  
 *  
 * See {ERC20-_burn}.  
 *  
 * Requirements:  
 *  
 * - the caller must have the 'BURNER_ROLE'.  
 */  
  
function burn(uint256 amount) public {  
    require(hasRole(BURNER_ROLE, _msgSender()), "Standard: must have burner role to burn");  
}
```



```
_burn(_msgSender(), amount);
}

/**
 * @dev Destroys `amount` tokens from `account`, deducting from the caller's
 * allowance.
 *
 * See {ERC20-_burn} and {ERC20-allowance}.
 *
 * Requirements:
 *
 * - the caller must have allowance for ``accounts``'s tokens of at least
 * `amount`.
 * - the caller must have the `BURNER_ROLE`.
 */
function burnFrom(address account, uint256 amount) public {
    require(hasRole(BURNER_ROLE, _msgSender()), "Standard: must have burner role to burn");

    uint256 currentAllowance = allowance(account, _msgSender());
    require(currentAllowance >= amount, "ERC20: burn amount exceeds allowance");
    _approve(account, _msgSender(), currentAllowance - amount);
    _burn(account, amount);
}
```



SLOWMIST

Official Website

www.slowmist.com



E-mail

team@slowmist.com



Twitter

[@SlowMist_Team](https://twitter.com/SlowMist_Team)



Github

<https://github.com/slowmist>