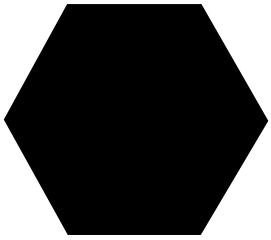
10 MAY 2019

BASIC SMART CONTRACT AUDIT REPORT

Hexlant.



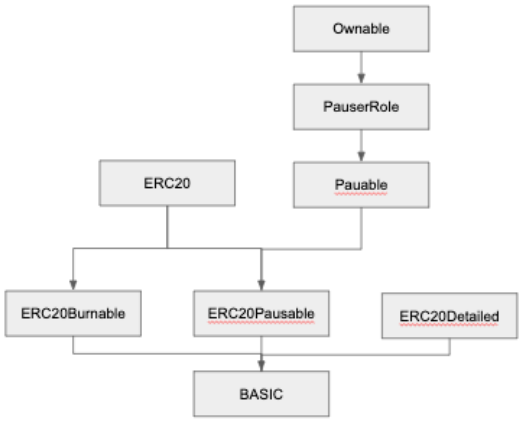
Analysis Purpose

본 리포트는 발행된 컨트랙트 코드가 요구사항을 충분히 만족하는지, 그리고 보안의 취약점과 실제 운영 하면서 발생 할 수 있는 문제들을 파악하고 해결방안을 찾기위해 분석을 수행하고 그 결과를 정리하였습니다. 이번 코드 분석은 다음과 같은 요소들을 검증하기위해 진행하였습니다.

- 구현된 기능의 정상작동 여부
- 기능 수행 중 보안 위험성
- **Off Chain**에서 발생하는 문제에 대한 대비
- 컨트랙트 코드의 가독성 및 코드 완성도

Function Summary

BASIC 컨트랙트는 **Open-Zeppelin**에서 제공하는 컨트랙트 코드로 작성되었으며, 다음과 같은 컨트랙트를 통해 **BASIC**의 기능을 구현하였습니다.



- **ERC20**
  ERC-20 표준 기능. 기본적인 토큰 전송을 비롯한 **ERC-20 Token**의 기본기능을 구현한 컨트랙트 입니다. **ERC20**인터페이스를 바탕으로, 토큰전송에 필요한 세부기능을 제공합니다.

- **Ownable**
  컨트랙트의 **Ownership**과 관련된 기능을 제공합니다. **onlyOwner modifier**을 사용하여 컨트랙트의 **Owner**일 경우 실행되도록 기능을 제공할 수 있습니다.

- **Pausable**
  컨트랙트의 정지와 관련된 기능을 제공합니다 **whenNotPaused, whenPaused modifer**을 사용하여 컨트랙트 상에서 토큰 전송 시 정지 상태를 체크할 수 있도록 기능을 제공합니다.

- **BASIC**
  BASIC Coin 고유의 토큰 생태계를 관리하는데 필요한 기능들을 제공합니다. 기본적으로 **ERC-20** 표준 기능에서 크게 벗어나지 않고, 개별 주소에 대한 계좌 동결 기능과 락업 기능을 추가적으로 제공합니다.

Variables

컨트랙트의 상태를 표현하는 변수들로, 컨트랙트에 필요한 정보들을 저장하고 있습니다

Variables	Description
_name	토큰의 이름
_symbol	토큰의 심볼
_decimal	최대 표현 가능한 소수점 자리수
_balance	Address의 잔액 테이블
_allowed	Address의 출금 권한 허용 테이블
_totalSupply	토큰 총 발행량
owner	컨트랙트 Owner
newOwner	컨트랙트의 새로운 Owner 후보
_pausers	Pause / Lock 기능을 수행할 수 있는 관리자 Address 리스트
_paused	Pause 상태
implementation	확장된 기능이 구현된 컨트랙트 주소
timelockList	Address 별 락업정보 테이블
frozenAccount	Address의 동결 여부

Modifier

함수의 한정요소로, 특정 기능을 수행할 때 한정된 조건에서만 호출 할 수 있도록 만들기 위해 사용합니다.
대표적으로 **onlyOwner** 가 있으며, 해당 **modifier**가 붙은 기능은 **Ownership**을 가진 **Address**만이 호출 가능합니다.

Modifier	Description
onlyOwner	컨트랙트 Owner만 실행가능
onlyNewOwner	새로운 컨트랙트 Owner 후보만 실행 가능
onlyPauser	Pause / Lock 관리자만 실행 가능
whenNotPaused	Pause 상태가 아닐 때만 기능을 허용
whenPaused	Pause 상태일 때만 기능을 허용
notFrozen	동결되지 않은 Address만 실행 가능

Function

컨트랙트의 함수들로, 실제 기능을 수행하는 코드들 입니다.
대표적으로 **transfer** 가 있으며, 해당 기능은 토큰을 전송하여 잔액을 변화시킵니다.

Function	Description
name	토큰의 이름 확인
symbol	토큰의 심볼 확인
decimals	토큰의 최대 표현 가능한 소수점 자리수 확인
transfer	Address에 토큰 전송
transferFrom	Address로부터 출금 권한을 부여받은 토큰을 전송
freezeAccont	Address의 토큰 전송을 동결
unfreezeAccount	Address의 동결 상태를 해제
lock	Address의 일정 토큰을 락업
transferWithLock	Address에 토큰 전송 후 락업
unlock	Address의 락업된 토큰 중 idx번째를 언락
upgradeTo	컨트랙트 기능 Upgrade
_lock	Address의 일정 토큰을 락업
_unlock	Address의 락업된 토큰 중 idx번째를 언락
_autoUnlock	Address의 락업된 토큰 중 만료기간이 지난 락업 해제
_setImplementation	업그레이드할 컨트랙트의 주소를 저장
isOwner	Address가 컨트랙트의 Owner인지 확인
transferOwnership	Address로 컨트랙트 Owner권한을 이전
acceptOwnership	컨트랙트의 New Owner를 승인
isPauser	Address가 Pauser인지 확인
addPauser	Address에 Pauser권한 부여
removePauser	Address의 Pauser권한 제거
renouncePauser	자신의 Pauser권한 포기
_addPauser	Address에 Pauser권한 부여
_removePauser	Address의 Pauser권한 제거
pause	컨트랙트를 토큰 전송 정지 상태로 변경
unpause	컨트랙트의 토큰 정지 상태를 해제
totalSupply	토큰의 총 유통량 확인
balanceOf	Address의 토큰 잔액 확인
allowance	Address에 부여된 출금 권한 토큰 잔액 확인
approve	Address에 일정 토큰의 출금 권한 부여
increaseAllowance	Address에 부여된 출금 권한 토큰 증액
decreaseAllowance	Address에 부여된 출금 권한 토큰 감액
_transfer	Address로 토큰 전송
_mint	토큰 추가 발행
_burn	토큰 소각
_burnFrom	출금 권한이 부여된 토큰 소각
burn	토큰 소각
burnFrom	출금 권한이 부여된 토큰 소각

Function Profile

Function Profile에서는 컨트랙트 함수들의 세부적인 내용들을 표시합니다. 함수의 세부적인 매개변수와, 다양한 옵션을 살펴보고, 콜스택을 통해 함수간의 호출관계를 정리합니다. 이를 통해 함수 호출관계를 파악하고, 함수들 간의 논리적 충돌이 있는지 쉽게 파악 할 수 있습니다.

Function Name	(BASIC) balanceOf		
Parameter	address		
Visibility	public	Modifier	-
Constant	view	Return	-
Callstack			
balanceOf └ super.balanceOf			

Function Name	(BASIC) transfer		
Parameter	address, uint256		
Visibility	public	Modifier	notFrozen
Constant	-	Return	bool
Callstack			
transfer └ _autoUnlock └ super.transfer			

Function Name	(BASIC) transferFrom		
Parameter	address, address, uint256		
Visibility	public	Modifier	notFrozen
Constant	-	Return	bool
Callstack			
transferFrom └_autoUnlock └super.transferFrom			

Function Name	(BASIC) freezeAccount		
Parameter	address		
Visibility	public	Modifier	onlyPauser
Constant	-	Return	bool
Callstack			
freezeAccount			

Function Name	(BASIC) unfreezeAccount		
Parameter	address		
Visibility	public	Modifier	onlyPauser
Constant	-	Return	bool
Callstack			
unfreezeAccount			

Function Name	(BASIC) lock		
Parameter	address, uint256, uint256		
Visibility	public	Modifier	onlyPauser
Constant	-	Return	bool
Callstack			
lock └_lock			

Function Name	(BASIC) transferWithLock		
Parameter	address, uint256, uint256		
Visibility	public	Modifier	onlyPauser
Constant	-	Return	bool
Callstack			
transferWithLock └_transfer └_lock			

Function Name	(BASIC) unlock		
Parameter	address, uint256		
Visibility	public	Modifier	onlyPauser
Constant	-	Return	bool
Callstack			
unlock └ _unlock			

Function Name	(BASIC) upgradeTo		
Parameter	address		
Visibility	public	Modifier	onlyOwner
Constant	-	Return	-
Callstack			
upgradeTo └_setImplementation			

Function Name	(BASIC) _lock		
Parameter	address, uint256, uint256		
Visibility	internal	Modifier	-
Constant	-	Return	bool
Callstack			
_lock			

Function Name	(BASIC) _unlock		
Parameter	address, uint256		
Visibility	internal	Modifier	-
Constant	-	Return	bool
Callstack			
_unlock			

Function Name	(BASIC) _autoUnlock		
Parameter	address		
Visibility	internal	Modifier	-
Constant	-	Return	bool
Callstack			
_autoUnlock └_unlock			

Function Name	(BASIC) _setImplementation		
Parameter	address		
Visibility	internal	Modifier	-
Constant	-	Return	-
Callstack			
_setImplementation			

Function Name	(Ownable) isOwner		
Parameter	address		
Visibility	public	Modifier	-
Constant	view	Return	bool
Callstack			
isOwner			

Function Name	(Ownable) transferOwnership		
Parameter	address		
Visibility	public	Modifier	onlyOwner
Constant	-	Return	-
Callstack			
transferOwnership			

Function Name	(Ownable) acceptOwnership		
Parameter	-		
Visibility	public	Modifier	onlyNewOwner
Constant	-	Return	bool
Callstack			
acceptOwnership			

Function Name	(PauserRole) isPauser		
Parameter	address		
Visibility	public	Modifier	-
Constant	view	Return	bool
Callstack			
isPauser			

Function Name	(PauserRole) addPauser		
Parameter	address		
Visibility	public	Modifier	onlyPauser
Constant	-	Return	-
Callstack			
addPauser			

Function Name	(PauserRole) removePauser		
Parameter	address		
Visibility	public	Modifier	onlyOwner
Constant	-	Return	-
Callstack			
removePauser └_removePauser			

Function Name	(PauserRole) _addPauser		
Parameter	address		
Visibility	internal	Modifier	-
Constant	-	Return	-
Callstack			
_addPauser			

Function Name	(PauserRole) _removePauser		
Parameter	address		
Visibility	internal	Modifier	-
Constant	-	Return	-
Callstack			
_removePauser			

Function Name	(PauserRole) pause		
Parameter	-		
Visibility	public	Modifier	onlyPauser whenNotPaused
Constant	-	Return	-
Callstack			
pause			

Function Name	(PauserRole) unpause		
Parameter	-		
Visibility	public	Modifier	onlyPauser whenNotPaused
Constant	-	Return	-
Callstack			
Unpause			

Function Name	(ERC20) totalSupply		
Parameter	-		
Visibility	public	Modifier	-
Constant	view	Return	uint256
Callstack			
totalSupply			

Function Name	(ERC20) balanceOf		
Parameter	address		
Visibility	public	Modifier	-
Constant	view	Return	uint256
Callstack			
balanceOf			

Function Name	(ERC20) allowance		
Parameter	address		
Visibility	public	Modifier	-
Constant	view	Return	uint256
Callstack			
Allowance			

Function Name	(ERC20) transfer		
Parameter	address, uint256		
Visibility	public	Modifier	-
Constant	-	Return	uint256
Callstack			
transfer └ _transfer			

Function Name	(ERC20) approve		
Parameter	address, uint256		
Visibility	public	Modifier	-
Constant	-	Return	bool
Callstack			
Approve			

Function Name	(ERC20) approve		
Parameter	address, uint256		
Visibility	public	Modifier	-
Constant	-	Return	bool
Callstack			
approve			

Function Name	(ERC20) transferFrom		
Parameter	address, address, uint256		
Visibility	public	Modifier	-
Constant	-	Return	bool
Callstack			
transferFrom └ _transfer			

Function Name	(ERC20) increaseAllowance		
Parameter	address, uint256		
Visibility	public	Modifier	-
Constant	-	Return	bool
Callstack			
decreaseAllowance			

Function Name	(ERC20) _transfer		
Parameter	address, address, uint256		
Visibility	internal	Modifier	-
Constant	-	Return	-
Callstack			
_transfer			

Function Name	(ERC20) _mint		
Parameter	address, uint256		
Visibility	internal	Modifier	-
Constant	-	Return	-
Callstack			
_mint			

Function Name	(ERC20) _burn		
Parameter	address, uint256		
Visibility	internal	Modifier	-
Constant	-	Return	-
Callstack			
_burn			

Function Name	(ERC20) _burnFrom		
Parameter	address, uint256		
Visibility	internal	Modifier	-
Constant	-	Return	-
Callstack			
_burnFrom			

Function Name	(ERC20Burnable) _burn		
Parameter	address, uint256		
Visibility	public	Modifier	-
Constant	-	Return	-
Callstack			
burn └_burn			

Function Name	(ERC20Burnable) _burnFrom		
Parameter	address, uint256		
Visibility	public	Modifier	-
Constant	-	Return	-
Callstack			
burnFrom └_burnFrom			

Test Result

Code Coverage

코드 커버리지는 작성한 테스트가 얼마만큼 컨트랙트 코드의 기능들을 테스트 했는지 알 수있는 정량적인 지표입니다.

BASIC Contract의 경우 **SafeMath** 컨트랙트 코드에서, **add, sub** 기능만 사용하기 때문에, **mul, div, mod** 함수에 대한 테스트를 진행할 수 없습니다. 아래의 지표는 **SafeMath**의 미사용 함수를 제외하고 테스트한 결과입니다.

File Name	Statements	Functions	Lines
BASIC.sol	100% (152/152)	100% (67/67)	100% (163/163)

Test cases

Test case	Result
컨트랙트 초기 설정 값이 일치하는가?	PASS
초기 설정 이름과 동일한가?	PASS
초기 설정 심볼과 동일한가?	PASS
초기 설정 최대 소수점 자리와 동일한가?	PASS
초기 설정 총 발행량과 동일한가	PASS
컨트랙트의 오너는 컨트랙트를 배포한 자인가?	PASS
토큰 전송 시 받는 주소가 0일 때 예외처리가 되고 있는가?	PASS
보내는 토큰의 양이 음수일 경우 예외처리가 되고 있는가?	PASS
보내는 토큰의 양이 보유한 토큰보다 많을 경우 전송이 거부되는가?	PASS
토큰 전송 시, Overflow, Underflow에 대한 예외처리가 이루어지는가?	PASS
approve / increaseAllowance/ decreaseAllowance 기능이 동작하는가?	PASS
보유한 토큰 이상의 출금 권한 부여가 가능한가?	PASS
transferFrom 기능이 동작하는가?	PASS
출금 권한이 부여된 토큰 중 일부를 전송하면 나머지는 그대로 존재하는가?	PASS
출금 권한이 부여된 이상의 토큰을 전송하면 예외처리가 되고 있는가?	PASS
출금 권한이 부여되지않은 토큰을 전송하면 예외처리가 되고 있는가?	PASS
보유한 토큰 이상을 소각하면 예외처리가 되고 있는가?	PASS
토큰을 소각하면 총 발행량도 감액하는가?	PASS
출금 권한을 허용받은 토큰을 소각할 수 있는가?	PASS
기본적인 컨트랙트 정지 기능은 동작하는가?	PASS
컨트랙트가 정지 상태일 때 토큰 전송을 시도하면 예외처리가 되고 있는가?	PASS
일반 유저가 컨트랙트의 정지상태를 변경 시도하면 예외처리가 되고 있는가?	PASS
컨트랙트가 정지 상태를 해제 가능한가?	PASS

Test case	Result
컨트랙트가 정지 상태일 때 출금 권한이 허용된 토큰을 전송 시 예외처리가 되고 있는가?	PASS
일반 유저가 타 유저의 계좌 동결을 시도하면 예외처리가 되고 있는가?	PASS
계좌가 동결된 유저가 토큰 전송을 시도하면 예외처리가 되고 있는가?	PASS
계좌가 동결된 유저가 출금 권한이 허용된 토큰을 전송 시 예외처리가 되고 있는가?	PASS
오너는 유저의 동결된 계좌를 해제할 수 있는가?	PASS
Ownership 변경기능이 동작하는가?	PASS
일반 유저가 Ownership 변경을 시도하면 예외처리가 되고 있는가?	PASS
Pauser권한을 가지고 있는 유저는 타 유저에게 Pauser권한을 부여할 수 있는가?	PASS
Pauser권한을 가지고 있는 유저는 자신의 권한을 포기할 수 있는가?	PASS
오너는 유저가 보유한 토큰을 락업시킬 수 있는가?	PASS
오너는 유저의 락업된 토큰을 해제할 수 있는가?	PASS
오너는 유저에게 토큰을 락업하여 전송할 수 있는가?	PASS
일반 유저가 타 유저의 토큰에 락업을 시도하면 예외처리가 되고 있는가?	PASS
일반 유저가 타 유저에게 토큰을 락업하여 전송 시도하면 예외처리가 되고 있는가?	PASS
토큰 전송 시 만료 기간이 지난 락업 토큰을 사용할 수 있는가?	PASS

Vulnerability Analysis

Critical Severity

심각성 치명적 단계는 일반적인 상황에서 보안 또는 큰 문제를 야기 할 수 있는 오류로 반드시 수정해야 하는 항목입니다

해당 항목 없음

High Severity

심각성 높음 단계는 일반적인 상황에서 발생하는 문제는 아니지만, 특수한 조건이나 예외상황에 의해서 문제가 발생 할 수 있는 항목입니다.
추가적인 예외처리나, 코너케이스에 대하여 분석하고 오류를 막을 수 있도록 수정이 필요한 항목입니다.

해당 항목 없음

Medium Severity

심각성 중간단계는 크게 문제가 되는 항목은 아니지만, 좀더 효율적으로 동작 할 수 있도록 수정을 권유하는 항목입니다

해당 항목 없음

Low Severity

낮은 위험도는 성능이나 보안에는 문제는 없지만, 코드 가독성, 컨트랙트 구조 개선을 위해 수정을 권유하는 항목입니다.

해당 항목 없음

Conclusion

BASIC 컨트랙트는 **Open-Zepelin**의 **ERC-20** 컨트랙트를 기반으로 작성되었습니다. 추가적으로 **Burnable, Pausable** 기능을 제공하여, 토큰소각 및 토큰전송의 정지기능을 제공합니다. 산술연산에 있어 **SafeMath**를 이용하기 때문에 **overflow/underflow**에 의한 오류를 발견할 수 없었습니다. **BASIC**은 전체 또는 개별 계좌에 대한 동결을 통해 해킹 및 피싱을 통해 탈취된 토큰의 유통을 통제할 수 있습니다. 일정기간동안의 락업기능을 통해서는 토큰의 초기 유통량을 통제 할 수 있으며, 이를 통해 덤핑을 방지 할 수 있습니다.
BASIC은 **Upgradable** 컨트랙트로, **Fallback**함수와 **Delegate Call**을 통해 확장된 기능을 제공할 수 있습니다. 이를 통해 서비스 확장에 필요한 기능을 추가 할 수 있습니다.

Declare

해당 리포트는 **Hexlant**의 스마트 컨트랙트 보안 감사 결과를 바탕으로 작성되었습니다. 해당 리포트는 비즈니스 모델의 적합성과 법적 규제, 투자에 대한 의견을 보증하지 않습니다. 리포트에 기술한 문제점 이외에 메인넷기술 또는 가상머신을 비롯하여 발견되지 않은 문제점이 있을 수 있습니다. 해당 리포트는 논의 목적으로만 사용됩니다.

```
pragma solidity ^0.5.0;

library SafeMath {
    /**
    * @dev Multiplies two unsigned integers, reverts on overflow.
    */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
    * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by zero.
    */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
        return c;
    }

    /**
    * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than
minuend).
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
    * @dev Adds two unsigned integers, reverts on overflow.
    */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }
   /**
    * @dev Divides two unsigned integers and returns the remainder (unsigned integer modulo),
    * reverts when dividing by zero.
    */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}
```

```
library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    /**
     * @dev give an account access to this role
     */
    function add(Role storage role, address account) internal {
        require(account != address(0));
        require(!has(role, account));

        role.bearer[account] = true;
    }

    /**
     * @dev remove an account's access to this role
     */
    function remove(Role storage role, address account) internal {
        require(account != address(0));
        require(has(role, account));

        role.bearer[account] = false;
    }

    /**
     * @dev check if an account has this role
     * @return bool
     */
    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0));
        return role.bearer[account];
    }
}
contract Ownable {
    address public owner;
    address public newOwner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    constructor() public {
        owner = msg.sender;
        newOwner = address(0);
    }

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }
    modifier onlyNewOwner() {
        require(msg.sender != address(0));
        require(msg.sender == newOwner);
        _;
    }

    function isOwner(address account) public view returns (bool) {
        if( account == owner ){
            return true;
        }
        else {
            return false;
        }
    }
```

```
    function transferOwnership(address _newOwner) public onlyOwner {
        require(_newOwner != address(0));
        newOwner = _newOwner;
    }

    function acceptOwnership() public onlyNewOwner returns(bool) {
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
        newOwner = address(0);
    }
}

contract PauserRole is Ownable{
    using Roles for Roles.Role;

    event PauserAdded(address indexed account);
    event PauserRemoved(address indexed account);

    Roles.Role private _pausers;

    constructor () internal {
        _addPauser(msg.sender);
    }

    modifier onlyPauser() {
        require(isPauser(msg.sender)|| isOwner(msg.sender));
        _;
    }
    function isPauser(address account) public view returns (bool) {
        return _pausers.has(account);
    }

    function addPauser(address account) public onlyPauser {
        _addPauser(account);
    }

    function removePauser(address account) public onlyOwner {
        _removePauser(account);
    }

    function renouncePauser() public {
        _removePauser(msg.sender);
    }

    function _addPauser(address account) internal {
        _pausers.add(account);
        emit PauserAdded(account);
    }

    function _removePauser(address account) internal {
        _pausers.remove(account);
        emit PauserRemoved(account);
    }
}
contract Pausable is PauserRole {
    event Paused(address account);
    event Unpaused(address account);

    bool private _paused;

    constructor () internal {
        _paused = false;
    }
```

```
    /**
     * @return true if the contract is paused, false otherwise.
     */
    function paused() public view returns (bool) {
        return _paused;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     */
    modifier whenNotPaused() {
        require(!_paused);
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     */
    modifier whenPaused() {
        require(_paused);
        _;
    }

    /**
     * @dev called by the owner to pause, triggers stopped state
     */
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }

    /**
     * @dev called by the owner to unpause, returns to normal state
     */
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
}

interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns (uint256);

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) internal _balances;
```

```
    mapping (address => mapping (address => uint256)) internal _allowed;

    uint256 private _totalSupply;

    /**
    * @dev Total number of tokens in existence
    */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
    * @dev Gets the balance of the specified address.
    * @param owner The address to query the balance of.
    * @return An uint256 representing the amount owned by the passed address.
    */
    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }

    /**
     * @dev Function to check the amount of tokens that an owner allowed to a spender.
     * @param owner address The address which owns the funds.
     * @param spender address The address which will spend the funds.
     * @return A uint256 specifying the amount of tokens still available for the spender.
     */
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowed[owner][spender];
    }

    /**
    * @dev Transfer token for a specified address
    * @param to The address to transfer to.
    * @param value The amount to be transferred.
    */
    function transfer(address to, uint256 value) public returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    /**
     * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
     * 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 desired value
afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     * @param spender The address which will spend the funds.
     * @param value The amount of tokens to be spent.
     */
    function approve(address spender, uint256 value) public returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);
        return true;
    }
```

```
    /**
    * @dev Transfer token for a specified addresses
    * @param from The address to transfer from.
    * @param to The address to transfer to.
    * @param value The amount to be transferred.
    */
    function _transfer(address from, address to, uint256 value) internal {
        require(to != address(0));

        _balances[from] = _balances[from].sub(value);
        _balances[to] = _balances[to].add(value);
        emit Transfer(from, to, value);
    }

    /**
     * @dev Internal function that mints an amount of the token and assigns it to
     * an account. This encapsulates the modification of balances such that the
     * proper events are emitted.
     * @param account The account that will receive the created tokens.
     * @param value The amount that will be created.
     */
    function _mint(address account, uint256 value) internal {
        require(account != address(0));
        _totalSupply = _totalSupply.add(value);
        _balances[account] = _balances[account].add(value);
        emit Transfer(address(0), account, value);
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account.
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    function _burn(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }
    /**
     * @dev Internal function that burns an amount of the token of a given
     * account, deducting from the sender's allowance for said account. Uses the
     * internal burn function.
     * Emits an Approval event (reflecting the reduced allowance).
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    function _burnFrom(address account, uint256 value) internal {
        _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(value);
        _burn(account, value);
        emit Approval(account, msg.sender, _allowed[account][msg.sender]);
    }
}
contract ERC20Burnable is ERC20 {
    /**
     * @dev Burns a specific amount of tokens.
     * @param value The amount of token to be burned.
     */
    function burn(uint256 value) public {
        _burn(msg.sender, value);
    }
```

```
    /**
     * @dev Burns a specific amount of tokens from the target address and decrements allowance
     * @param from address The address which you want to send tokens from
     * @param value uint256 The amount of token to be burned
     */
    function burnFrom(address from, uint256 value) public {
        _burnFrom(from, value);
    }
}

contract ERC20Pausable is ERC20, Pausable {
    function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transfer(to, value);
    }

    function transferFrom(address from, address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }

    function approve(address spender, uint256 value) public whenNotPaused returns (bool) {
        return super.approve(spender, value);
    }

    function increaseAllowance(address spender, uint addedValue) public whenNotPaused returns (bool
success) {
        return super.increaseAllowance(spender, addedValue);
    }

    function decreaseAllowance(address spender, uint subtractedValue) public whenNotPaused returns (bool
success) {
        return super.decreaseAllowance(spender, subtractedValue);
    }
}
contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }
    /**
     * @return the name of the token.
     */
    function name() public view returns (string memory) {
        return _name;
    }

    /**
     * @return the symbol of the token.
     */
    function symbol() public view returns (string memory) {
        return _symbol;
    }

    /**
     * @return the number of decimals of the token.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}
```

```
contract BASIC is ERC20Detailed, ERC20Pausable, ERC20Burnable {

    struct LockInfo {
        uint256 _releaseTime;
        uint256 _amount;
    }

    address public implementation;

    mapping (address => LockInfo[]) public timelockList;
       mapping (address => bool) public frozenAccount;

    event Freeze(address indexed holder);
    event Unfreeze(address indexed holder);
    event Lock(address indexed holder, uint256 value, uint256 releaseTime);
    event Unlock(address indexed holder, uint256 value);

    modifier notFrozen(address _holder) {
        require(!frozenAccount[_holder]);
        _;
    }

    constructor() ERC20Detailed("BASIC Token", "BASIC", 18) public  {

        _mint(msg.sender, 10000000000 * (10 ** 18));
    }

    function balanceOf(address owner) public view returns (uint256) {

        uint256 totalBalance = super.balanceOf(owner);
        if( timelockList[owner].length >0 ){
            for(uint i=0; i<timelockList[owner].length;i++){
                totalBalance = totalBalance.add(timelockList[owner][i]._amount);
            }
        }

        return totalBalance;
    }
    function transfer(address to, uint256 value) public notFrozen(msg.sender) returns (bool) {
        if (timelockList[msg.sender].length > 0 ) {
            _autoUnlock(msg.sender);
        }
        return super.transfer(to, value);
    }

    function transferFrom(address from, address to, uint256 value) public notFrozen(from) returns (bool) {
        if (timelockList[from].length > 0) {
            _autoUnlock(msg.sender);
        }
        return super.transferFrom(from, to, value);
    }

    function freezeAccount(address holder) public onlyPauser returns (bool) {
        require(!frozenAccount[holder]);
        frozenAccount[holder] = true;
        emit Freeze(holder);
        return true;
    }
    function unfreezeAccount(address holder) public onlyPauser returns (bool) {
        require(frozenAccount[holder]);
        frozenAccount[holder] = false;
        emit Unfreeze(holder);
```

```
    function lock(address holder, uint256 value, uint256 releaseTime) public onlyPauser returns (bool) {
        require(_balances[holder] >= value,"There is not enough balances of holder.");
        _lock(holder,value,releaseTime);


        return true;
    }

    function transferWithLock(address holder, uint256 value, uint256 releaseTime) public onlyPauser
returns (bool) {
        _transfer(msg.sender, holder, value);
        _lock(holder,value,releaseTime);
        return true;
    }

    function unlock(address holder, uint256 idx) public onlyPauser returns (bool) {
        require( timelockList[holder].length > idx, "There is not lock info.");
        _unlock(holder,idx);
        return true;
    }

    /**
     * @dev Upgrades the implementation address
     * @param _newImplementation address of the new implementation
     */
    function upgradeTo(address _newImplementation) public onlyOwner {
        require(implementation != _newImplementation);
        _setImplementation(_newImplementation);
    }

    function _lock(address holder, uint256 value, uint256 releaseTime) internal returns(bool) {
        _balances[holder] = _balances[holder].sub(value);
        timelockList[holder].push( LockInfo(releaseTime, value) );

        emit Lock(holder, value, releaseTime);
        return true;
    }

    function _unlock(address holder, uint256 idx) internal returns(bool) {
        LockInfo storage lockinfo = timelockList[holder][idx];
        uint256 releaseAmount = lockinfo._amount;

        delete timelockList[holder][idx];
        timelockList[holder][idx] = timelockList[holder][timelockList[holder].length.sub(1)];
        timelockList[holder].length -=1;

        emit Unlock(holder, releaseAmount);
        _balances[holder] = _balances[holder].add(releaseAmount);

        return true;
    }
    function _autoUnlock(address holder) internal returns(bool) {
        for(uint256 idx =0; idx < timelockList[holder].length ; idx++ ) {
            if (timelockList[holder][idx]._releaseTime <= now) {
                // If lockupinfo was deleted, loop restart at same position.
                if( _unlock(holder, idx) ) {
                    idx -=1;
                }
            }
        }
        return true;
    }
```

```
    /**
     * @dev Sets the address of the current implementation
     * @param _newImp address of the new implementation
     */
    function _setImplementation(address _newImp) internal {
        implementation = _newImp;
    }

    /**
     * @dev Fallback function allowing to perform a delegatecall
     * to the given implementation. This function will return
     * whatever the implementation call returns
     */
    function () payable external {
        address impl = implementation;
        require(impl != address(0));
        assembly {
            let ptr := mload(0x40)
            calldatacopy(ptr, 0, calldatasize)
            let result := delegatecall(gas, impl, ptr, calldatasize, 0, 0)
            let size := returndatasize
            returndatacopy(ptr, 0, size)

            switch result
            case 0 { revert(ptr, size) }
            default { return(ptr, size) }
        }
    }
}
```

Hexlant.
Blockchain Lab
-
contact@hexlant.com
www.hexlant.com