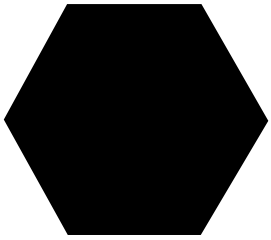
20 MAY 2020

Ludena
SMART CONTRACT AUDIT REPORT

Hexlant.



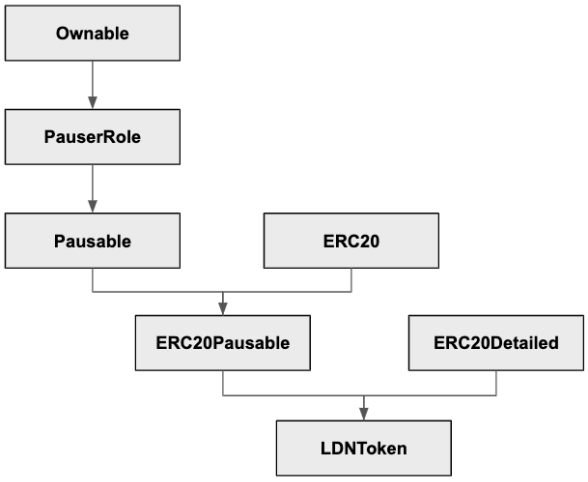
Analysis Purpose

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

Ludena 컨트랙트는 **Open-Zeppelin**에서 제공하는 컨트랙트 코드와 직접 구현한 컨트랙트 코드로 작성되었으며, 다음과 같은 컨트랙트를 통해 **Ludena**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**
  Ownership과 관련된 기능을 제공합니다. **onlyOwner modifier**을 사용하여 컨트랙트의 **Owner**일 경우 실행되도록 기능을 제공할 수 있습니다.

- **ERC20Detailed**
  컨트랙트 내 토큰의 이름, 심볼, 데시멀에 대한 기본 정보를 정의하며 정보를 제공합니다.

- **ERC20Pausable**
  컨트랙트 내 토큰 전송 정지와 관련된 기능을 제공합니다. **whenNotPaused, whenPaused modifier**를 사용하여 토큰 전송 시 현재 컨트랙트의 유통 제제 상황을 확인 후 전송이 이루워지도록 합니다.

- **LDNToken**
  Ludena에 필요한 추가 기능을 제공합니다. 특정 주소에 대한 동결 뿐아니라 락업 기능을 추가로 제공합니다.

Contract

상태 변수와 함수를 포함하여 컨테이너 형태의 계약을 표현하기 위해 사용

Contract	Description
ERC20	ERC20 표준 기능
Ownable	컨트랙트 오너 권한 관리
PauserRole	컨트랙트 정지 관리자 권한 관리
Pausable	컨트랙트 토큰 정지 상태 관리
ERC20Pausable	컨트랙트 토큰 정지 상태에 따른 전송 관리
ERC20Detailed	토큰 기본 정보 관리
LDNToken	메인 컨트랙트, 특정 주소 동결 및 락업 기능

Interface

컨트랙트 내 구현하고자 하는 표준함수를 정의하기 위해 사용

Interface	Description
IERC20	ERC20 인터페이스

Library

상태 변수를 갖을 수 없고 상속을 지원하지 않는 컨트랙트 라이브러리로, 라이브러리 내 함수가 호출되며 호출한 컨트랙트의 컨텍스트에서 실행

Library	Description
SafeMath	산술 연산시 발생가능 한 이슈 제어
Roles	특정 권한에 대한 주소의 추가 삭제 제어

Variable

컨트랙트의 상태를 표현하는 변수들로, 컨트랙트에 필요한 정보들을 저장하기위해 사용

Variable	Description
owner	컨트랙트 오너
newOwner	컨트랙트 신규 오너
_pausers	컨트랙트 정지 관리자 해시 테이블
_paused	컨트랙트 정지 상태
_balances	특정 주소의 토큰 잔액 해시 테이블
_allowed	특정 주소에게 출금이 위임된 토큰 잔액 해시 테이블
_totalSupply	토큰 총 발행량
_name	토큰 이름
_symbol	토큰 심볼
_decimals	토큰 데시멀
implementation	업그레이드 컨트랙트 주소
timelockList	특정 주소의 락업 리스트를 담은 테이블
frozenAccount	특정 주소의 동결 여부를 담은 해시 테이블

Modifier

함수의 한정요소로, 특정 기능을 수행할 때 한정된 조건에서만 실행 될 수 있도록 하기 위해 사용

Modifier	Description
onlyOwner	컨트랙트 오너만 실행 가능
onlyNewOwner	컨트랙트 신규 오너만 실행 가능
onlyPauser	컨트랙트 정지 관리자만 실행 가능
whenNotPaused	컨트랙트 비 정지 상태일 경우 실행 가능
whenPaused	컨트랙트 정지 상태일 경우 실행 가능
notFrozen	비 동결 상태일 경우 실행가능

Event

컨트랙트 함수 실행에 따른 로그 이벤트로 추후 애플리케이션 적용에 있어 컨트랙트 상황을 보다 쉽게 대응하기 위해 사용

Event	Description
OwnershipTranferred	오너 권한 이전 시 이벤트 발생
PauserAdded	정지 관리자 추가 시 이벤트 발생
PauserRemoved	정지 관리자 제거 시 이벤트 발생
Paused	정지 상태로 변경 시 이벤트 발생
Unpaused	비 정지 상태로 변경 시 이벤트 발생
Transfer	토큰 전송 시 이벤트 발생
Approval	출금 위임 시 이벤트 발생
Freeze	특정 주소 동결 시 이벤트 발생
Unfreeze	특정 주소 동결 해제 시 이벤트 발생
Lock	특정 주소의 토큰 락업 시 이벤트 발생
Unlock	특정 주소의 토큰 락업 해제 시 이벤트 발생

Function

컨트랙트의 함수들 로서 컨트랙트에 필요한 특정 로직을 담아 기능 실행을 하기 위해 사용

Function	Description
isOwner	컨트랙트 오너 인지 확인
transferOwnership	컨트랙트 오너 권한 이전
acceptOwnership	컨트랙트 오너 권한 이전 제안 수락
isPauser	컨트랙트 정지 관리자 인지 확인
addPauser	컨트랙트 정지 관리자 권한 부여
removePauser	컨트랙트 정지 관리자 권한 제거
renouncePauser	컨트랙트 정지 관리자 권한 포기
_addPauser	컨트랙트 정지 관리자 권한 추가
_removePauser	컨트랙트 정지 관리자 권한 제거
paused	컨트랙트 정지 상태 확인
pause	컨트랙트 정지
unpause	컨트랙트 정지 해제
totalSupply	토큰 총 발행량
balanceOf	특정 주소의 토큰 잔액
allowance	특정 주소에게 출금 위임된 토큰 잔액
transfer	토큰 전송
approve	특정 주소에 출금 위임
transferFrom	출금 위임된 토큰 전송
increaseAllowance	출금 위임된 토큰 잔액 증액
decreaseAllowance	출금 위임된 토큰 잔액 감액
_mint	토큰 추가 발행
_burn	토큰 소각
_burnFrom	출금 위임된 토큰 소각
name	토큰 이름
symbol	토큰 심볼
decimals	토큰 데시멀
freezeAccount	특정 주소 동결
unfreezeAccount	특정 주소 동결 해제
lock	특정 주소의 보유 토큰 락업
transferWithLock	특정 주소에게 락업된 토큰 전송
transferWithSliceLock	특정 주소에게 몇 차례 락업된 토큰 전송
upgradeTo	컨트랙트 업그레이드 주소 삽입
_lock	특정 주소의 보유 토큰 락업
_releaseTimeLock	특정 주소의 보유 락업 토큰 중 만료기간이 지난 토큰 락업 해제
_autoUnlock	만료기간을 지난 락업 토큰 해제
_setImplementation	컨트랙트 업그레이드 주소 삽입

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	(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	onlyOwner
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 └_addPauser			

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

Function Name	(PauserRole) renouncePauser		
Parameter	-		
Visibility	public	Modifier	-
Constant	-	Return	-
Callstack			
renouncePauser └_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	(Pausable) paused		
Parameter	-		
Visibility	public	Modifier	-
Constant	view	Return	bool
Callstack			
paused			

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

Function Name	(Pausable) unpause		
Parameter	-		
Visibility	public	Modifier	onlyPauser whenPaused
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, address		
Visibility	public	Modifier	-
Constant	view	Return	uint256
Callstack			
allowance			

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

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			
increaseAllowance			

Function Name	(ERC20) decreaseAllowance		
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 └ _burn			

Function Name	(ERC20Pausable) transfer		
Parameter	address, uint256		
Visibility	public	Modifier	whenNotPaused
Constant	-	Return	bool
Callstack			
transfer └ super.transfer			

Function Name	(ERC20Pausable) transferFrom		
Parameter	address, address, uint256		
Visibility	public	Modifier	whenNotPaused
Constant	-	Return	bool
Callstack			
transferFrom └ super.transferFrom			

Function Name	(ERC20Detailed) name		
Parameter	-		
Visibility	public	Modifier	-
Constant	view	Return	string
Callstack			
name			

Function Name	(ERC20Detailed) symbol		
Parameter	-		
Visibility	public	Modifier	-
Constant	view	Return	string
Callstack			
symbol			

Function Name	(ERC20Detailed) decimals		
Parameter	-		
Visibility	public	Modifier	-
Constant	view	Return	uint8
Callstack			
decimals			

Function Name	(LDNToken) balanceOf		
Parameter	address		
Visibility	public	Modifier	-
Constant	view	Return	uint256
Callstack			
balanceOf └ super.balanceOf			

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

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

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

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

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

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

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

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

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

Function Name	(LDNToken) _releaseTiemLock		
Parameter	address, uint256		
Visibility	internal	Modifier	-
Constant	-	Return	bool
Callstack			
_releaseTimeLock			

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

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

Test Result

Code Coverage

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

Ludena 컨트랙트는 Internal로 구현된 몇 개의 함수에 대해 추가적인 호출이 발생하지 않습니다.

아래의 Coverage 지표는 위 사항을 반영한 결과입니다.

File Name	Statements	Functions	Lines
LDNToken.sol	100% (147/147)	100% (56/56)	100% (154/154)

Test cases

Test case	Result
토큰 이름은 지정한 이름과 일치한다	PASS
토큰 심볼은 지정한 심볼과 일치한다.	PASS
토큰 데시멀은 지정한 데시멀과 일치한다.	PASS
지정한 초기 발행량이 총 발행량으로 할당된다.	PASS
지정한 초기 발행량이 컨트랙트의 오너(배포 실행 주소)에게 할당된다.	PASS
배포 후 오너 외 주소들의 토큰 잔액은 0이다.	PASS
토큰 전송 시 받는 주소가 0x00 일 경우 예외처리가 되는가?	PASS
토큰 전송 시 보내는 수량이 음수 일 경우 예외처리가 되는가?	PASS
토큰 전송 시 보유한 수량을 초과한 경우 예외처리가 되는가?	PASS
특정 주소에게 출금 위임 시 해당 주소의 출금 위임 잔액이 증액하는가?	PASS
특정 주소에게 출금 위임된 토큰 잔액을 증액 혹은 감액할 수 있는가?	PASS
특정 주소에게 출금 위임된 토큰 잔액 이상을 감액시키면 0이 되는가?	PASS
출금 위임받은 토큰 전송이 가능한가?	PASS
출금 위임받은 토큰 전송 시 관련 주소들의 토큰 잔액이 올바르게 업데이트 되는가?	PASS
출금 위임받은 토큰 전송 시 받는 주소가 0일 경우 예외처리가 되는가?	PASS
출금 위임받은 토큰 잔액을 초과한 수량을 전송 시 예외처리가 되는가?	PASS
출금을 위임한 주소의 토큰 보유 잔액이 부족할 경우 예외처리가 되는가?	PASS
컨트랙트 정지 관리자 외 주소로부터 토큰 전송 정지 기능 실행 시 예외처리가 되는가?	PASS

Test case	Result
컨트랙트 토큰 전송 정지 상태에서 토큰 전송 시 예외처리가 되는가?	PASS
컨트랙트 토큰 전송 정지 상태에서 출금 위임을 통한 토큰 전송 시 예외처리가 되는가?	PASS
컨트랙트 토큰 정지 상태를 해제 가능한가?	PASS
컨트랙트 정지 관리자 외 주소로부터 특정 주소 동결 시 예외처리가 되는가?	PASS
컨트랙트 정지 관리자 외 주소로부터 특정 주소 동결 해제 시 예외처리가 되는가?	PASS
동결된 주소로부터 토큰 전송 시 예외처리가 되는가?	PASS
동결된 주소로부터 출금 위임된 토큰이 전송 될 때 예외처리가 되는가?	PASS
컨트랙트 오너인지 여부를 확인 가능한가?	PASS
컨트랙트 오너 외의 주소로부터 오너 권한 이전 시 예외처리가 되는가?	PASS
컨트랙트 오너는 자신의 권한 이전에 대해 제안이 가능한가?	PASS
컨트랙트의 새로운 오너 제안을 받은 주소는 해당 제안에 수락 가능한가?	PASS
컨트랙트 정지 관리자 외 주소로부터 특정 주소에게 정지 관리자 권한 부여 시 예외처리가 되는가?	PASS
컨트랙트 오너 외 주소로부터 특정 주소의 정지 관리자 권한 박탈 시 예외처리가 되는가?	PASS
컨트랙트 정지 관리자는 특정 주소에게 정지 관리자 권한을 부여할 수 있는가?	PASS
컨트랙트 오너는 정지 관리자 권한을 박탈 할 수 있는가?	PASS
컨트랙트 정지 관리자는 자신의 권한을 포기할 수 있는가?	PASS
컨트랙트 정지 관리자 외 주소로부터 특정 주소의 보유 토큰 락업 시 예외처리가 되는가?	PASS
컨트랙트 정지 관리자 외 주소로부터 특정 주소에게 락업된 토큰 전송 시 예외처리가 되는가?	PASS
컨트랙트 정지 관리자 외 주소로부터 특정 주소에게 단계별 락업된 토큰 전송 시 예외처리가 되는가?	PASS
특정 주소의 보유 토큰 수량을 초과한 락업 시 예외처리가 되는가?	PASS
컨트랙트 정지 관리자는 특정 주소의 보유 토큰을 초과하지 않는 잔액을 락업 가능한가?	PASS
락업된 토큰이 만료기간을 지나지 않았을 경우 토큰 전송 시 예외처리가 되는가?	PASS
락업된 토큰이 만료기간을 지나면 토큰 전송이 가능한가?	PASS
락업된 토큰이 만료기간을 지나면 출금 위임을 통한 토큰 전송이 가능한가?	PASS
특정 주소에게 단계별 분배된 락업 토큰 전송 시 해당 주소는 일정 기간으로 락업 해제 된 토큰을 전송 가능한가?	PASS
최초 발행된 토큰이 컨트랙트 오너에게 전송되며 이벤트가 발생하는가?	PASS
토큰 전송 시 이벤트가 발생하는가?	PASS
특정 주소에게 출금 위임 시 이벤트가 발생하는가?	PASS
특정 주소에게 출금 위임 된 토큰 잔액을 증액 혹은 감액 시 이벤트가 발생하는가?	PASS
특정 주소에게 출금 위 임 된 토큰 전송 시 이벤트가 발생하는가?	PASS
특정 주소를 동결 및 동결 해제 시 이벤트가 발생하는가?	PASS
컨트랙트 오너 권한을 이전 시 이벤트가 발생하는가?	PASS
컨트랙트 정지 관리자 권한을 추가, 박탈 및 포기 시 이벤트가 발생하는가?	PASS
토큰이 락업 될 시 이벤트가 발생하는가?	PASS
락업된 토큰이 해제 될 시 이벤트가 발생하는가?	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

Ludena 컨트랙트는 **ERC-20** 인터페이스에 맞춰 작성되었으며 토큰의 소각 및 추가 발행과 같은 토큰 수량을 증액, 감액할 수 있는 기능은 별도 구현되어 있지않습니다. 다만, 특정 주소의 동결 및 컨트랙트 토큰 전송 정지와 같은 제제 기능을 통해 토큰 유통량에 대해 일시적 제제를 가할 수 있습니다. 위 기능은 주소 해킹 및 토큰 탈취와 같은 상황에 있어 악의적 주소의 이득을 쉽게 막을 수 있습니다. 이러한 기능에 대한 권한은 하나의 주소에 국한되어 있지 않고 정지 관리자라는 권한을 두어 다수에게 권한을 분배할 수 있도록 합니다.

락업 기능의 경우, 만료 시점이 지나면 모두 해제되는 락업과 만료 시점을 기점으로 일정 간격으로 해제가 가능한 락업이 존재합니다. 락업 정보는 특정 주소의 권한을 통해 변경, 해제할 수 없도록 구현되어 있습니다.

컨트랙트가 기능 별로 짜임새 있게 구성되어 있으며 **Openzeppelin**의 검증된 코드를 기반으로 작성되어 있기 때문에 보안적 문제를 발견하지 못하였습니다.

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 ^0.5.17;

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 tokens from one address to another.
     * Note that while this function emits an Approval event, this is not required as per the
specification,
     * and other compliant implementations may not emit the event.
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
```

```
 * @param value uint256 the amount of tokens to be transferred
 */
function transferFrom(address from, address to, uint256 value) public returns (bool) {
    _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
    _transfer(from, to, value);
    emit Approval(from, msg.sender, _allowed[from][msg.sender]);
    return true;
}

/**
 * @dev Increase the amount of tokens that an owner allowed to a spender.
 * approve should be called when allowed_[_spender] == 0. To increment
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * Emits an Approval event.
 * @param spender The address which will spend the funds.
 * @param addedValue The amount of tokens to increase the allowance by.
 */
function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
    require(spender != address(0));

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

/**
 * @dev Decrease the amount of tokens that an owner allowed to a spender.
 * approve should be called when allowed_[_spender] == 0. To decrement
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * Emits an Approval event.
 * @param spender The address which will spend the funds.
 * @param subtractedValue The amount of tokens to decrease the allowance by.
 */
function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
    require(spender != address(0));

    _allowed[msg.sender][spender] = _allowed[msg.sender][spender].sub(subtractedValue);
    emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
    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 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);
    }

    /*
     * approve/increaseApprove/decreaseApprove can be set when Paused state
     */

    /*
     * 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 LDNToken is ERC20Detailed, ERC20Pausable {

    struct LockInfo {
        uint256 _releaseTime;
        uint256 _amount;
        uint256 _remainingAmount;
        uint256 _termOfRound;
        uint256 _unlockAmountPerRound;
    }

    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("Ludena Protocol", "LDN", 18) public  {

        _mint(msg.sender, 1200000000 * (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]._remainingAmount );
            }
        }

        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(from);
        }
        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);
        return true;
    }

    function lock(address holder, uint256 value, uint256 releaseStart, uint256 termOfRound, uint256
unlockAmountPerRound) public onlyPauser returns (bool) {

        _lock(holder, value, releaseStart, termOfRound, unlockAmountPerRound);

        return true;
    }

    function transferWithLock(address holder, uint256 value, uint256 releaseTime) public onlyPauser
returns (bool) {
        _transfer(msg.sender, holder, value);
        _lock(holder, value, releaseTime, 1, value);
        return true;
    }
```

```
    function transferWithSliceLock(address holder, uint256 value, uint256 releaseStart, uint256
termOfRound, uint256 unlockAmountPerRound) public onlyPauser returns (bool) {
        _transfer(msg.sender, holder, value);
        _lock(holder, value, releaseStart, termOfRound, unlockAmountPerRound);
        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, uint256 termOfRound, uint256
unlockAmountPerRound) internal returns(bool) {
        require(_balances[holder] >= value,"There is not enough balances of holder.");
        _balances[holder] = _balances[holder].sub(value);
        timelockList[holder].push( LockInfo(releaseTime, value, value, termOfRound,
unlockAmountPerRound) );

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

    function _releaseTimeLock(address _holder, uint256 _idx) internal returns(bool) {
        require(_idx < timelockList[_holder].length);

        // If lock status of holder is finished, delete lockup info.
        LockInfo storage info = timelockList[_holder][_idx];
        uint256 releaseAmount = info._unlockAmountPerRound;
        uint256 sinceFrom = now.sub(info._releaseTime);
        uint256 sinceRound = sinceFrom.div(info._termOfRound);
        releaseAmount = releaseAmount.add( sinceRound.mul(info._unlockAmountPerRound) );

        if(releaseAmount >= info._remainingAmount) {
            releaseAmount = info._remainingAmount;

            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;
        } else {
            timelockList[_holder][_idx]._releaseTime     = timelockList[_holder]
[_idx]._releaseTime.add( sinceRound.add(1).mul(info._termOfRound) );
            timelockList[_holder][_idx]._remainingAmount = timelockList[_holder]
[_idx]._remainingAmount.sub(releaseAmount);

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

            return false;
        }
    }
```

```
    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( _releaseTimeLock(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) }
        }
    }
}
```

```
/**
 *Submitted for verification at Etherscan.io on 2020-04-20
*/

pragma solidity ^0.5.4;

//////////////////////////////////////////////
////////LOAPROTOCOL Tame simonKim//////////////
//////////////////////////////////////////////

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);
}



library SafeMath {

    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;
    }


    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;
    }


    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }
```

```
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }


    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}


contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) internal _balances;

    mapping (address => mapping (address => uint256)) private _allowed;

    uint256 private _totalSupply;


    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }


    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }


    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowed[owner][spender];
    }


    function transfer(address to, uint256 value) public returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }


    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;
    }


    function transferFrom(address from, address to, uint256 value) public returns (bool) {
        _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
        _transfer(from, to, value);
        emit Approval(from, msg.sender, _allowed[from][msg.sender]);
        return true;
    }
```

```
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        require(spender != address(0));

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


    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        require(spender != address(0));

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

    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);
    }


    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);
    }


    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);
    }


    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 LOAPROTOCOL is ERC20 {
    string public constant name = "LOAPROTOCOL"; // solium-disable-line uppercase
    string public constant symbol = "LOA"; // solium-disable-line uppercase
    uint8 public constant decimals = 18; // solium-disable-line uppercase
    uint256 public constant initialSupply = 2000000000 * (10 ** uint256(decimals));

    constructor() public {
        super._mint(msg.sender, initialSupply);
        owner = msg.sender;
    }
```

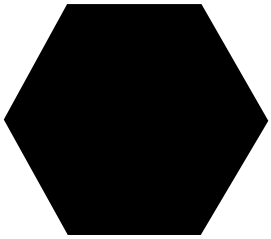
```
  //ownership
  address public owner;

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

  modifier onlyOwner() {
      require(msg.sender == owner, "Not owner");
      _;
  }

/**
 * @dev Allows the current owner to relinquish control of the contract.
 * @notice Renouncing to ownership will leave the contract without an owner.
 * It will not be possible to call the functions with the `onlyOwner`
 * modifier anymore.
 */
  function renounceOwnership() public onlyOwner {
      emit OwnershipRenounced(owner);
      owner = address(0);
  }

/**
 * @dev Allows the current owner to transfer control of the contract to a newOwner.
 * @param _newOwner The address to transfer ownership to.
 */
  function transferOwnership(address _newOwner) public onlyOwner {
      _transferOwnership(_newOwner);
  }

/**
 * @dev Transfers control of the contract to a newOwner.
 * @param _newOwner The address to transfer ownership to.
 */
  function _transferOwnership(address _newOwner) internal {
      require(_newOwner != address(0), "Already owner");
      emit OwnershipTransferred(owner, _newOwner);
      owner = _newOwner;
  }

  //pausable
  event Pause();
  event Unpause();

  bool public paused = false;

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

  /**
  * @dev Modifier to make a function callable only when the contract is paused.
  */
  modifier whenPaused() {
      require(paused, "Not paused now");
      _;
  }
```

```
/**
 * @dev called by the owner to pause, triggers stopped state
 */
function pause() public onlyOwner whenNotPaused {
    paused = true;
    emit Pause();
}

/**
* @dev called by the owner to unpause, returns to normal state
*/
function unpause() public onlyOwner whenPaused {
    paused = false;
    emit Unpause();
}

//freezable
event Frozen(address target);
event Unfrozen(address target);

mapping(address => bool) internal freezes;

modifier whenNotFrozen() {
    require(!freezes[msg.sender], "Sender account is locked.");
    _;
}

function freeze(address _target) public onlyOwner {
    freezes[_target] = true;
    emit Frozen(_target);
}

function unfreeze(address _target) public onlyOwner {
    freezes[_target] = false;
    emit Unfrozen(_target);
}

function isFrozen(address _target) public view returns (bool) {
    return freezes[_target];
}

function transfer(
    address _to,
    uint256 _value
)
  public
  whenNotFrozen
  whenNotPaused
  returns (bool)
{
    releaseLock(msg.sender);
    return super.transfer(_to, _value);
}

function transferFrom(
    address _from,
    address _to,
    uint256 _value
)
  public
  whenNotPaused
  returns (bool)
```

```
    {
        require(!freezes[_from], "From account is locked.");
        releaseLock(_from);
        return super.transferFrom(_from, _to, _value);
    }



    //lockable
    struct LockInfo {
        uint256 releaseTime;
        uint256 balance;
    }
    mapping(address => LockInfo[]) internal lockInfo;

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

    function balanceOf(address _holder) public view returns (uint256 balance) {
        uint256 lockedBalance = 0;
        for(uint256 i = 0; i < lockInfo[_holder].length ; i++ ) {
            lockedBalance = lockedBalance.add(lockInfo[_holder][i].balance);
        }
        return super.balanceOf(_holder).add(lockedBalance);
    }

    function releaseLock(address _holder) internal {

        for(uint256 i = 0; i < lockInfo[_holder].length ; i++ ) {
            if (lockInfo[_holder][i].releaseTime <= now) {
                _balances[_holder] = _balances[_holder].add(lockInfo[_holder][i].balance);
                emit Unlock(_holder, lockInfo[_holder][i].balance);
                lockInfo[_holder][i].balance = 0;

                if (i != lockInfo[_holder].length - 1) {
                    lockInfo[_holder][i] = lockInfo[_holder][lockInfo[_holder].length - 1];
                    i--;
                }
                lockInfo[_holder].length--;

            }
        }
    }
    function lockCount(address _holder) public view returns (uint256) {
        return lockInfo[_holder].length;
    }
    function lockState(address _holder, uint256 _idx) public view returns (uint256, uint256) {
        return (lockInfo[_holder][_idx].releaseTime, lockInfo[_holder][_idx].balance);
    }

    function lock(address _holder, uint256 _amount, uint256 _releaseTime) public onlyOwner {
        require(super.balanceOf(_holder) >= _amount, "Balance is too small.");
        _balances[_holder] = _balances[_holder].sub(_amount);
        lockInfo[_holder].push(
            LockInfo(_releaseTime, _amount)
        );
        emit Lock(_holder, _amount, _releaseTime);
    }

    function lockAfter(address _holder, uint256 _amount, uint256 _afterTime) public onlyOwner {
        require(super.balanceOf(_holder) >= _amount, "Balance is too small.");
        _balances[_holder] = _balances[_holder].sub(_amount);
        lockInfo[_holder].push(
            LockInfo(now + _afterTime, _amount)
        );
```

```
       emit Lock(_holder, _amount, now + _afterTime);
    }

    function unlock(address _holder, uint256 i) public onlyOwner {
        require(i < lockInfo[_holder].length, "No lock information.");

        _balances[_holder] = _balances[_holder].add(lockInfo[_holder][i].balance);
        emit Unlock(_holder, lockInfo[_holder][i].balance);
        lockInfo[_holder][i].balance = 0;

        if (i != lockInfo[_holder].length - 1) {
            lockInfo[_holder][i] = lockInfo[_holder][lockInfo[_holder].length - 1];
        }
        lockInfo[_holder].length--;
    }

    function transferWithLock(address _to, uint256 _value, uint256 _releaseTime) public onlyOwner returns
(bool) {
        require(_to != address(0), "wrong address");
        require(_value <= super.balanceOf(owner), "Not enough balance");

        _balances[owner] = _balances[owner].sub(_value);
        lockInfo[_to].push(
            LockInfo(_releaseTime, _value)
        );
        emit Transfer(owner, _to, _value);
        emit Lock(_to, _value, _releaseTime);

        return true;
    }

    function transferWithLockAfter(address _to, uint256 _value, uint256 _afterTime) public onlyOwner
returns (bool) {
        require(_to != address(0), "wrong address");
        require(_value <= super.balanceOf(owner), "Not enough balance");

        _balances[owner] = _balances[owner].sub(_value);
        lockInfo[_to].push(
            LockInfo(now + _afterTime, _value)
        );
        emit Transfer(owner, _to, _value);
        emit Lock(_to, _value, now + _afterTime);

        return true;
    }

    function currentTime() public view returns (uint256) {
        return now;
    }

    function afterTime(uint256 _value) public view returns (uint256) {
        return now + _value;
    }
}
```

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



HEXLANT CONTRACT CERTIFICATION

This contract speicifics that it has been validated by the Hexlant Technical Team and notifies that it has not any technical defects.

PUBLISHIED INFORMATION

REPORT NUMBER	ERC20200520
DATE	2020/05/20
PUBLISHIER	SEONGEUN CHO eun@hexlant.com

TOKEN INFORMATION

TOKEN NAME	Ludena Protocol		
SYMBOL	LDN		
PLATFORM	ETHEREUM	**TOKEN TYPE**	ERC-20
TOTAL SUPPLY	1,200,000,000 LDN		
CONTRACT ADDRESS	0xb29663Aa4E2e81e425294193616c1B102B70a158		

VULNERABLILLITY ANALYSIS

CRITICAL	0	No relevant provision
HIGH	0	No relevant provision
MEDIUM	0	No relevant provision
LOW	0	No relevant provision

CENTRALIZED FUNCTIONS

FREEZE	YES	Ability to freeze tokens in accounts. (The administrator can freeze the hacker's account in case of hacking.)
PAUSE	YES	Ability to pause functions related to token transmission in a contract. (This is used when the administrator needs to prevent the movement of assets due to token swaps or hacking.)
LOCKUP	YES	Ability to block token transfers for a period of time (Administrators can use to set lockout periods for investors, team members, advisors, etc.)
BURN	NO	Ability to reduce total supply by burning tokens
MINT	NO	Ability to increase total supply by minting tokens

Certified by Hexlant. _____