Cyberscope

Audit Report

PinkSale Lock

May 2022

Github	https://github.com/pinkmoonfinance/pink-lock-contracts-v3
Commit	a4c47e837a098fd1be62bac21e0abd1094b24a2e
Audited by	© cyberscope

Table of Contents

Contract Review

Github	https://github.com/pinkmoonfinance/pink-lock-contracts-v3
Commit	a4c47e837a098fd1be62bac21e0abd1094b24a2e

Audit Updates

Initial Audit	10th May 2022
Corrected	

Source Files

Filename	SHA256
Address.sol	aafa8f3e41700a8353aabcdf020e06735753e6bc4b615279b43de53cfbb4f2cd
EnumerableSet.sol	42a618e7d36efd2319d1bf05fedb31f3042baf535cfb97e783128cc1fe326686
FullMath.sol	96da42d2eaedb03bd63f35d99eded7c53c91af25238187839eeb7b88649eaf55
IERC20.sol	c2b06bb4572bb4f84bfc5477dadc0fcc497cb66c3a1bd53480e68bedc2e154a6
IPinkLock.sol	da78c9501b2bf3bfb9b54caee6cc3bdc89dbe175a9fd3aeed83bb91c290d9e78
IUniswapV2Factory.sol	706cad52cb3050b2daeaee694350292b686f1f22b5f2f62d67316ede0d01f313
IUniswapV2Pair.sol	70f822016e80f20f143d1ee3000be556e60582ca89eb13619825197155ada0bc
IUniswapV2Router02.sol	dc735ab516e369252436feb5b1fcb9700b805acdca9f64671503703810752038
PinkLock02.sol	cdcfd52a8ffec610662950758e830fe16fcf555d086bbb2cf8ff42ba461cd9da
SafeERC20.sol	931862da14d546841de2f9f1b1b183502a276d0c89255d867dc147738c80d2c5

Contract Diagnostics

🔴 Critical 🟠 Medium ● Minor

Severity	Code	Description
●	IDTGC	ID Token Generation Conflict
●	NLTD	Normal and LP Token Duplication
●	LPTM	LP Token Mocking
●	TPC	Transposition Property Check
●	PI	Performance Improvement (1/2)
●	PI	Performance Improvement (2/2)
●	LLS	Locker Logic Simplification
●	L12	Using Variables before Declaration
●	L13	Divide before Multiply Operation
●	L14	Uninitialized Variables in Local Scope

ID Token Generation Conflict

Criticality	minor
Location	contract/PinkLock02.sol#L368

Description

The contract generates ids based on the fact that the previous versions will not reach the ID_PADDING amount of locks. This is an arbitrary assumption that may break if someone abuses the previous versions and generates an ID_PADDING amount of locks.

```
id = _locks.length + ID_PADDING;
```

Recommendation

The contract could use a reverse counter mechanism so it will be essentially impossible to generate such a large amount of lockers.

```
const id = ~uint256(0) - _locks.length
```

```
_getActualIndex -> ~uint256(0) - id
```

Normal and LP Token Duplication

Criticality	minor
Location	contract/PinkLock02.sol#L348,456

Description

The user has the ability to lock the LP token in the normal token lock. Later, if the user locks the LP token the factory address will not be registered since the token will already exist. Hence, since the contract uses the `bool isLpToken = tokenInfo.factory != address(0);` to determine the locker state, the contract will always mark it as a simple token and the LP token locker will be inaccessible.

```
// Normal Token
CumulativeLockInfo storage tokenInfo = cumulativeLockInfo[token];
if (tokenInfo.token == address(0)) {
    tokenInfo.token = token;
    tokenInfo.factory = address(0);
}
// LP Token
CumulativeLockInfo storage tokenInfo = cumulativeLockInfo[token];
if (tokenInfo.token == address(0)) {
    tokenInfo.token = token;
    tokenInfo.factory = factory;
}
```

Recommendation

The contract could implement a structure that is not based on the fact that a token address will be either a normal token or an LP token. Alternatively, the contract could check the factory shape in order to not allow locking an LP token as a simple token.

LP Token Mocking

Criticality	minor
Location	contract/PinkLock02.sol#L862

Description

The user can create a contract that implements the factory(), getPair() and token0(), token1() method in order to mock the LP token validator. As a result the user will be able to lock an LP Token that essentially is not an LP token.

```
function _isValidLpToken(address token, address factory)
    private
    view
    returns (bool)
{
    IUniswapV2Pair pair = IUniswapV2Pair(token);
    address factoryPair = IUniswapV2Factory(factory).getPair(
        pair.token0(),
        pair.token1()
    );
    return factoryPair == token;
}
```

Transposition Property Check

Criticality	minor
Location	contract/PinkLock02.sol#L138

Description

According to the transposition property, if the sum of two integers is less than n, then each integer will be less than n.

```
require(tgeBps > 0 && tgeBps < 10_000, "Invalid bips for TGE");
require(cycleBps > 0 && cycleBps < 10_000, "Invalid bips for cycle");
require(
    tgeBps + cycleBps <= 10_000,
    "Sum of TGE bps and cycle should be less than 10000"
);
```

Recommendation

The contract could skip the individual checks since the sum is checked.

Performance Improvement (1/2)

Criticality	minor
Location	contract/PinkLock02.sol#L205

Description

Since the owners and the amount array have the same size and the owners array is iterated, then the initial _sumAmount() calculation is redundant. The calculation of the amount could be moved inside the owners loop.

```
uint256 sumAmount = _sumAmount(amounts);
uint256 count = owners.length;
uint256[] memory ids = new uint256[](count);
for (uint256 i = 0; i < count; i++) {
    ids[i] = _createLock(
        owners[i],
        token,
        isLpToken,
        amounts[i],
        vestingSettings[0], // TGE date
        vestingSettings[1], // TGE bps
        vestingSettings[2], // cycle
        vestingSettings[3], // cycle bps
        description
    );
    emit LockAdded(
        ids[i],
        token,
        owners[i],
        amounts[i],
        vestingSettings[0] // TGE date
    );
}
_safeTransferFromEnsureExactAmount(
    token,
    msg.sender,
    address(this),
    sumAmount
);
```

Recommendation

The contract could merge the two loops in one and calculate the sumAmount in the owners loop.

Performance Improvement (2/2)

Criticality	minor
Location	contract/PinkLock02.sol#L534

Description

Since the `block.timestamp < userLock.tgeDate` is checked in the beginning of the method, then the expression `if (block.timestamp >= userLock.tgeDate) {` is redundant.

```
if (block.timestamp < userLock.tgeDate) return 0;
if (userLock.cycle == 0) return 0;

uint256 tgeReleaseAmount = FullMath.mulDiv(
    userLock.amount,
    userLock.tgeBps,
    10_000
);
uint256 cycleReleaseAmount = FullMath.mulDiv(
    userLock.amount,
    userLock.cycleBps,
    10_000
);
uint256 currentTotal = 0;
if (block.timestamp >= userLock.tgeDate) {
    currentTotal =
        (((block.timestamp - userLock.tgeDate) / userLock.cycle) *
            cycleReleaseAmount) +
        tgeReleaseAmount; // Truncation is expected here
}
```

Recommendation

The contract could remove the if branch since the expression will always yield positive value.

Locker Logic Simplification

Criticality	minor
Location	contract/PinkLock02.sol#L21

Description

The normal and vesting lock could be the same since normal lock is equal to the vesting lock with 100% return in the first circle.

```
Normal lock == Vesting with:
tgeBps = 10_000
tgeDate = lock date
cycleBps = lock start date
cycle = 1
```

Recommendation

The contract could merge and simplify the logic of the vesting and normal token lock.

```
struct Lock {
    uint256 id;
    address token;
    address owner;
    uint256 amount;
    uint256 lockDate;
    uint256 tgeDate; // TGE date for vesting locks, unlock date for normal locks
    uint256 tgeBps; // In bips. Is 0 for normal locks
    uint256 cycle; // Is 0 for normal locks
    uint256 cycleBps; // In bips. Is 0 for normal locks
    uint256 unlockedAmount;
    string description;
    bool isVesting;
}
```

L12 - Using Variables before Declaration

Criticality	minor
Location	contract/PinkLock02.sol#L849

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or the variable has been declared in a different scope.

```
factory
```

Recommendation

The variables should be declared before any usage of them.

L13 - Divide before Multiply Operation

Criticality	minor
Location	contract/PinkLock02.sol#L513

Description

Performing divisions before multiplications may cause lose of prediction.

```
currentTotal = (((block.timestamp - userLock.tgeDate) / userLock.cycle) *
cycleReleaseAmount) + tgeReleaseAmount
```

Recommendation

The multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	minor
Location	contract/PinkLock02.sol#L849,848

Description

The are variables that are defined in the local scope and are not initialized.

```
possibleFactoryAddress
factory
```

Recommendation

All the local scoped variables should be initialized.

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
EnumerableSet	Library			
	_add	Private	✓	
	_remove	Private	✓	
	_contains	Private		
	_length	Private		
	_at	Private		
	_values	Private		
	add	Internal	✓	
	remove	Internal	✓	
	contains	Internal		
	length	Internal		
	at	Internal		
	values	Internal		
	add	Internal	✓	
	remove	Internal	✓	

	contains	Internal		
	length	Internal		
	at	Internal		
	values	Internal		
	add	Internal	✓	
	remove	Internal	✓	
	contains	Internal		
	length	Internal		
	at	Internal		
	values	Internal		
FullMath	Library			
	mulDiv	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IPinkLock	Interface			
	lock	External	✓	-
	vestingLock	External	✓	-
	multipleVestingLock	External	✓	-
	unlock	External	✓	-
	editLock	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-

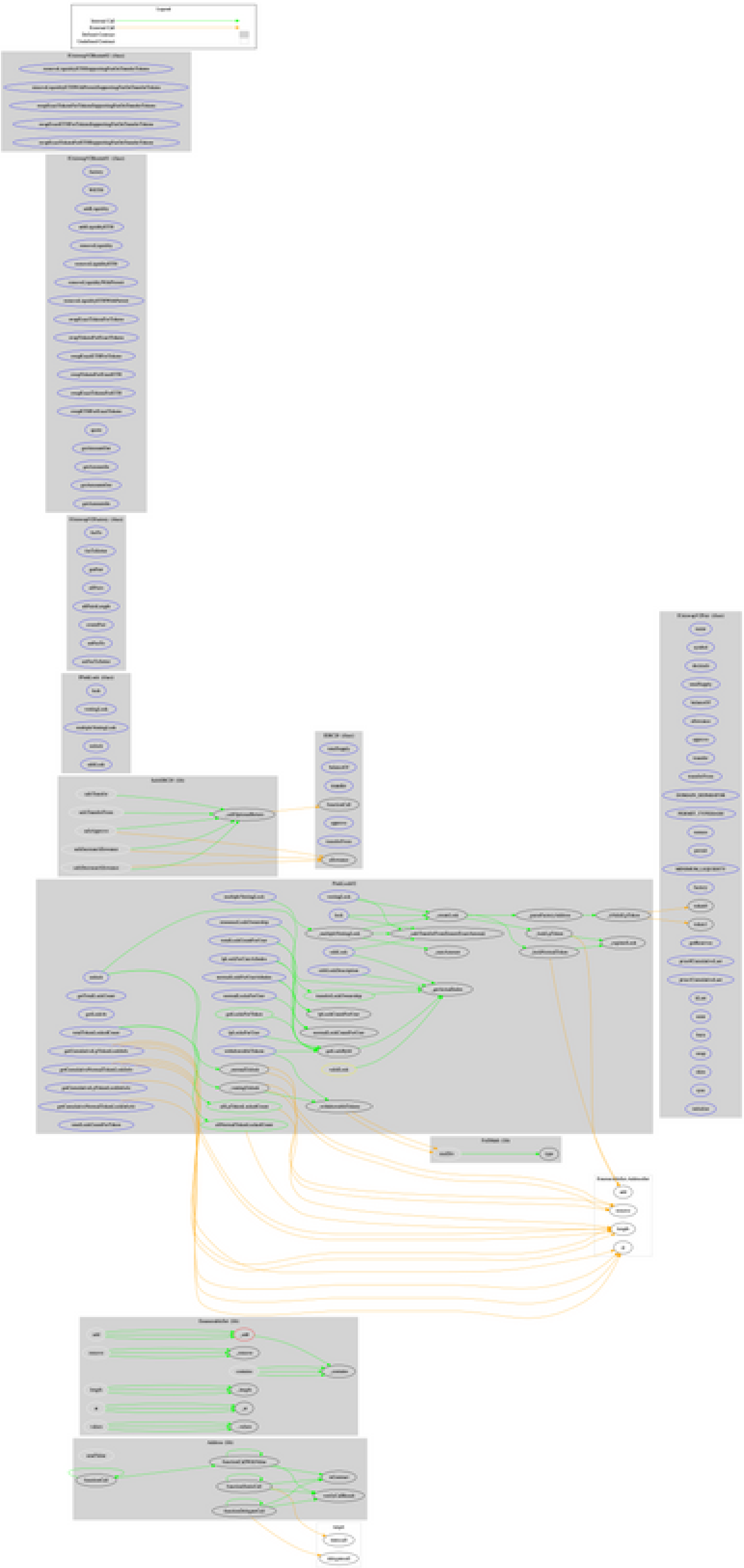
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			

	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
PinkLock02	Implementation	IPinkLock		
	lock	External	✓	-
	vestingLock	External	✓	-
	multipleVestingLock	External	✓	-

	_multipleVestingLock	Internal	✓	
	_sumAmount	Internal		
	_createLock	Internal	✓	
	_lockLpToken	Private	✓	
	_lockNormalToken	Private	✓	
	_registerLock	Private	✓	
	unlock	External	✓	validLock
	_normalUnlock	Internal	✓	
	_vestingUnlock	Internal	✓	
	withdrawableTokens	External		-
	_withdrawableTokens	Internal		
	editLock	External	✓	validLock
	editLockDescription	External	✓	validLock
	transferLockOwnership	Public	✓	validLock
	renounceLockOwnership	External	✓	-
	_safeTransferFromEnsureExactAmount	Internal	✓	
	getTotalLockCount	External		-
	getLockAt	External		-
	getLockById	Public		-
	allLpTokenLockedCount	Public		-
	allNormalTokenLockedCount	Public		-
	getCumulativeLpTokenLockInfoAt	External		-
	getCumulativeNormalTokenLockInfoAt	External		-
	getCumulativeLpTokenLockInfo	External		-
	getCumulativeNormalTokenLockInfo	External		-
	totalTokenLockedCount	External		-
	lpLockCountForUser	Public		-
	lpLocksForUser	External		-
	lpLockForUserAtIndex	External		-
	normalLockCountForUser	Public		-
	normalLocksForUser	External		-
	normalLockForUserAtIndex	External		-
	totalLockCountForUser	External		-
	totalLockCountForToken	External		-
	getLocksForToken	Public		-

	_getActualIndex	Internal		
	_parseFactoryAddress	Internal		
	_isValidLpToken	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	

Contract Flow



Summary

PinkSale locker implements a token logic functionality for normal and LP tokens. The contract supports two different types of lock. The normal lock, where the tokens are locked for a specific period of time. The vesting lock, where tokens are locked proportional to the time period that has elapsed. This audit focuses on the business logic implementation, security concerns and performance optimizations.

Disclaimer

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment.

Cyberscope team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed.

The Cyberscope team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Cyberscope receive a payment to manipulate those results or change the awarding badge that we will be adding in our website.

Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token.

The Cyberscope team disclaims any liability for the resulting losses.

About Cyberscope

Coinscope audit and K.Y.C. service has been rebranded to Cyberscope.

Coinscope is the leading early coin listing, voting and auditing authority firm. The audit process is analyzing and monitoring many aspects of the project. That way, it gives the community a good sense of security using an informative report and a generic score.

Cyberscope and Coinscope are aiming to make crypto discoverable and efficient globally. They provides all the essential tools to assist users draw their own conclusions.

The Cyberscope team

https://www.cyberscope.io