Cyberscope

Audit Report

GemPad Lockers

February 2024

Network BSC

Address 0x9306eb244e68a8337ad2f54a42595b07e0c0622b

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **3**
Audit Updates 3
Source Files 3
Overview **6**
Function multipleLock 6
Function multipleVestingLock 6
Function lockLpV3 6
Function unlock 7
Function unlockAllAvailable 7
Function editLock 7
Function increaseLiquidityCurrentRange 8
Function decreaseLiquidityCurrentRange 8
Function collectFees 8
Function editLockDescription 9
Function editProjectTokenMetaData 9
Function transferProjectOwnerShip 9
Function transferLockOwnership 9
Roles **11**
Owner 11
Users 11
Findings Breakdown **13**
Diagnostics **14**
CCR - Contract Centralization Risk 15
Description 15
Recommendation 15
ELFM - Exceeds Fees Limit 17
Description 17
Recommendation 17
MOA - Misleading Owner Assignment 19
Description 19
Recommendation 20
MU - Modifiers Usage 22
Description 22
Recommendation 22
L19 - Stable Compiler Version 23
Description 23
Recommendation 23
Functions Analysis **24**

Inheritance Graph 27
Flow Graph 28
Summary 29
Disclaimer 30
About Cyberscope 31

Review

Explorer	https://bscscan.com/address/0x9306eb244e68a8337ad2f54a42595b07e0c0622b

Audit Updates

Initial Audit	30 Jan 2024 https://github.com/cyberscope-io/audits/blob/main/gempad/v1/audit.pdf
Corrected Phase 2	02 Feb 2024

Source Files

Filename	SHA256
GempadLock.sol	d1fc4c73738a8222492d0ab7d5d636b0c89b9d3f031dff4ef0d0520aa074ac25
FullMath.sol	cf33688bcc87ed97503be8c33c733686b15469e83b15e4d9969f9f0ed4f04fcc
interfaces/IUniswapV3Pair.sol	d8314d7f1e0ace4292a06daa32bccd07ae4af97558da6d484e24830a9c7266a9
interfaces/IUniswapV3Factory.sol	cd9959c50c8bc592d729c6458bfb1d471e0ff1a799de609f54f57946da855dcf
interfaces/IUniswapV2Pair.sol	d2a719db1ef447e334a57cba344e05ea85ec3fb6766ef717b4448fc4a5032634
interfaces/IUniswapV2Factory.sol	a63a844ad84f9df76c9822e73adf2f66b6e0c100eece0c4af5103734cb3f4698
interfaces/IPoolInitializer.sol	1c6c3661807129156f46ac0e3a8a582a2600cbfe983751b79844981b573ac33a

interfaces/IPeripheryPayments.sol	ccecfe115faf38c5aa2f805f9b11826b8d0b906334fd8782c6e27e8fef43860e
interfaces/IPeripheryImmutableState.sol	87a6be0e845e5fef99ea73f2503ff2e9ab4244f6e87fd1f82416d93d152082eb
interfaces/INonfungiblePositionManager.sol	313e6595a8a74ffcddbf3cd385eb60d4af5240a5b279d71ba5bf9add360627dd
interfaces/ILockV2.sol	f053c5c524e980ebbbdd096a2d5bc40d1c6625eaabec6b8d011ce4de82f93d05
interfaces/IGempadLock.sol	7f2a6119252e38a8ef41b0751e63c9ccbac50c7817c94bffcc688c21723c4044
interfaces/IERC721Permit.sol	d6e1c8868b63e77027761be60b3c7e9724416c4cea8b148889b9066458a144c7
contracts/interfaces/IUniswapV3Pair.sol	d8314d7f1e0ace4292a06daa32bccd07ae4af97558da6d484e24830a9c7266a9
contracts/interfaces/IUniswapV3Factory.sol	cd9959c50c8bc592d729c6458bfb1d471e0ff1a799de609f54f57946da855dcf
contracts/interfaces/IUniswapV2Pair.sol	d2a719db1ef447e334a57cba344e05ea85ec3fb6766ef717b4448fc4a5032634
contracts/interfaces/IUniswapV2Factory.sol	a63a844ad84f9df76c9822e73adf2f66b6e0c100eece0c4af5103734cb3f4698
contracts/interfaces/IPoolInitializer.sol	1c6c3661807129156f46ac0e3a8a582a2600cbfe983751b79844981b573ac33a
contracts/interfaces/IPeripheryPayments.sol	ccecfe115faf38c5aa2f805f9b11826b8d0b906334fd8782c6e27e8fef43860e
contracts/interfaces/IPeripheryImmutableState.sol	87a6be0e845e5fef99ea73f2503ff2e9ab4244f6e87fd1f82416d93d152082eb
contracts/interfaces/INonfungiblePositionManager.sol	313e6595a8a74ffcddbf3cd385eb60d4af5240a5b279d71ba5bf9add360627dd

contracts/interfaces/ILockV2.sol	f053c5c524e980ebbbdd096a2d5bc40d1c6625eaabec6b8d011ce4de82f93d05
contracts/interfaces/IERC721Permit.sol	d6e1c8868b63e77027761be60b3c7e9724416c4cea8b148889b9066458a144c7
contracts/Lock/LockV2.sol	e1d87db983cd4f5442e160f920494652e58c5d1db21aea9aa977d036c000aa8c
contracts/Lock/FullMath.sol	cf33688bcc87ed97503be8c33c733686b15469e83b15e4d9969f9f0ed4f04fcc

Overview

The GemPad `GempadLock` contract offers a versatile locking mechanism for both v2 and v3 liquidity pool tokens, as well as normal tokens, providing users with a wide array of functionalities to lock, unlock, and manage their assets. This comprehensive audit evaluates the contract's security, examines its business logic, and identifies potential improvements, ensuring the contract's robustness, efficiency, and the safety of its users in handling various types of token locks.

Function multipleLock

This function enables multiple users to lock tokens in the contract. Users specify the token to be locked, whether it's a liquidity pool token, the amounts to be locked, and the unlock date. The function also handles fee deductions for users who are not excluded from fees, with the fee amount varying based on the type of token being locked. It ensures that the number of owners matches the number of amounts specified and that the unlock date is set in the future. The function ultimately calls an internal function to process the multiple locks, handling token transfers and lock registrations.

Function multipleVestingLock

Similar to multipleLock, this function allows multiple users to lock tokens, but with additional vesting parameters. It includes `tgeDate`, `tgeBps`, `cycle`, and `cycleBps` to define the vesting schedule. The function also incorporates a fee mechanism for non-exempt users and performs checks to ensure the validity of the token, the vesting cycle, and the percentage of tokens released at the Token Generation Event (TGE) and each cycle. The function concludes by calling an internal function to execute the vesting locks, ensuring the correct handling of token transfers and lock registration with vesting conditions.

Function lockLpV3

This function is specifically designed for locking v3 liquidity pool positions. It allows a user to lock an NFT representing a liquidity position in a Uniswap v3 pool. The function checks for fee requirements, the validity of the NFT manager, and the future unlock date. It also verifies the project token's involvement in the liquidity pool and ensures the pool's

existence. Upon successful validation, the function locks the NFT position, updates cumulative lock information, and transfers the NFT from the user to the contract. This lock is particularly tailored for DeFi scenarios involving liquidity positions in Uniswap v3 pools.

Function unlock

This function allows users to unlock their locked tokens by specifying the lock ID. It first verifies that the caller is the owner of the lock and then determines the unlocking mechanism based on the `tgeBps` value of the lock. If `tgeBps` is greater than zero, indicating a vesting schedule, the function calls `_vestingUnlock` to handle the unlock process. Otherwise, it calls `_normalUnlock` for standard unlocks without vesting. The function ensures that the unlock conditions are met, such as the current time being past the unlock date, and then proceeds to transfer the tokens back to the user, updating the lock's state accordingly.

Function unlockAllAvailable

This function is designed for users who wish to unlock all their available locks in one transaction. It iterates through all the locks associated with the caller's address, both liquidity pool (LP) and normal locks. For each lock, it checks the `tgeBps` value to determine the appropriate unlocking mechanism, similar to the unlock function. The function then either calls `_vestingUnlock` for vesting locks or `_normalUnlock` for standard locks. This batch process allows users to efficiently unlock multiple locks without the need to individually specify each lock ID.

Function editLock

The editLock function provides users with the flexibility to modify certain aspects of their existing locks. It allows the lock owner to increase the amount locked or extend the unlock date. The function first ensures that the caller is the owner of the lock and that the lock has not been unlocked yet. If a new unlock date is provided, the function checks that it is both after the current time and the original unlock date, ensuring the lock's time extension is valid. For normal locks (non-NFT based), the function permits the addition of more tokens to the lock. This is achieved by updating the lock's amount and the cumulative lock information, followed by transferring the additional tokens from the user to the contract. The function ensures the exact amount of tokens is transferred and updates the lock's details

accordingly. This feature is particularly useful for users who wish to increase their stake or extend their commitment period in the locking contract.

Function increaseLiquidityCurrentRange

This function is designed for users who wish to increase the liquidity of their existing v3 liquidity pool (LP) locks. It allows the lock owner to add more liquidity to their current position by specifying the lock ID and the amounts of the two tokens to be added. The function first ensures that the lock is a v3 LP lock and that the caller is the owner of the lock. It also checks that the lock has not been unlocked yet. The function then transfers the specified amounts of the two tokens from the user to the contract and approves the NFT manager to use these tokens. It calls the `increaseLiquidity` function of the NFT manager to add liquidity to the position, updating the lock's amount and the cumulative lock information accordingly. This function is essential for users looking to enhance their position in a liquidity pool while maintaining the lock.

Function decreaseLiquidityCurrentRange

This function enables users to decrease the liquidity of their v3 LP locks. Users specify the lock ID and the amount of liquidity they wish to remove. The function verifies that the lock is a v3 LP lock, the caller is the owner, and the lock's unlock date has passed. It also checks that the lock's amount is sufficient for the requested liquidity decrease and that the lock has not been unlocked yet. The function then calls the decreaseLiquidity function of the NFT manager to remove the specified liquidity, transferring the corresponding amounts of the two tokens back to the user. It updates the lock's amount and the cumulative lock information, reflecting the decreased position. This function is crucial for users who need to partially withdraw their stake from a liquidity pool while keeping the remaining position locked.

Function collectFees

This function allows users to collect accumulated fees from their v3 liquidity pool (LP) locks without unlocking the position. Users specify the lock ID, and the function ensures that the lock is a v3 LP lock and that the caller is the owner of the lock. It then sets the maximum amounts for both tokens to collect all fees and calls the collect function of the NFT manager. The collected fees are transferred to the contract and then safely sent to the lock

owner. This function is particularly useful for LP lock owners who want to harvest their earned fees while keeping their liquidity position intact in the pool.

Function editLockDescription

This function provides the ability for users to update the description of their existing locks. Users specify the lock ID and the new description. The function verifies that the caller is the owner of the lock and then updates the lock's description with the provided new description. This feature is beneficial for users who need to modify the details or notes associated with their locks, allowing for better management and organization of their locked assets.

Function editProjectTokenMetaData

This function enables project owners to update the metadata associated with their project tokens. Users specify the token address and the new metadata. The function checks that the caller is the owner of the project token and then updates the project's metadata with the new information. This functionality is essential for project owners who need to keep their token information up-to-date. It helps maintain accurate and current information about the project tokens within the contract ecosystem.

Function transferProjectOwnerShip

This function allows the current owner of a project to transfer ownership to a new owner. The user specifies the token associated with the project and the address of the new owner. The function first verifies that the caller is indeed the current owner of the project. Upon validation, it updates the project's ownership to the new owner. This functionality is crucial for scenarios where project ownership needs to be reassigned. The transfer of ownership ensures continuity in project management and control.

Function transferLockOwnership

This function enables users to transfer the ownership of a specific lock to a new owner. Users provide the lock ID and the address of the new owner. The function checks that the caller is the current owner of the lock and then updates the lock's ownership to the new owner. It also updates the user lock IDs for both the current and new owners, ensuring that the lock is correctly associated with the new owner. This feature is particularly useful for

users who need to transfer their locked positions to another party. The ability to transfer lock ownership adds flexibility and utility to the locking mechanism, allowing for dynamic management of locked assets.

Roles

Owner

The owner has the authority to initialize the LockV2 contract. The owner is responsible for setting up various fee variables and excluding specific addresses from these fees.

The owner can interact with the following functions:

- function updateAvailabilityForNFT
- function updateFee
- function excludeFromFee

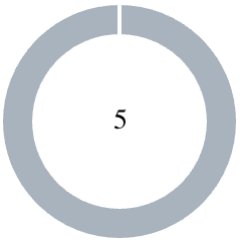
Users

The users can interact with the following functions:

- function multipleLock
- function multipleVestingLock
- function lockLpV3
- function unlock
- function unlockAllAvailable
- function withdrawableTokens
- function editLock
- function increaseLiquidityCurrentRange
- function decreaseLiquidityCurrentRange
- function collectFees
- function editLockDescription
- function editProjectTokenMetaData
- function transferProjectOwnerShip
- function transferLockOwnership
- function getTotalLockCount
- function getLockAt
- function allLpTokenLockedCount
- function allNormalTokenLockedCount
- function getCumulativeLpTokenLockInfoAt
- function getCumulativeNormalTokenLockInfoAt

- function lpLockCountForUser
- function lpLockForUserAtIndex
- function normalLockCountForUser
- function normalLockForUserAtIndex
- function totalLockCountForToken
- function getLocksForToken
- function getProject

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	5

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	5	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CCR	Contract Centralization Risk	Unresolved
	ELFM	Exceeds Fees Limit	Unresolved
	MOA	Misleading Owner Assignment	Unresolved
	MU	Modifiers Usage	Unresolved
	L19	Stable Compiler Version	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	GempadLock.sol#L138
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

The functionality and behavior of this contract are significantly influenced by the ability of the owner to modify settings, particularly in terms of NFT availability. The contract code allows the owner to accurately set the availability of specific NFT contract addresses and utilize the appropriate contracts, ensuring that the owner has the necessary control to manage the platform effectively. This flexibility, while beneficial, introduces several risks related to centralization. These risks include a Single Point of Control, as the `onlyOwner` modifier grants exclusive rights to the contract owner to change the available NFT addresses that will be used. It also increases vulnerability to attacks if the owner's account is compromised. Operational Delays could occur since changes depend solely on the owner's actions. Trust Dependencies are heightened, as users must trust the owner to manage NFT availability responsibly.

```
   function updateAvailabilityForNFT(address nft, bool isAvailable) external
onlyOwner {
        require(isAvailableNFT[nft] != isAvailable, "the same");
        isAvailableNFT[nft] = isAvailable;
        emit NFTAvailableUpdated(nft, isAvailable);
    }
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	LockV2.sol#L144
Status	Unresolved

Description

The contract is designed to allow the contract owner to update various fees through the `updateFee` function. This function accepts the five parameters, `projectCreationFee`, `lpTokenNormalLockFee`, `lpTokenVestingLockFee`, `normalTokenNormalLockFee`, and `normalTokenVestingLockFee`. The `onlyOwner` modifier ensures that only the contract owner can call this function. However, there are no checks or limitations on the values that can be set for these fees. This lack of restrictions means the contract owner has the authority to set the fees to any number, potentially leading to scenarios where the fees could be set to excessively high or arbitrary amounts. This could result in unreasonable costs for users interacting with the contract and may lead to trust issues or misuse of the contract.

```
    function updateFee(
        uint256 projectCreationFee,
        uint256 lpTokenNormalLockFee,
        uint256 lpTokenVestingLockFee,
        uint256 normalTokenNormalLockFee,
        uint256 normalTokenVestingLockFee
    ) external onlyOwner {
        fee = Fee({
            projectCreationFee: projectCreationFee,
            lpTokenNormalLockFee: lpTokenNormalLockFee,
            lpTokenVestingLockFee: lpTokenVestingLockFee,
            normalTokenNormalLockFee: normalTokenNormalLockFee,
            normalTokenVestingLockFee: normalTokenVestingLockFee
        });

        emit FeeUpdated(fee);
    }
```

Recommendation

It is recommended to introduce a maximum limit for each fee parameter within the `updateFee` function. Implementing a cap on the fee values would prevent the contract owner from setting unreasonably high fees, thereby protecting users from potential exploitation. This can be achieved by adding require statements that check if the fee values are within a predefined acceptable range.

MOA - Misleading Owner Assignment

Criticality	Minor / Informative
Location	LockV2.sol#L324,514,554
Status	Unresolved

Description

The contract includes the `lockLpV3` , `_lockLpToken` , and `_lockNormalToken` functions, which are designed to handle different types of token locks. A common issue across these functions is the assignment of the project owner. The contract sets the `msg.sender` who initiates the first lock as the project owner. This approach assumes that the first user to lock tokens is the rightful project owner, which might not always be true. As a result, the contract could incorrectly assign project ownership, leading to potential misrepresentation and management issues. This misalignment between the actual project owner and the owner recorded in the contract could create confusion and governance problems.

```
  function lockLpV3(
        address _owner,
        address nftManager,
        uint256 nftId,
        uint256 unlockDate,
        string memory description,
        string memory _metaData,
        address projectToken,
        address referrer
    ) external payable override returns (uint256 id) {
        ...

        CumulativeLockInfo storage tokenInfo = cumulativeLockInfo[token];
        if (tokenInfo.projectToken == address(0)) {
            tokenInfo.projectToken = projectToken;
            tokenInfo.factory = factory;
        } else {
            projectToken = tokenInfo.projectToken;
        }
        ...
        }

  function _lockLpToken(
        ...
    ) private returns (uint256 id) {
        ...
        if (project.owner == address(0)) {
            project.owner = msg.sender;
            project.metaData = _metaData;
        }
        project.lpLockedTokens.add(token);
    }

    function _lockNormalToken(
       ...
    ) private returns (uint256 id) {
        ...

        Project storage project = projects[token];
        if (project.owner == address(0)) {
            project.owner = msg.sender;
            project.metaData = _metaData;
        }
    }
```

Recommendation

It is recommended to revise the contract's approach to assigning project ownership in these functions. A more reliable and verifiable method should be implemented to ensure that project ownership is accurately assigned to the legitimate owner. This could involve a dedicated ownership claim process, where project owners explicitly assert their ownership, or a verification mechanism to confirm the legitimacy of the `msg.sender` as the project owner. By adopting a more precise and secure method for determining project ownership, the contract can avoid misleading representations and enhance its overall reliability and governance structure.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	LockV2.sol#L632,774,809,871
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(userLock.unlockedAmount == 0, "Nothing to unlock");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	interfaces/IGempadLock.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.22;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

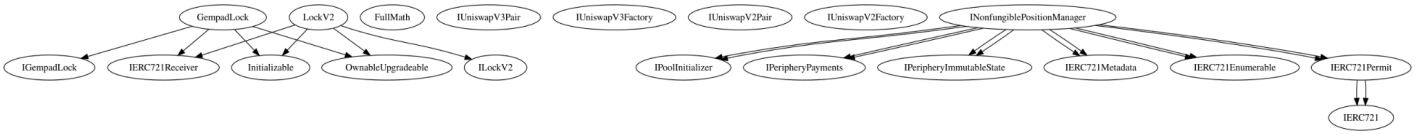
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
GempadLock	Implementation	IGempadLock, IERC721Receiver, Initializable, OwnableUpgradeable		
	initialize	Public	✓	initializer
	updateAvailabilityForNFT	External	✓	onlyOwner
	updateFee	External	✓	onlyOwner
	excludeFromFee	External	✓	onlyOwner
	_payFee	Internal	✓	
	multipleLock	External	Payable	-
	multipleVestingLock	External	Payable	-
	lockLpV3	External	Payable	validNFT
	_multipleLock	Internal	✓	
	_sumAmount	Internal		
	_createLock	Internal	✓	
	_lockLpToken	Private	✓	
	_lockNormalToken	Private	✓	
	_registerLock	Private	✓	
	unlock	External	✓	validLock isLockOwner
	unlockAllAvailable	External	✓	-

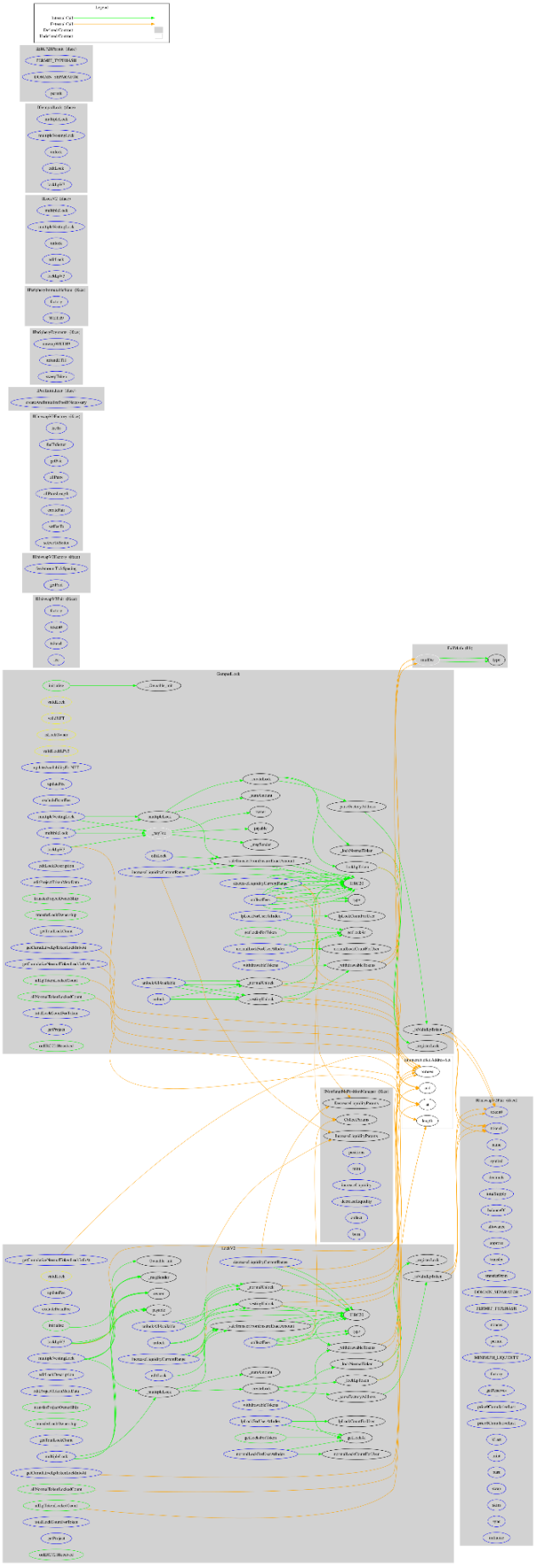
	_normalUnlock	Internal	✓	
	_vestingUnlock	Internal	✓	
	withdrawableTokens	External		-
	_withdrawableTokens	Internal		
	editLock	External	✓	validLock isLockOwner
	increaseLiquidityCurrentRange	External	✓	validLock isLockOwner validLockLPv3
	decreaseLiquidityCurrentRange	External	✓	validLock isLockOwner validLockLPv3
	collectFees	External	✓	isLockOwner validLockLPv3
	editLockDescription	External	✓	validLock isLockOwner
	editProjectTokenMetaData	External	✓	-
	transferProjectOwnerShip	Public	✓	-
	transferLockOwnership	Public	✓	validLock isLockOwner
	_safeTransferFromEnsureExactAmount	Internal	✓	
	getTotalLockCount	External		-
	getLockAt	Public		-
	allLpTokenLockedCount	Public		-
	allNormalTokenLockedCount	Public		-
	getCumulativeLpTokenLockInfoAt	External		-
	getCumulativeNormalTokenLockInfoAt	External		-
	lpLockCountForUser	Public		-
	lpLockForUserAtIndex	External		-

	normalLockCountForUser	Public		-
	normalLockForUserAtIndex	External		-
	totalLockCountForToken	External		-
	getLocksForToken	Public		-
	_parseFactoryAddress	Internal		
	_isValidLpToken	Private		
	getProject	External		-
	onERC721Received	Public	✓	-

Inheritance Graph



Flow Graph



Summary

GemPad Lockers contract implements a locker mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io