Cyberscope

Audit Report
Rocket Protocol Liquidity Locker

August 2024

Network BSC

Address 0x99090d2d220901De904c6E3d003D7ceD4B6eC2A4

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	UniLockerV3LP
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	20000 runs
Explorer	https://bscscan.com/token/0x99090d2d220901De904c6E3d003D7ceD4B6eC2A4
Address	0x99090d2d220901de904c6e3d003d7ced4b6ec2a4
Network	BSC
Symbol	UL-V3LP

Audit Updates

Initial Audit	27 Aug 2024

Source Files

Filename	SHA256
UniLockerV3LP.sol	3e7d599a00c2855e53ad88d361a2fd60c50194a4d11c6ca67c8684b7f8e5749d
IUniLocker.sol	522dd5e5c7337f9a25b038ec534e1d3e4704a101673f22b9b154269137ebb04f
AbstractUniLocker.sol	8dfb5fcdb32a6e0e47d9a8b560459f7aa4ba93a6811b647e5fd97dc9dccc2f5d
uniswap/INonfungiblePositionManager.sol	86283f552cf94052c790c49738d0d80a915d28b4ba768058ca4429494f53ac1c

Overview

The contract is designed to facilitate the locking of Uniswap V3 liquidity positions, represented as NFTs, within a decentralized finance (DeFi) environment. By locking these NFTs, users effectively deposit their liquidity positions into the contract. The contract also includes functionality for managing fees and allows the owner to set the recipient of these fees, promoting secure and structured management of liquidity assets.

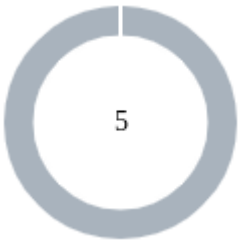
Lock Functionality

The `lock` function allows users to lock their Uniswap V3 liquidity position NFTs into the contract by transferring the NFT to the contract's custody and setting a future block number as the unlock time. Upon locking, the user is issued a new ERC721 token representing the locked position. This token serves as proof of ownership and can potentially be used for other purposes within the ecosystem. The locked NFT remains inaccessible, ensuring that users cannot prematurely withdraw their liquidity position, thereby providing a secure and reliable locking mechanism.

ClaimProfit Functionality

The `claimProfit` function allows the owner of a locked liquidity position NFT to collect the profits generated from their Uniswap V3 liquidity position while it remains locked. The function calculates the amount of tokens accrued as profit, deducts a fee based on the predefined rate, and then transfers the remaining tokens back to the owner. This mechanism ensures that users can still benefit from their liquidity positions' earnings even while the positions are locked within the contract, while also compensating the contract owner or a designated fee recipient.

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
●	MEE	Missing Events Emission	Unresolved
●	NUM	NFT Unlocking Missing	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	AbstractUniLocker.sol#L40
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
    function setFeeTo(address _feeTo) external onlyOwner {
        feeTo = _feeTo;
    }
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

NUM - NFT Unlocking Missing

Criticality	Minor / Informative
Location	AbstractUniLocker.sol#L43
Status	Unresolved

Description

The contract is designed to lock Uniswap V3 liquidity position NFTs by transferring them from the `msg.sender` to the contract and setting an `unlockBlock` time, with the intent to allow the NFT to be unlocked after the specified block time has passed. However, the contract lacks any functionality to actually unlock and retrieve the locked NFTs. This poses a significant risk, particularly if a user selects an `unlockBlock` that is set far into the future, effectively locking the NFT forever without the possibility of retrieval. Alternatively, if the intended functionality is to lock NFTs permanently, there is no need to utilize the `unlockBlock` parameter, as it serves no practical purpose in such a scenario.

```
    function lock(
        address lpToken,
        uint256 amountOrId,
        uint256 unlockBlock
    ) public override returns (uint256 id) {
        require(
            unlockBlock > block.number,
            "UniLocker: unlockBlock must be in the future"
        );
       ...

        _transferLP(lpToken, msg.sender, address(this), amountOrId);

        uint256 tokenId = _tokenIDTracker++;
        _mint(msg.sender, tokenId);
        lockItems[tokenId] = LockItem(lpToken, amountOrId, unlockBlock);

        emit Lock(lpToken, tokenId, amountOrId, unlockBlock,
msg.sender);
        return tokenId;
    }
```

Recommendation

It is recommended to clarify the intended functionality regarding the locking and unlocking of NFTs. If the ability to unlock NFTs after a specified block time is desired, the contract should include a mechanism for users to retrieve their NFTs once the `unlockBlock` has passed. On the other hand, if the goal is to lock NFTs permanently, the contract should remove the `unlockBlock` parameter to avoid confusion and simplify the logic. This will ensure that the contract operates as intended and that users are fully aware of the implications when locking their NFTs.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	UniLockerV3LP.sol#L25 AbstractUniLocker.sol#L39
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint256 _id
address _feeTo
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	UniLockerV3LP.sol#L20,22 AbstractUniLocker.sol#L40
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
feeTo = _feeTo
positionManager = _positionManager
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	UniLockerV3LP.sol#L2 AbstractUniLocker.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 ^0.8.24;
```

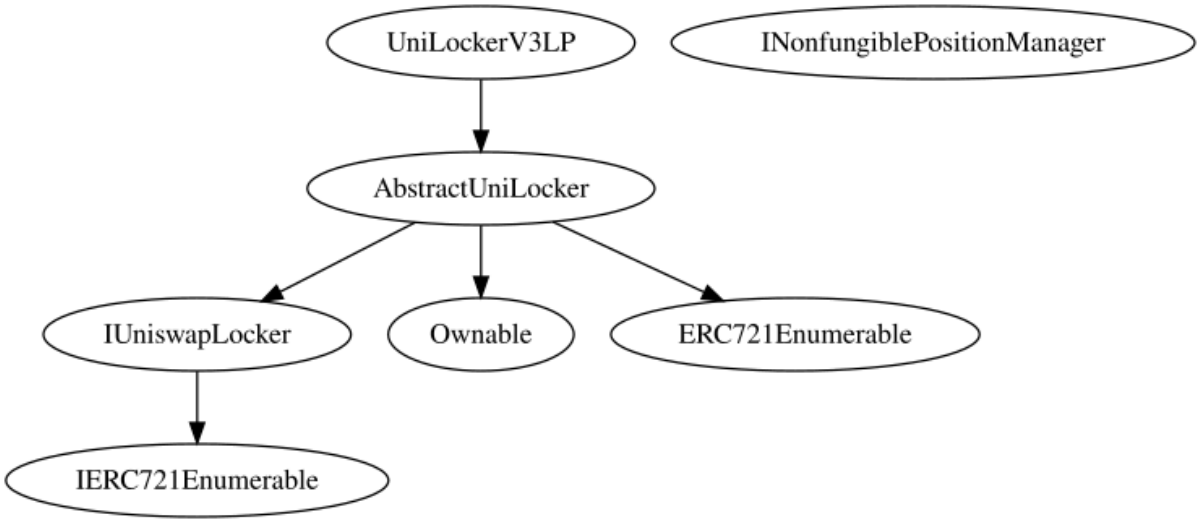
Recommendation

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.

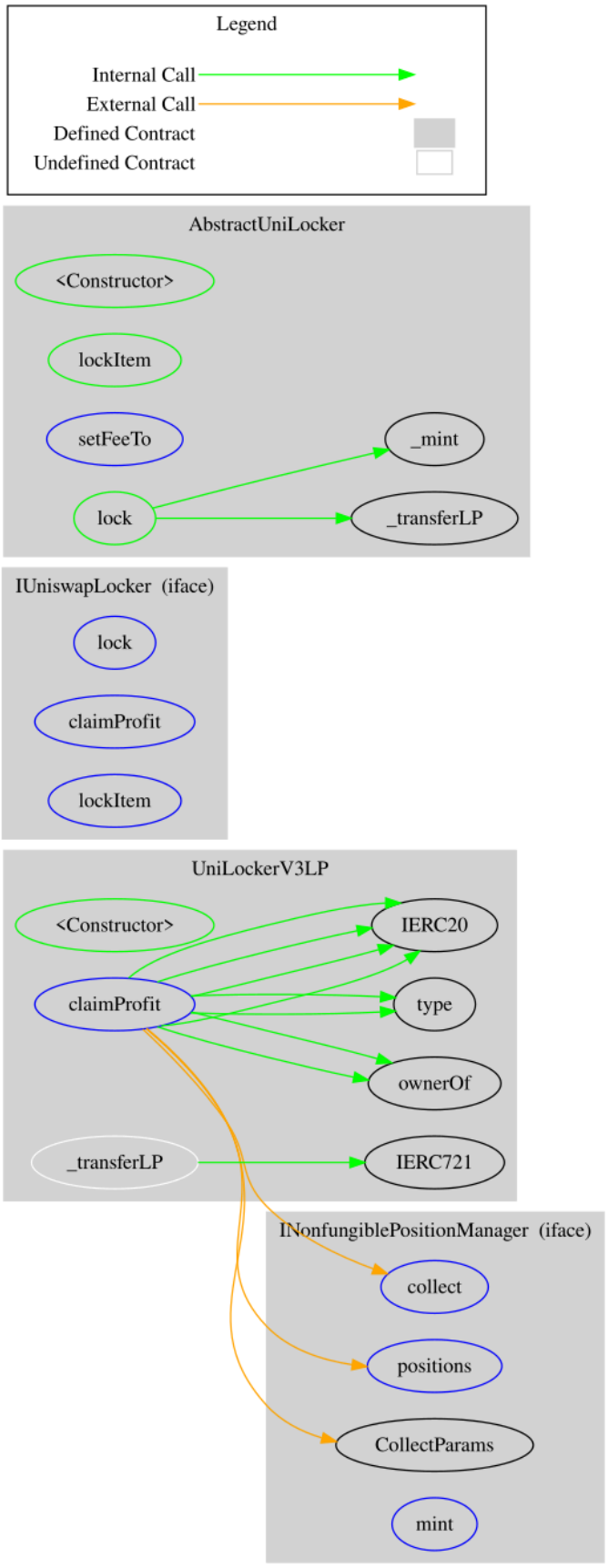
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
UniLockerV3LP	Implementation	AbstractUniLocker		
		Public	✓	AbstractUniLocker
	claimProfit	External	✓	-
	_transferLP	Internal	✓	
AbstractUniLocker	Implementation	IUniswapLocker, Ownable, ERC721Enumerable		
		Public	✓	ERC721 Ownable
	lockItem	Public		-
	setFeeTo	External	✓	onlyOwner
	lock	Public	✓	-
	_transferLP	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
AbstractUniLocker
<Constructor>
lockItem
setFeeTo
_mint
lock
_transferLP
IUniswapLocker (iface)
lock
claimProfit
lockItem
UniLockerV3LP
<Constructor>
IERC20
claimProfit
type
ownerOf
_transferLP
IERC721
INonfungiblePositionManager (iface)
collect
positions
CollectParams
mint

Summary

The UniLockerV3LP contract implements a locker mechanism for Uniswap V3 liquidity positions, allowing users to lock their NFTs on the contract securely. This audit investigates security vulnerabilities, examines the accuracy and robustness of the business logic, and identifies potential areas for improvements to enhance the contract's functionality and user safety.

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

cyberscope.io