Unicrypt

Uniswap LP Token Locking Contract v2

SMART CONTRACT AUDIT

17.12.2020

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Unicrypt.network. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (09.12.2020)	Layout and details (Metrics / Scope of work)
0.5 (10.12.2020)	Automated Security Testing Manual Security Testing
0.8 (11.12.2020)	Adding of SWC, Special Checks
1.0 (12.12.2020)	Final document (Summary and Recommendation)
1.5 (16.12.2020)	Fixed issues
1.6 (17.12.2020)	Added deployed contract address

2. About the Project and Company

Company address: NA (ANON)

Website: https://unicrypt.network/

GitHub: NA

Twitter: https://twitter.com/UNCX_token

Telegram: https://t.me/uncx_token

Etherscan (UNCX Token): https://etherscan.io/token/0xaDB2437e6F65682B85F814fBc12FeC0508A7B1D0

Medium: https://unicrypt.medium.com/

2.1 Project Overview

The Unicrypt platform allows yield farming virtually any ERC20 token. It provides safe vault contracts for other tokens to deposit the farm rewards into, and a dApp thats targeted for mobile and desktop use with connections to all major wallets for users to farm their favourite tokens on. Uncrypt is one of the major platforms for Proof of liquidity, which helps users find new pairs on uniswap that have locked their liquidity (Uniswap LP Token). This means it is impossible for that liquidity to be pulled until the unlock date expires. For taking part in this program, tokens are awarded a trust score, and are highly visible to investors searching on the platform for new tokens.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts

1. SafeMath.sol (0.6.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol
2. Ownable.sol (0.6.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol
3. Context.sol (0.6.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/GSN/Context.sol
4. EnumerableSet.sol (0.6.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/EnumerableSet.sol

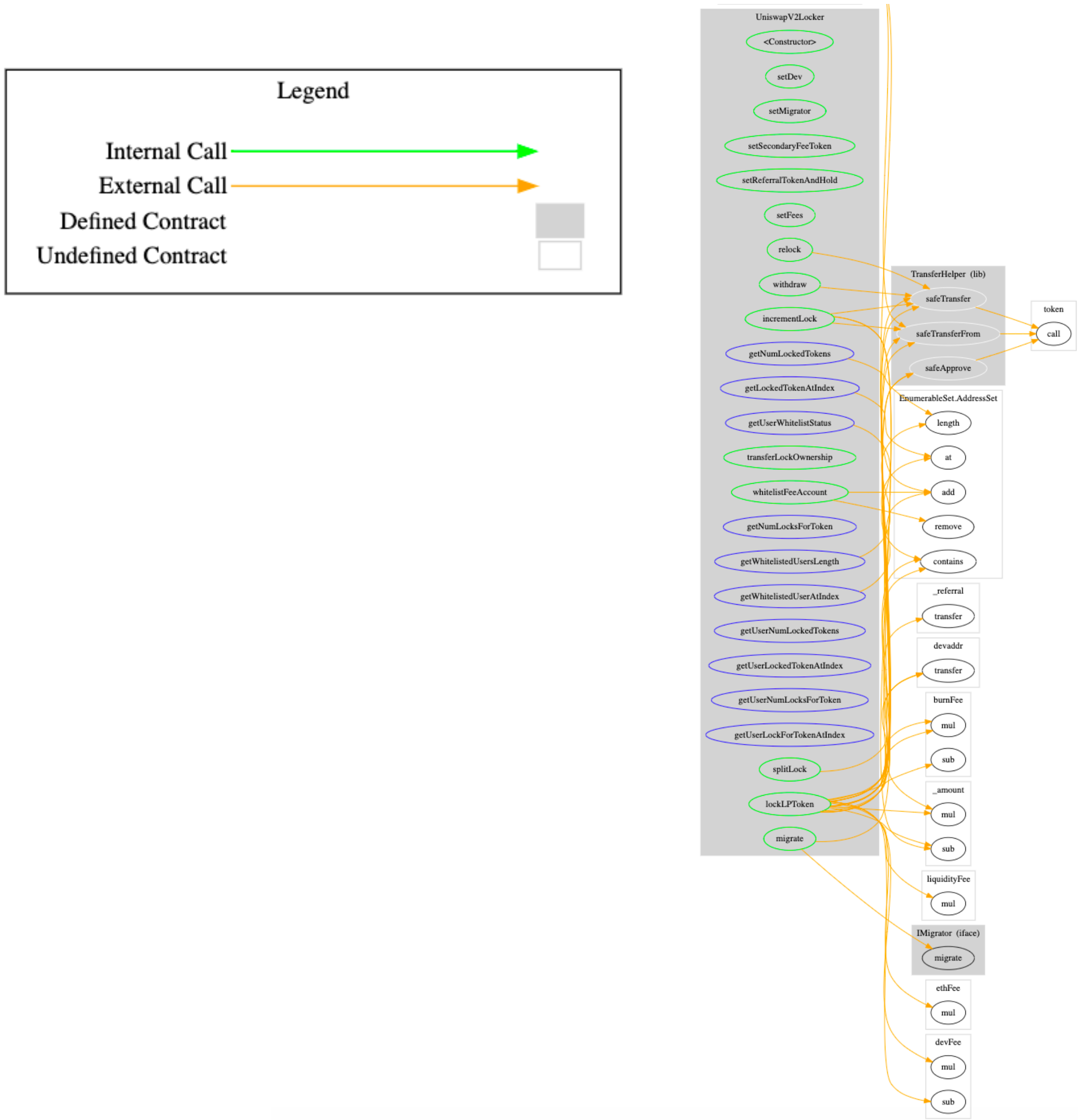
5. TransferHelper (0.6.0)
https://github.com/Uniswap/uniswap-lib/blob/master/contracts/libraries/TransferHelper.sol

4.3 Tested Contract Files

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

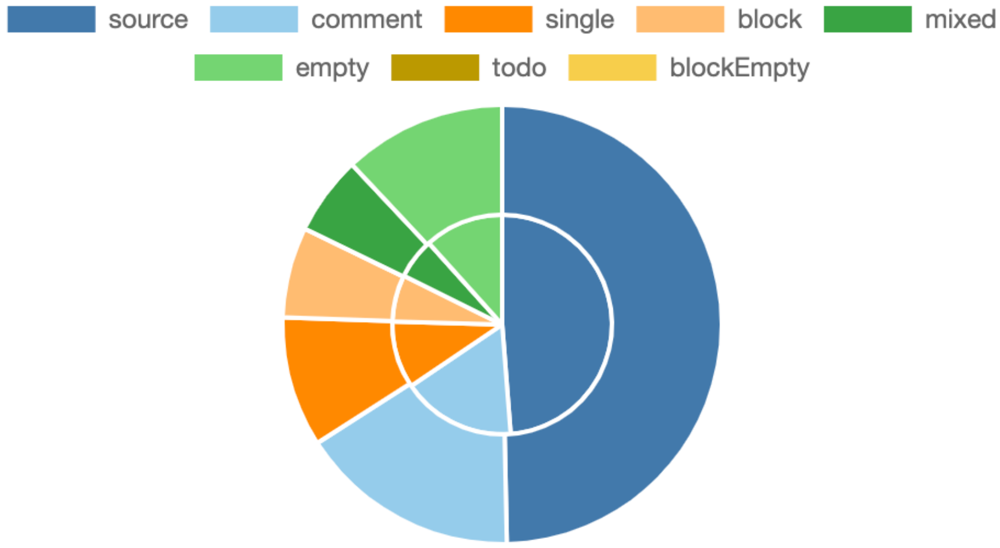
File	Fingerprint (SHA256)
UniswapV2Locker.sol	D10E293CD960B70FF407FED61717144FCBACFA57ADBAC11F11C5373533E1B471

4.4 Metrics / CallGraph



4.5 Metrics / Source Lines

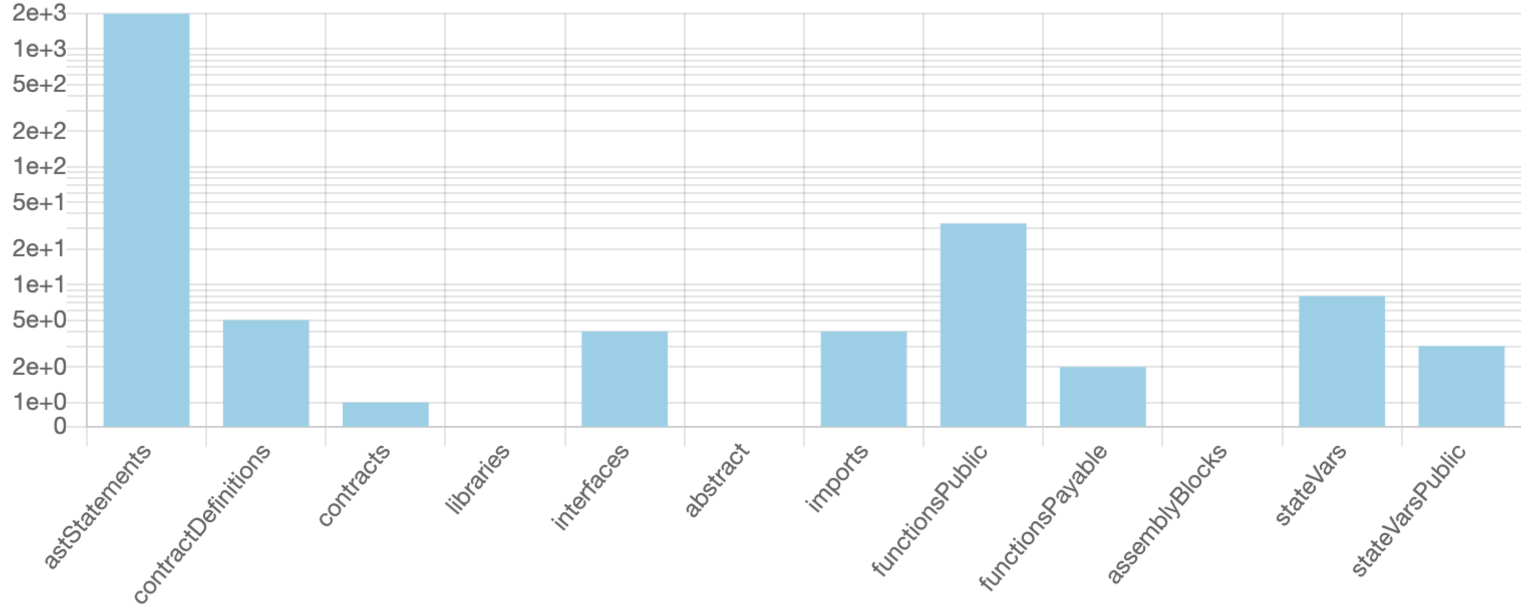
Source Lines (sloc vs. nsloc)



4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`0.6.12`		`yes`	`no` (0 asm blocks)	`no`

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`	`no`	`no`	`no`	`no`	`no`



4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝🔍	contracts-v2/UniswapV2Locker.sol	1	4	421	403	95	218	💰📤🌞
📝🔍	**Totals**	**1**	**4**	**421**	**403**	**95**	**218**	💰📤🌞

5. Scope of Work & Results

The UniCrypt team provided us with the files that needs to be tested. The scope of the audit is UniswapV2Locker.sol contract with its direct imports.

The team put forward the following assumptions regarding the security of the UniswapV2Locker.sol Audit contract:

- After locking period ends, tokens are possible to withdraw
- Unicrypt (Deployer) is not able to withdraw/ steal locked tokens
- Everything is working as it supposed to be

The main goal of this audit was to verify these claims and check the overall security of the codebase.

Old Unicrypt V1 Locker Contract:
https://etherscan.io/address/0x17e00383a843a9922bca3b280c0ade9f8ba48449#code

5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

5.1.1 Wrong import of OpenZeppelin library

Severity: MEDIUM
Status: FIXED
File(s) affected: UniswapV2Locker.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation, OpenZeppelin files are added to the repo. This violates OpenZeppelin's MIT license, which requires the license and copyright notice to be included if its code is used. Moreover, updating code manually is error-prone.	NA	We highly recommend using npm (import "@openzeppelin/contracts/..) in order to guarantee that original OpenZeppelin contracts are used with no modifications. This also allows for any bug-fixes to be easily integrated into the codebase.

LOW ISSUES

5.1.2 Checks-effects-interactions pattern
Severity: LOW
Status: FIXED
File(s) affected: UniswapV2Locker.sol

Attack / Description	Code Snippet	Result/Recommendation

Potential violation of Checks-Effects-Interaction pattern in UniswapV2Locker.lockLPToken(address,uint256,uint256,address payable,bool,address payable): Could potentially lead to re-entrancy vulnerability.	`Line: 145`
	
	`function lockLPToken (address _lpToken, uint256 _amount, uint256 _unlock_date, address payable _referral, bool _fee_in_eth, address payable _withdrawer) public payable {`
		OpenZeppelin has it's own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex.
		View the source code for the OpenZeppelin ReentrancyGuard library here: https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/utils/ReentrancyGuard.sol
		Keep in mind that a nonReentrant function should be external. If another function calls the nonReentrant function it is no longer protected.
		
		`function lockLPToken (address _lpToken, uint256 _amount, uint256 _unlock_date, address payable _referral, bool _fee_in_eth, address payable _withdrawer) public payable nonReentrant {`

INFORMATIONAL ISSUES

5.1.3 Fix Spelling and Grammatical Errors

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: UniswapV2Locker.sol

Attack / Description	Code Snippet	Result/Recommendation

Language mistakes were identified in the messages in the codebase. Fixing these mistakes can help improve the end-user experience by providing clear information on errors encountered, and improve the maintainability and auditability of the codebase.	Line: 167
	`require(msg.value == ethFee, 'Fee not met');`
	
	Line 249:
	`require(_amount > 0, 'Zero withdrawl');`
		Keep the capitalization of letters consistent
		`require(msg.value == ethFee, 'FEE NOT MET');`
		
		Fix spelling error
		`require(_amount > 0, 'ZERO WITHDRAWAL');`

5.2. SWC Attacks & Special Checks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✅
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✅
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✅
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✅
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✅
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✅
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✅
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✅
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✅
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✅
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✅
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✅
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✅
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✅

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✅
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✅
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✅
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅
1	After locking period ends, tokens are possible to withdraw	We deployed the contract in our test network and tried the withdrawal after the locking period ends. Result: it was possible	✅
2	Unicrypt (Deployer) is not able to withdraw/ steal locked tokens	We deployed the contract in our test network and tried to steal locked token funds with the deployer address. Result: it was not possible	✅

6. Executive Summary

The smart contract are written as simple as possible and also not overloaded with unnecessary functions, these is greatly benefiting the security of the contract. It correctly implemented widely-used and reviewed contracts from OpenZeppelin and for safe mathematical operations. The main goal of the audit was to verify the claims regarding the security of the smart contract (see the scope of work section). According to the code, the implementation of the locking functions consider all security checks for a safe locking and withdrawal of UniswapV2Pair Token.
Both claims appear valid. During the audit, no critical or high issues were found after the manual and automated security testing.

Edit: The Unicrypt Team reacted promptly on our findings and fixed all bugs.

7. Deployed Smart Contract

Deployed Unicrypt Contracts (Mainnet)

UniswapV2Locker.sol

https://etherscan.io/address/0x663A5C229c09b049E36dCc11a9B0d4a8Eb9db214#code (approved)