Unicrypt

Uniswap LP Token Locker v3

SMART CONTRACT AUDIT

21.02.2022

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer 4
2. About the Project and Company 5
   2.1 Project Overview 6
3. Vulnerability & Risk Level 7
4. Auditing Strategy and Techniques Applied 8
   4.1 Methodology 8
   4.2 Used Code from other Frameworks/Smart Contracts 9
   4.3 Tested Contract Files 10
   4.4 Metrics / CallGraph 11
   4.5 Metrics / Source Lines & Risk 12
   4.6 Metrics / Capabilities 13
   4.7 Metrics / Source Unites in Scope 14
5. Scope of Work 18
   5.1 Manual and Automated Vulnerability Test 19
   5.1.1 Checking for boolean equality 19
   5.1.2 Missing natspec documentation 20
   5.1.3 Missing Zero address validation 20
   5.1.4 Division before multiplication 22
   5.1.5 Unchecked transfer 23
   5.1.6 Redundant override function 23
   5.1.7 Different Compiler version used 24
   5.1.8 A floating pragma is set. 24

5.1.9 SafeMath for pragma version higher than 0.8.0 25
5.1.10 ABIEncoder v2 26
5.1.11 Public function could be declared external 27
5.1.12 Unused state variable 28
5.2. SWC Attacks 28
5.3. Verify Claims 32
6. Executive Summary 33
7. Deployed Smart Contract 33

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 SDD Tech OÜ 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 (28.10.2021)	Layout
0.5 (01.11.2021)	Verify Claims and Test Deployment
0.6 (02.11.2021)	Testing SWC Checks
0.8 (02.11.2021)	Automated Security Testing Manual Security Testing
0.9 (02.11.2021)	Summary and Recommendation
1.0 (03.11.2021)	Final Document
1.1 (10.11.2021)	Re-check
1.2 (21.02.2022)	Re-check
1.3 (TBA)	Added deployed contract

2. About the Project and Company

Company address:

SDD Tech OÜ
Mustamäe tee 6b
Tallinn Harjumaa 10616

Website: https://unicrypt.network

Twitter: https://twitter.com/UNCX_token

Telegram: https://t.me/uncx_token

Medium: https://unicrypt.medium.com

2.1 Project Overview

UniCrypt is a decentralized services provider which offers several ways for DeFi projects to build community trust and keep users safe. Famously, UniCrypt created the first-ever liquidity locking smart contracts for Uniswap on Ethereum, known as Proof-of-Liquidity or POL. From there the project continued to develop new features, combining liquidity locking with a decentralized launchpad.

Liquidity Lockers: these are smart contracts that enable teams to publicly lock liquidity on Uniswap or other AMMs for a predetermined period. Essentially, it's a guarantee to investors that the project developers can't drain the pool of all the funds. A key innovation is UniCrypt's lockers will be able to migrate liquidity to Uniswap V3 when the time comes.

FaaS: This is a yield farming-as-a-service protocol that enables the creation of a farm for any token. Launch a farm in a couple clicks using the UI, all automatic with no coding necessary.

Launchpad: Perhaps the most interesting service, a 100% decentralized and automated presale platform that is connected to the liquidity lockers. Once the presale ends a portion of the raised funds (between 30% to 100%) will create the DEX pair on a supported AMM and the liquidity will be locked.

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 (direct imports)

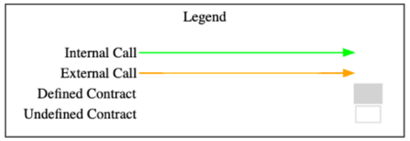
Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.0.0/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.0.0/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.0.0/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/utils/structs/EnumerableSet.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.0.0/contracts/utils/structs/EnumerableSet.sol

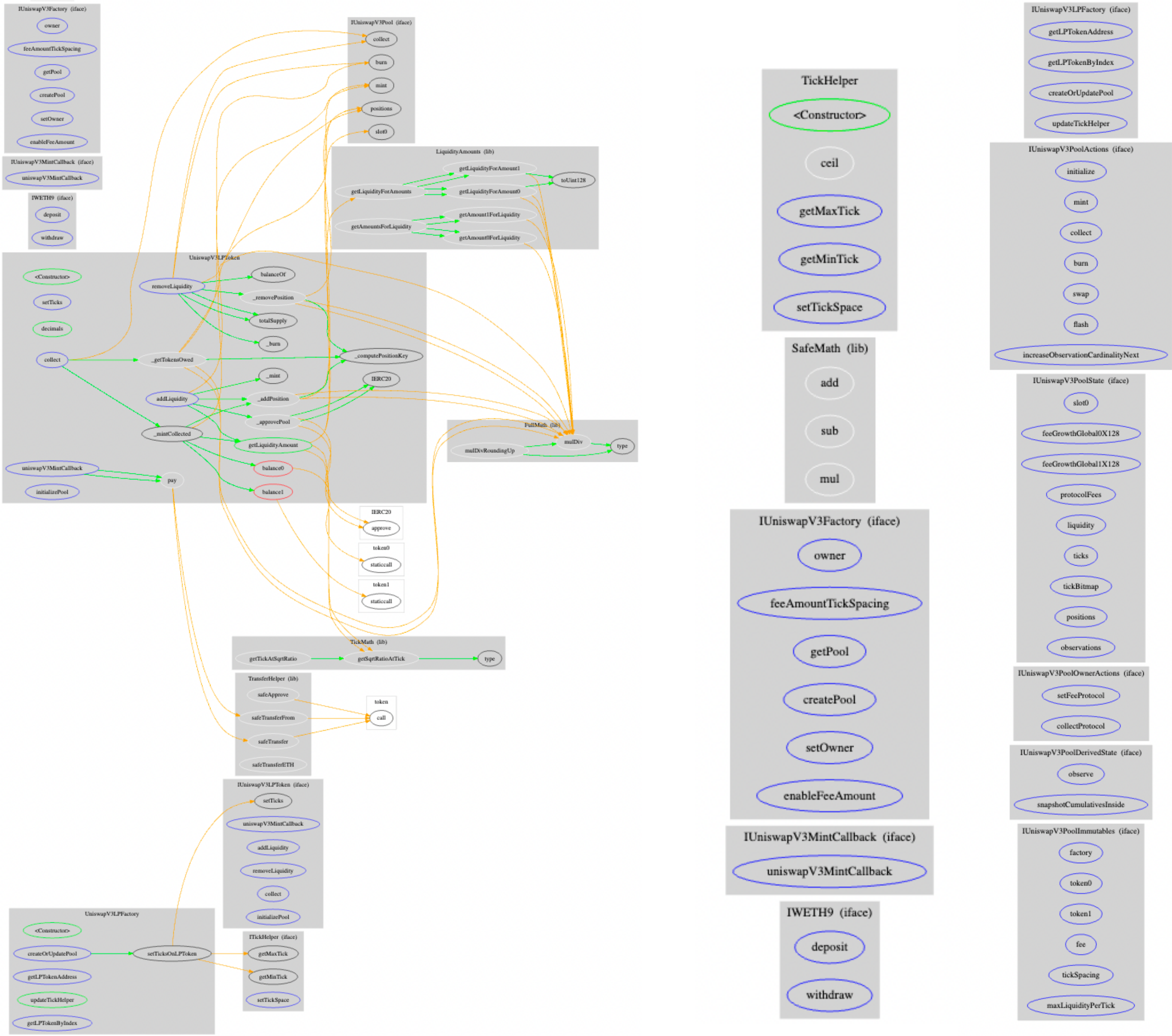
4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 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

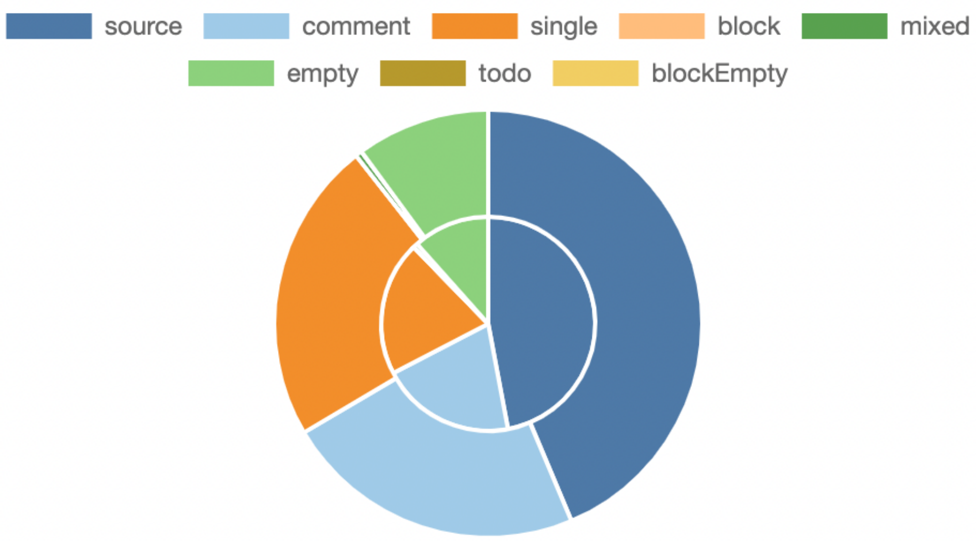
File	Fingerprint (MD5)
contracts/interfaces/pool/IUniswapV3PoolImmutables.sol	e236e09a9d654fb2f20a6da5dba2bd2f
contracts/interfaces/pool/IUniswapV3PoolDerivedState.sol	25b71180ec9f5132a158334971ee2ace
contracts/interfaces/pool/IUniswapV3PoolOwnerActions.sol	1b06ecc79e75f836c446ccf286e671e4
contracts/interfaces/pool/IUniswapV3PoolState.sol	0488495ef9087b4513d3b43634035ef9
contracts/interfaces/pool/IUniswapV3PoolEvents.sol	05abb59ec113db1046f7dadc78bb297b
contracts/interfaces/pool/IUniswapV3PoolActions.sol	83d338eb1394008c808a20ac7c5bab0c
contracts/interfaces/IUniswapV3LPFactory.sol	5f99ae71a69a0f76689d80f2921288dd
contracts/interfaces/IUniswapV3LPToken.sol	bcc1d88e0fda303a4fdb7ea42f5f6efd
contracts/interfaces/IUniswapV3Pool.sol	e6badd8268772b99e7ca397aff11a965
contracts/interfaces/ITickHelper.sol	93b1ea785db3abe5810cbaf2633287f8
contracts/interfaces/IWETH9.sol	1b896d3c1b3cb9a0b51a9b5653f393cd
contracts/interfaces/IUniswapV3MintCallback.sol	6a5f2f2fa37a7a9fc5dde34d7b037de2
contracts/interfaces/IUniswapV3Factory.sol	01639906a2fb82a249761378d373087a
contracts/libraries/FixedPoint96.sol	1efcb98c35798050bb5ad4c7cba0ca20
contracts/libraries/FullMath.sol	ff352e773255ccdcc2b63611ff6cdb49
contracts/libraries/TransferHelper.sol	35870498d775cb4b7f2802713893cfff
contracts/libraries/FixedPoint128.sol	621f45db95e839cfc3a5d1e1082bd207
contracts/libraries/TickMath.sol	2a00a4821b57c817f3c3317ac015680d
contracts/libraries/SafeMath.sol	2cbb7b0de5d5a7b798a1ac0693cc0b10
contracts/libraries/LiquidityAmounts.sol	5f9feb034fa05809431fe20f2a711835
contracts/UniswapV3LPFactory.sol	8dfc56a6706b502865961178d98cce75
contracts/TickHelper.sol	a924f2ba91a1629dfe5a5d9c1af258e5
contracts/UniswapV3LPToken.sol	989646de86a6466cae6f0e80be977f96

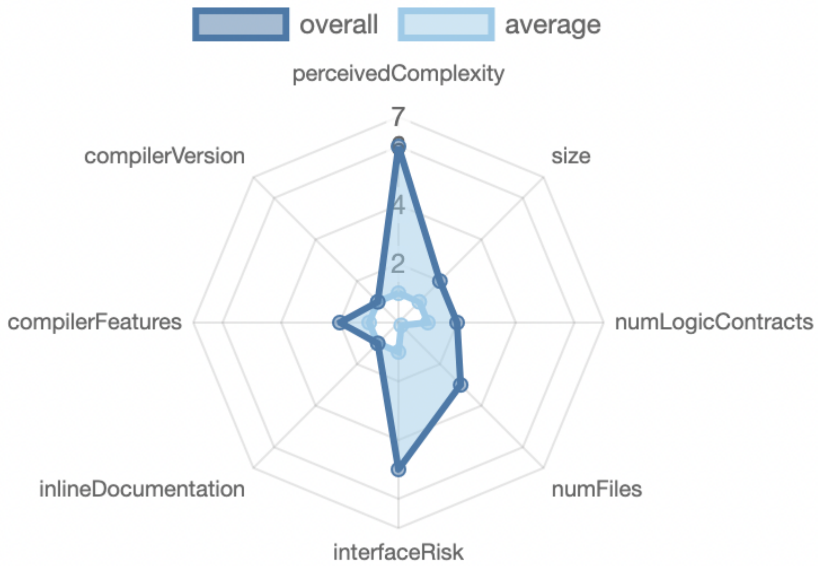
4.4 Metrics / CallGraph





4.5 Metrics / Source Lines & Risk





4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts

`>=0.5.0`
`>=0.8.0`
`>=0.7.6`
`>=0.8.3`
`>=0.4.0`
`>=0.6.0`
		`yes`	`yes`
			(29 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`			`yes`		`yes` → `NewContract:UniswapV3LPToken`

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
63	6

External	Internal	Private	Pure	View
60	67	3	15	33

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/pool/IUniswapV3PoolImmutables.sol		1	35	9	3	20	13	
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/pool/IUniswapV3PoolDerivedState.sol		1	40	18	3	23	5	
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/pool/IUniswapV3PoolOwnerActions.sol		1	23	10	3	12	5	
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/pool/IUniswapV3PoolState.sol		1	116	21	3	55	19	
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/pool/IUniswapV3PoolEvents.sol		1	121	121	52	60	1	
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/pool/IUniswapV3PoolActions.sol		1	103	15	4	59	15	

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/IUniswapV3LPFactory.sol		1	50	15	3	24	9	
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/IUniswapV3LPToken.sol		1	61	14	3	27	22	💰
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/IUniswapV3Pool.sol		1	24	24	16	5	13	
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/ITickHelper.sol		1	25	12	3	13	7	
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/IWETH9.sol		1	13	9	4	4	10	💰
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/IUniswapV3MintCallback.sol		1	18	13	3	9	3	
🔍	unicrypt-uniswap-v3-lp-master/contracts/interfaces/IUniswapV3Factory.sol		1	79	35	12	43	13	
📚	unicrypt-uniswap-v3-lp-master/contracts/libraries/FixedPoint96.sol	1		9	9	5	3	3	

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📚	unicrypt-uniswap-v3-lp-master/contracts/libraries/FullMath.sol	1		126	118	57	59	104	🖥
📚	unicrypt-uniswap-v3-lp-master/contracts/libraries/TransferHelper.sol	1		60	47	21	21	26	
📚	unicrypt-uniswap-v3-lp-master/contracts/libraries/FixedPoint128.sol	1		8	8	4	3	2	
📚	unicrypt-uniswap-v3-lp-master/contracts/libraries/TickMath.sol	1		205	205	168	23	584	🖥
📚	unicrypt-uniswap-v3-lp-master/contracts/libraries/SafeMath.sol	1		17	17	12	1	4	
📚	unicrypt-uniswap-v3-lp-master/contracts/libraries/LiquidityAmounts.sol	1		137	110	53	45	30	
📝	unicrypt-uniswap-v3-lp-master/contracts/UniswapV3LPFactory.sol	1		80	80	60		53	🌀
📝	unicrypt-uniswap-v3-lp-master/contracts/TickHelper.sol	1		48	48	36		18	
📝	unicrypt-uniswap-v3-lp-master/contracts/UniswapV3LPToken.sol	1		312	312	235	5	155	💰📤🧮

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍	**Totals**	**10**	**13**	**1710**	**1270**	**763**	**514**	**1114**	🖥💰📤🧮🌀

Legend: [➖]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

5. Scope of Work

The Unicrypt Team provided us with the files that needs to be tested. The scope of the audit is the Uniswap LP Locking v3 contracts.

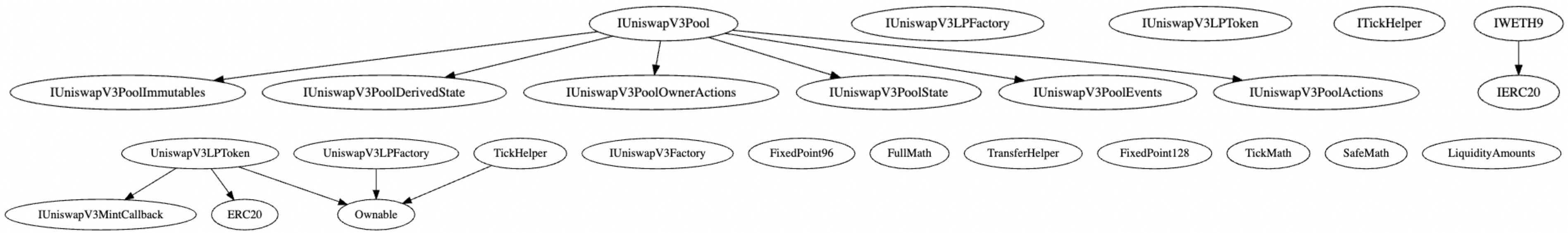
Following contracts with the direct imports has been tested:

- UniswapV3LPFactory.sol
- UniswapV3LPToken.sol

The team put forward the following assumptions regarding the security, usage of the contracts:

- Works for rebasing and highly deflationary tokens, basically any token that works in a Uniswap v3 pool
- Unicrypt (Deployer) is not able to withdraw locked LP tokens
- The smart contract is coded according to the newest standards and in a secure way.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



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

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

LOW ISSUES

5.1.1 Checking for boolean equality

Severity: LOW
Status: FIXED
Code: NA
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/3f08f36dd5670404fabfbf4226dc42f18963dd9c
File(s) affected: UniswapV3LPToken.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several require checks are using a comparison to a Boolean constant. This leads to unnecessary gas consumption.	Line 308: initializePool require(initilized == false)	It is recommended to remove the equality check to the Boolean constant and use the value itself.

5.1.2 Missing natspec documentation

Severity: LOW
Status: FIXED
Code: NA
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/4b0c306cc07508b3c889643521f41dd2ef34d2bf
File(s) affected: UniswapV3LPToken.sol, UniswapV3LPFactory.sol

Attack / Description	Code Snippet	Result/Recommendation
Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).	NA	It is recommended to include natspec documentation and follow the doxygen style including @author, @title, @notice, @dev, @param, @return and make it easier to review and understand your smart contract.

5.1.3 Missing Zero address validation

Severity: LOW
Status: FIXED
Code: NA
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/3f08f36dd5670404fabfbf4226dc42f18963dd9c
File(s) affected: UniswapV3LPToken.sol, UniswapV3LPFactory.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several functions are not	UniswapV3LPFactory Line 30:	It is recommended to check addresses for the zero address before setting them.

checking for zero addresses. Setting an address to the zero address can result in loosing funds by sending it to the zero address.	`constructor(address _uniswapV3Factory, address _WETH9, address _tickHelper) {`
	`        uniswapV3Factory = _uniswapV3Factory;`
	`        WETH9 = _WETH9;`
	`        tickHelper = _tickHelper;`
	`    }`
	
	Line 72:
	`function updateTickHelper(address _newtickHelper) public onlyOwner {`
	`        tickHelper = _newtickHelper;`
	`    }`
	
	UniswapV3LPToken
	Line 61:
	
	`constructor(string memory name, string memory symbol, address _token0, address _token1, uint24 _fee, address _pool, address _WETH9) ERC20(name, symbol) {`
	`        token0 = _token0;`
	`        token1 = _token1;`
	`        fee = _fee;`
	`        pool = _pool;`
	`        WETH9 = _WETH9;`
	`        poolKey = PoolKey(token0, token1, fee);`
	`    }`	

5.1.4 Division before multiplication

Severity: LOW
Status: FIXED
Code: NA
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/b85ee8b10c3ef01667c11827fc3ffe472de4f637
File(s) affected: TickHelper.sol

<table>
<tr><th>Attack / Description</th><th>Code Snippet</th><th>Result/Recommendation</th></tr>
<tr><td>In the current implementation several functions are not checking for zero addresses. Setting an address to the zero address can result in loosing funds by sending it to the zero address.</td><td>Line 35:
<code>function getMaxTick(uint24 fee) external view returns(int24 maxTick) {
int24 tickSpacing = int24(tick_spacings[fee]);
maxTick = (maxTickValue / tickSpacing) * tickSpacing;
}</code>

Line 40:
<code>function getMinTick(uint24 fee) external view returns(int24 minTick) {
int24 tickSpacing = int24(tick_spacings[fee]);
int24 maxTick = (maxTickValue / tickSpacing) * tickSpacing;
minTick = -maxTick;
}</code></td><td>We highly recommend ordering multiplications before division.</td></tr>
</table>

5.1.5 Unchecked transfer

Severity: LOW
Status: FIXED
Code: SWC-104
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/3f08f36dd5670404fabfbf4226dc42f18963dd9c
File(s) affected: UniswapV3LPToken.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several functions ignore return values of external function calls. Execution will resume even if the called contract throws an exception.	Line 83: `IWETH9(WETH9).transfer(recipient, value);` return value of transfer function is ignored.	We highly recommend checking the return value of external function calls See SWC-104: https://swcregistry.io/docs/SWC-104

5.1.6 Redundant override function

Severity: LOW
Status: FIXED
Code: SWC-104
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/3f08f36dd5670404fabfbf4226dc42f18963dd9c
File(s) affected: UniswapV3LPToken.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation the decimals function of UniswapV3LPToken.sol is overriding the decimal function of ERC20.sol with the exact same code.	Line 75-77:
	
	`function decimals() public view virtual override returns (uint8) {`
	`        return 18;`
	`    }`
		We recommend removing the redundant overriding function from UniswapV3LPToken.sol.

	Decimals() is overriding ERC20 decimals() function with	

INFORMATIONAL ISSUES

5.1.7 Different Compiler version used

Severity: INFORMATIONAL
Status: FIXED
Code: NA
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/3f08f36dd5670404fabfbf4226dc42f18963dd9c
File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation, several pragma versions have been identified, which can lead to inconsistency and further problems while deployment.	>=0.8.3 >=0.8.0 >=0.7.6 >=0.6.0 >=0.5.0 >=0.4.0	It is recommended to normalize all files to one consistent pragma version. ex. 0.8.0 (Most used version)

5.1.8 A floating pragma is set.

Severity: INFORMATIONAL
Status: FIXED
Code: SWC-103
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/3f08f36dd5670404fabfbf4226dc42f18963dd9c
File(s) affected: ALL

Attack / Description	Code Snippet	Result/Recommendation
The current pragma Solidity	Line 1:	It is recommended to follow the latter example, as

directive is "^>=0.8.3". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.	`pragma solidity ^>=0.8.3;`	future compiler versions may handle certain language constructions in a way the developer did not foresee. i.e. Pragma solidity 0.8.0 See SWC-103: https://swcregistry.io/docs/SWC-103

5.1.9 SafeMath for pragma version higher than 0.8.0

Severity: INFORMATIONAL
Status: FIXED
Code: NA
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/3f08f36dd5670404fabfbf4226dc42f18963dd9c
File(s) affected: UniswapV3LPToken.sol

Attack / Description	Code Snippet	Result/Recommendation
In the code is SafeMath used for compiler version >=0.8.3. Since Pragma version 0.8.0 the compiler automatically checks arithmetic operations for underflow and overflow.	Line 7: `import "./libraries/SafeMath.sol";`	It is recommended to remove SafeMath from the code to avoid unnecessary gas consumption.

5.1.10 ABIEncoder v2

Severity: INFORMATIONAL
Status: FIXED
Code: NA
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/3f08f36dd5670404fabfbf4226dc42f18963dd9c
File(s) affected: UniswapV3LPToken.sol

Attack / Description	**Code Snippet**	**Result/Recommendation**
The second change since solidity 0.8.0 that is very visible is that the ABI coder v2 is activated by default. You can activate the old coder using pragma abicoder v1, or explicitly select v2 using pragma abicoder v2 - which has the same effect as pragma experimental ABIEncoderV2 had. ABI coder v2 is more complex than v1 but also performs additional checks on the input and supports a larger set of types than v1.	Line 2: `pragma abicoder v2;`	ABIEncoderV2 is activated by default since 0.8.0 and can be removed. https://blog.soliditylang.org/2020/12/16/solidity-v0.8.0-release-announcement/

5.1.11 Public function could be declared external
Severity: INFORMATIONAL
Status: FIXED
Code: NA
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/3f08f36dd5670404fabfbf4226dc42f18963dd9c
File(s) affected: UniswapV3LPFactory

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation several functions are declared as public where they could be external. For public functions Solidity immediately copies array arguments to memory, while external functions can read directly from calldata. Because memory allocation is expensive, the gas consumption of public functions is higher.	Line 72:
	`function updateTickHelper(address _newtickHelper) public onlyOwner {`
	`        tickHelper = _newtickHelper;`
	`    }`
		We recommend declaring functions as external if they are not used internally. This leads to lower gas consumption and better code readability.

5.1.12 Unused state variable
Severity: INFORMATIONAL
Status: FIXED
Code: NA
Commit: https://github.com/Boka44/unicrypt-uniswap-v3-lp/commit/3f08f36dd5670404fabfbf4226dc42f18963dd9c
File(s) affected: TickHelper.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation is a declared state variable, which is never used by any contract.	Line 8: minTickValue is nerver used in TickHelper	It is recommended to remove the unused state variable to decrease gas consumption.

5.2. SWC Attacks

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	✅

ID	Title	Relationships	Test Result
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	✅
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	✅

ID	Title	Relationships	Test Result
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')	✅
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	✅

ID	Title	Relationships	Test Result
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	✅
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	✅

5.3. Verify Claims

5.3.1 Works for rebasing and highly deflationary tokens, basically any token that works in a Uniswap v3 pool
Status: tested and verified ✅

5.3.2 Unicrypt (Deployer) is not able to withdraw locked LP tokens
Status: tested and verified ✅

5.3.3 The smart contract is coded according to the newest standards and in a secure way.
Status: tested and verified ✅

6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the November 03, 2021.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the functions. During the audit, no critical issues were found after the manual and automated security testing and the claims been successfully verified. The code readability and documentation can be slightly increased in our opinion.

Update (10.11.2021): All issues have been fixed

Update (21.02.2022)
Latest Commit: 6629bee2d62dc5d33c2ffc4c2879ab7fc7a066d2
We have checked the latest codebase; all issues have been fixed and no new issues appeared

7. Deployed Smart Contract

PENDING