Unicrypt

LP Locker

SMART CONTRACT AUDIT

29.01.2022

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 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 (16.01.2022)	Layout
0.2 (18.01.2022)	Test Deployment
0.5 (20.01.2022)	Automated Security Testing Manual Security Testing
0.7 (21.01.2022)	Verify Claims
0.9 (25.01.2022)	Summary and Recommendation
1.0 (28.01.2022)	Final document
1.1 (TBA)	Added deployed contract addresses

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 Dependencies

Dependency / Import Path	Source
anchor-lang	https://crates.io/crates/anchor-lang
anchor-spl	https://crates.io/crates/anchor-spl

4.3 Tested 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)
./programs/country-list/src/lib.rs	0d1f34a283b09282babb1cbc4b84d3ec
./programs/locker/src/lib.rs	dffe4711234e5fe570dd824e5916b5f1

4.4 Metrics / Source Unites in Scope

Total : 2 files, 895 codes, 22 comments, 150 blanks, all 1067 lines

filename	language	code	comment	blank	total
lp-locker-main/programs/country-list/src/lib.rs	Rust	100	3	22	125
lp-locker-main/programs/locker/src/lib.rs	Rust	795	19	128	942

5. Scope of Work

The Unicrypt Team provided us with the files that needs to be tested. The scope of the audit is the locker program for solana network.

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

- The program 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 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 Missing rustdoc documentation
Severity: LOW
Status: ACKNOWLEDGED
Code: CWE-1056
File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
rust can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named as rustdoc.	NA	It is recommended to include rustdoc documentation and follow https://doc.rust-lang.org/rustdoc/the-doc-attribute.html Rustdoc makes it easier to review and understand your codebase. There needs to be way more inline documentation for types, traits, functions.

INFORMATIONAL ISSUES

5.1.2 Outdated anchor versions
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: Cargo.toml

Attack / Description	Code Snippet	Result/Recommendation
Locker is using anchor-spl and anchor-lang dependency version 0.18 which is not the latest version.	[dependencies] anchor-lang = "0.18" anchor-spl = "0.18"	It is recommended to use the latest possible version of solana dependency (0.20.1 at the moment of this audit) unless rust programs are bounded to specific anchor-spl = "0.20.1" anchor-lang = "0.20.1"

5.2 Automated Vulnerability Tests

Test	Result	Recommendation
Cargo Geiger (Dependency Checker)	`error: Found 7 warnings`	No changes needs to be made
RustSec Advisory Client	`error: 2 vulnerabilities found!` `warning: 1 allowed warning found`	No changes needs to be made
Clippy	`Checking country-list v0.1.0 (/Users/softstack/Desktop/Unicrypt /Unicrypt Solana Locker Audit /lp-locker-main/programs/country-list)`	No changes needs to be made

Test	Result	Recommendation

	warning: this expression borrows a reference (`&str`) that is immediately dereferenced by the compiler
	  --> programs/country-list/src/lib.rs:76:42
	   |
	76 | let array = string_to_byte_array(&country);
	   |
	^^^^^^^^ help: change this to: `country`
	   |
	   = note: `#[warn(clippy::needless_borrow)]` on by default
	   = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#needless_borrow
	
	warning: this `impl` can be derived
	  --> programs/country-list/src/lib.rs:85:1
	   |
	85 | / impl Default for CountryBanList {
	86 | | fn default() -> Self {
	87 | | Self {
	88 | | countries: Default::default(),
	... |
	91 | | }
	92 | | }
	   | |_^
	   |
	   = note: `#[warn(clippy::derivable_impls)]` on by default
	   = help: try annotating `CountryBanList` with `#[derive(Default)]`	

Test	Result	Recommendation

	= **help**: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#derivable_impls
	
	warning: `country-list` (lib) generated 2 warnings
	Checking locker v0.1.0 (/Users/softstack/Desktop/Unicrypt /Unicrypt Solana Locker Audit /lp-locker-main/programs/locker)
	warning: **this `impl` can be derived**
	--> programs/locker/src/lib.rs:513:1
	|
	513 | / impl Default for MintInfo {
	514 | | fn default() -> Self {
	515 | | Self {
	516 | | bump: Default::default(),
	... |
	519 | | }
	520 | | }
	| |_^
	|
	= **note**: `#[warn(clippy::derivable_impls)]` on by default
	= **help**: try annotating `MintInfo` with `#[derive(Default)]`
	= **help**: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#derivable_impls	

prusti	NA	NA

6. Executive Summary

Our Chainsulting experts performed an audit of the program codebase (rust). The final debriefs took place on the January 30, 2022.

The main goal of the audit was to verify the claims regarding the security of the program. During the audit, no critical issues were found, after the manual and automated security testing and the claim have been successfully verified. The developers need to increase documentation of the codebase.

7. Deployed Program

PENDING