TOKEN HIT AUDIT

11.04.2022

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 HIT Exchange. 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 (16.03.2022)	Layout
0.5 (20.03.2022)	Automated Security Testing Manual Security Testing
0.8 (02.04.2022)	Adding of SWC, Special Checks
1.0 (05.04.2022)	Summary and Recommendation
2.0 (12.04.2022)	Final document
2.1 (12.04.2022)	Mainnet release

2. About the Project and Company

Company:

HIT TOKEN

Founder: Roberto Luis Ibarra Wiley
Co-Founder: Roberto Jesus Mejia Bejarano

Website: https:/hicoin.tech

2.1 Project Overview

Our Health-token ecosystem is based on the integration and development of products, services, smart devices and personal habits, allowing us to monitor the health and well-being of people, managing to generate predictive and meta-predictive analyzes of health in real time, with the objective of mitigating health risks in a timely manner. To do this, a token called Hicoin (HIT) was developed, which will allow the ecosystem community to connect.

Hicoin is the first token focused on the health and well-being of people, allowing commercial and financial transactions of products and services, with the direct support of companies and projects. The HIT community is made up of developers of technological projects, investors, medical alliances and end users, who interact in the ecosystem by generating health data, developing projects, purchasing products and services, creating health metadata that will be rewarded with Hicoins.

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.
Major	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.
Minor	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/Tokens

1. SafeMath.sol (0.6.2)
https://github.com/Hicoins1/solana/tree/master/clap-utils
2. Ownable.sol (0.6.2)
https://github.com/Hicoins1/solana/tree/master/clap-utils
3. Pausable.sol (0.6.2)
https://github.com/Hicoins1/solana/tree/master/download-utils
4. SafeERC20.sol (0.6.2)
https://github.com/Hicoins1/solana/tree/master/faucet
5. Context.sol (0.6.2)
https://github.com/Hicoins1/solana/tree/master/net-shaper
6. Address.sol (0.6.2)
https://github.com/Hicoins1/solana/tree/master/scripts

a. 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)
OneInchExchange.sol	211fb1df636950467711223fb2bd2d2ee43992a4530e2a84c2fc6f00ecb7d0f4
OneInchFlags.sol	422a9400755e85c7b5e2c1251fca1ff026504405c05b479fa996ce9f967f0443
RevertReasonParser.sol	b6e8ab8ea115b09362e93cf7166bdd373ad3dd36a7025a942cc847f7d45c0a18
UniERC20.sol	2c7ceb502077357a0f657217fa4e07d15bd875788af9faaaee3d523bfd852333
IOneInchCaller.sol	cfe6318b16502bf14c434a3772e173f0448da36e6327e5d3536435ecad1ac153
IERC20Permit.sol	f55d9339af4faee79c555c54bf0c95db434ad0f267d0848f55ae3acfcbc0ce6e
IChi.sol	a653c95fe096cb5de522db03a3eda4ca7bcfe3d784031693a99161def26ea5ab

5. Scope of Work & Results

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

OneInchExchange.sol
helpers/RevertReasonParser.sol
helpers/UniERC20.sol
OneInchFlags.sol

Indirect imports:
Interfaces/IOneInchCaller.sol
Interfaces/IERC20Permit.sol
Interfaces/IChi.sol

The rest of the repo was out of scope of the audit

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

- OneInchExchange contract allows to make trades that will be split to different DEXs in complex ways. They want to make sure that users' approvals on OneInchExchange contract are safe.
- That the function swap itself is safe, i.e. that user spends at most amount of srcToken and receives at least minReturnAmount of dstToken.
- We also want to be able to change the implementations of OneInchCaller freely so it is out of scope of the audit.

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.

MAJOR ISSUES

During the audit, Chainsulting’s experts found **no Major 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.

MINOR ISSUES

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

INFORMATIONAL ISSUES

5.1.2 Missing natspec documentation

Severity: INFORMATIONAL
File(s) affected: all

Attack / Description	Code Snippet	Result/Recommendation
Solidity token 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 token.

5.1.1 Hardcoded address

Severity: INFORMATIONAL
File(s) affected: helpers/UniERC20.sol

Attack / Description	Code Snippet	Result/Recommendation
The token contains unknown address. This address might be used for some malicious activity. Please check hardcoded address and it's usage.	Line: 14 `IERC20 private constant _SLD_ADDRESS = IERC20(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);`	The specific address was picked since it’s easy to remember and highly unlikely to collide with a real address. Not effecting the overall token functionality.

5.1.3 A floating pragma is set

Severity: INFORMATIONAL
Code: SWC-103
File(s) affected: all

Attack / Description	Code Snippet	Result/Recommendation
The current pragma Solidity directive is `^0.6.12`; 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.	Line: 1 `pragma solidity ^0.6.12;`	It is recommended to follow the example (0.6.12), as future compiler versions may handle certain language constructions in a way the developer did not foresee. Not effecting the overall token functionality.

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	✓
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	✓

5.3. Special Checks

5.3.1 Function: Approvals

Ressources : https://github.com/Hicoins1/solana/tree/master/scriptsCode:

```
interface ISafeERC20Extension {
    function safeApprove(token, address spender, uint256 amount) external; function
    safeTransfer(token, address payable target, uint256 amount) external;
}
```

Result:
The approval functions (safeApprove, safeTransfer) inside the OneInchExchange.sol contract are implemented in the right way and widely used.

5.3.2 Function: Swap

Code:

```
//@param srcToken source token contract address
//@param dstToken destination token contract address
//@param srcDestination address to send swapped tokens to
//@param amount amount of source tokens to be swapped
//@param minReturnAmount Minimum destination token amount expected out of this swap
//@param guaranteedAmount max number of tokens in swap outcome. will be sent to destAddress
```

```
struct SwapDescription {
        IERC srcToken;
        IERC dstToken;
        address srcDestination;
        uint256 amount;
        uint256 minReturnAmount;
        uint256 guaranteedAmount;
        uint256 flags;
        address referrer;
        bytes permit;
    }

@@
// @dev Get the initial gas amount. The intention here is to record the gas used in this function call. This gas used will be used for CHI calculations.
uint256 initialGas = gasleft();

require(desc.minReturnAmount > 0, "Min return should not be 0");

@@ function swap

returnAmount = desc.dstToken.uniBalanceOf(msg.sender).sub(initialDstBalance);

@@ function swap

initialSrcBalance.add(desc.amount).sub(desc.srcToken.uniBalanceOf(msg.sender));

@@ function swap

returnAmount = desc.dstToken.uniBalanceOf(msg.sender).sub(initialDstBalance);
```

```
@@ function _claim

   }
token.safeTransferFrom(msg.sender, dst, amount);

}
```

Result:

The implementation of this functions consider all security checks to initiate a swap where the user spends most amount of Token and receives at least minReturnAmount of Token, and makes sure the swap executes as expected.

5.3.3 Function: Rescue Funds

Resources: NA

Code:

```
function rescueFunds(IERC token, uint256 amount) external onlyOwner {
        token.uniTransfer(msg.sender, amount);
    }
```

Result: Calling this function can rescue funds, if they stuck.

5.3.4 Function: Pause

Resources: NA
Code: https://github.com/Hicoins1/solana

```
function pause() external onlyOwner {
        _pause();
    }
```

Result:
Calling this function by the contract owner can pause the contract and is needed in case of emergency, such as massive miss use of the service, future regulation, future vulnerabilities, outages of connected services such as Uniswap.

6. Executive Summary

The token as simple as possible and also not overloaded with unnecessary functions, these is greatly benefiting thesecurity of the contract. It correctly implemented widely-used and reviewed contracts from HIT and for safe mathematical operations. The main goal of the audit was to verify the claims regarding the security of the token (see the Scope of work section). According to the code, the implementation of this functions consider all security checks for a safe approval to initiate a swap where the user spends most amount of srcToken and receives at least minReturnAmount of dstToken, and makes sure the swap executes as expected. Both claims appear valid. During the audit, no critical, medium or minor issues were found after the manual andautomated security testing. 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.