
Armors Labs
Rpone.finance
Smart Contract Audit

Rpone.finance Audit Summary

Project name : Rpone.finance Contract

Project address: None

Code URL : https://github.com/rpone-fi/dex/tree/V0.0.2

Commit : 1c606cfe325d635a7ff372aa61d35a9fb40a6aa4

Code URL : https://github.com/rpone-fi/router/tree/v0.0.2

Commit : 954d8405be7a84bd81aaeddbe5d52415164edb40

Projct target : Rpone.finance Contract Audit

Blockchain : Huobi ECO Chain（Heco）

Test result : PASSED

Audit Info

Audit NO : 0X202104130008

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

Rpone.finance Audit

The Rpone.finance team asked us to review and audit their Rpone.finance contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
Rpone.finance Audit	Rock ,Hosea, Rushairer	1.0.0	2021-04-13

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Rpone.finance contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 18 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to

Armors Labs at the time of issuance of this report (" information provided " for short). Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

Audited target file

file	md5
./dex/contracts/interface/IFSwapFactory.sol	98626da281dab6d27c97c6b7461fe613
./dex/contracts/interface/IWETH.sol	1eb772a382686afd02024a4e1a172aca
./dex/contracts/interface/IFSwapERC20.sol	7c01dcda13428fc2acca2798db98c012
./dex/contracts/interface/IFSwapPair.sol	6aa2decb07f5b91cb0d926c61e07c1c3
./dex/contracts/interface/IFSwapCallee.sol	26262ec70c9306ebaa58659107045bc9
./dex/contracts/interface/IERC20.sol	191943e3f2d4fcd3e06481b46b24328d
./dex/contracts/FSwapFactory.sol	b87176ff52afe0eaa7f68df5d37adb24
./dex/contracts/FSwapPair.sol	3475db03b7734f2ebbb2fb643c73c725
./dex/contracts/mock/MockToken.sol	16e2d13a57002553d790b96a6b899548
./dex/contracts/mock/WETH.sol	a4cf19a723e4693cdd2bfb235b5f532e
./dex/contracts/mock/ExampleFlashSwap.sol	9ada4dbcb6895d1c0232f4c770a46422
./dex/contracts/mock/UniswapV2Library.sol	30b069aea6022f50dddfe3005276aaba
./dex/contracts/mock/LPToken.sol	81c60c0f31d51222cd7ccdb86371f74a
./dex/contracts/FSwapERC20.sol	4dea341238f9e8733d81d6dadb499610
./dex/contracts/SafeMath.sol	a2119f2e6933bed3f2e4882e55434bbb
./dex/contracts/UQ112x112.sol	47c83457e98580aab945da5f8cf4761e
./dex/contracts/Migrations.sol	5463d6a5c02a4465b29388b87cfaded9
./router/contracts/interface/ISwapPair.sol	4223c6ee9c2cb184f7c4bf94bf8cc0e5
./router/contracts/interface/ITokenLock.sol	354aebb17328a56328b11568c87ed8c5
./router/contracts/interface/IMasterChef.sol	590c6f2e1b7c417ec6491fc5cbec7c1f
./router/contracts/interface/ISwapFactory.sol	bbeefaaf2c51d1aa2f795edcf1bd7783
./router/contracts/interface/IWETH.sol	44f579be4157e9c5ceac4a115df98697
./router/contracts/interface/IUniswapV2Callee.sol	250afae8e5771823abf5002b0d1cafdf
./router/contracts/interface/ISwapRouter.sol	7b06c2db7865cb3fd555e9e32e9a9735
./router/contracts/interface/ISwapMining.sol	35cfe5ccbd9eea5edd52f97e04c0ec90
./router/contracts/interface/IERC20.sol	d9305730c12b59f3ffb87fa98186615a
./router/contracts/interface/ISwap.sol	437c550555fce0568a586198b334760b

file	md5
./router/contracts/interface/IOracle.sol	c8ed60de03d5a6d8fe7f367fb099b8d8
./router/contracts/Operatable.sol	be5c744ae0a5d1a478d4891994f85569
./router/contracts/Oracle.sol	6112a507dc4bf4997d354fff725a231b
./router/contracts/TransferHelper.sol	0ffc80a9936b643c7e931cd66409f20b
./router/contracts/SwapToken.sol	182265073691a9e4daa436f16e0e8899
./router/contracts/mock/uniswap/UniswapV2ERC20.sol	5bda4cb27e162f6dd98d98f3a67d3997
./router/contracts/mock/uniswap/UniswapV2Pair.sol	859d40ee672a9988b00452f06e3c021b
./router/contracts/mock/uniswap/UniswapV2Factory.sol	f1d88c40c1ad262c1eeda21a78784c2e
./router/contracts/mock/UniswapV2FactoryDeployer.sol	e868a37d3d76c9c3828dd9c8e0b252d0
./router/contracts/mock/MockToken.sol	6d467c4fd87be7e40f85e89125d34f07
./router/contracts/mock/WETH.sol	d4e0cd4ee4b409a4e14f75953ab6a8ef
./router/contracts/mock/ExampleFlashSwap.sol	57370bb85a2c2eed1888e1c724256dc0
./router/contracts/mock/MockMasterChef.sol	18a9befce261a632fac9a46234d2f9c6
./router/contracts/mock/UQ112x112.sol	3249e24858b1d913f4d4d0e1267588e8
./router/contracts/mock/UniswapV2Library.sol	41b4efb0b38254a743021582b3d7ef0b
./router/contracts/mock/MockUniswapV2Factory.sol	43deaf82476b1502a0cc56dcda00f42f
./router/contracts/mock/Math.sol	99b3c2973657334f96f33e795d5ee0e5
./router/contracts/mock/MockUniswapV2Pair.sol	827e1e39679bc88690175ba46f0f20cb
./router/contracts/SwapRouter.sol	8f3a89ca562d9acd6dce0f83b114d640
./router/contracts/RewardPool.sol	8553f322d8921620c75e6a8033479494
./router/contracts/Migrations.sol	820d7dce09279d5566e9e73caca4110a
./router/contracts/Repurchase.sol	396a6552034475a9b3cbf19f2f36d7f3
./router/contracts/Airdrop.sol	69bc4f0c982e7266f60fba68ffb473d1
./router/contracts/TokenReward.sol	661ea2e87b46f583c716ef430be98f7a
./router/contracts/TokenLock.sol	e80833ccaede78913a56735e426deaf4
./router/contracts/SwapMining.sol	4c84bca1ae9ca98753e1fd82b097fe94

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0

vulnerability level	number
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe

Contract file

```
dex/contracts
├── FSwapERC20.sol
├── FSwapFactory.sol
├── FSwapPair.sol
├── Migrations.sol
├── SafeMath.sol
├── UQ112x112.sol
├── interface
│   ├── IERC20.sol
│   ├── IFSwapCallee.sol
│   ├── IFSwapERC20.sol
│   ├── IFSwapFactory.sol
│   ├── IFSwapPair.sol
│   └── IWETH.sol
```

```
router/contracts
├── Airdrop.sol
├── Migrations.sol
├── Operatable.sol
├── Oracle.sol
├── Repurchase.sol
├── RewardPool.sol
├── SwapMining.sol
├── SwapRouter.sol
├── SwapToken.sol
├── TokenLock.sol
├── TokenReward.sol
├── TransferHelper.sol
├── interface
│   ├── IERC20.sol
│   ├── IMasterChef.sol
│   ├── IOracle.sol
│   ├── ISwap.sol
│   ├── ISwapFactory.sol
│   ├── ISwapMining.sol
│   ├── ISwapPair.sol
│   ├── ISwapRouter.sol
│   ├── ITokenLock.sol
│   ├── IUniswapV2Callee.sol
│   └── IWETH.sol
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing

conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io