Soft Yearn Finance

SMART CONTRACT AUDIT REPORT

Crypticocean

Audit Period : 22nd Sep 2020 - 24th Sep 2020

Introduction :

This Audit Report highlights the overall security of SYFI Smart Contracts. With this report, we have tried to ensure the reliability of their smart contract by complete assessment of their system's architecture and the smart contract codebase along with business logic.

Auditing Approach and Methodologies applied :

The tokens has performed thorough testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our tokens then performed a formal line by line inspection of the Smart Contract to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple tokens members and this included -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the process.
2. Analysing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests
4. Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analysing the security of the on-chain data.

Audit Details

- Project Name: SYFI
- website/Etherscan Code:

https://etherscan.io/address/0xa77d7af8b02aB276a71A749D16dcB57831825ABd#code
https://etherscan.io/address/0x1fb8ce2040f79927ecf10b359c8bd987c61b09cd#code
https://etherscan.io/address/0x91d0b6296e334b872ac6cb297d14eb7cd2612ad8#code
https://etherscan.io/address/0xc3ac881fb2a3fe48a8e8303ac251d341d1dd603d#code

- Languages: Solidity (Smart contract), Javascript (Unit Testing)
- Platforms and Tools: Remix IDE, Truffle, Truffle tokens, Ganache, Slither, Surya, Echidna, Manticore
- Audit Period : 22nd September 2020 - 24th September 2020

Summary of SYFI Smart Contract :

CrypticOcean conducted a security audit of a smart contract of SYFI. SYFI contract is used to create the basic ERC20 token with elastic supply, which is a SYFI TOKEN, Smart contract contains basic functionalities of an ERC20 token.

Name: Soft Yearn Finance
Symbol: SYFI
Total supply : 60000 SYFI

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security: Identifying security related issues within each contract and the system of contracts.

Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Security Level references :

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	Low	Medium	High
Open	6	2	1
Closed	0	0	0

High severity issues:-

1. Reentrancy Issue posibility

Reentrancy in UFragmentsPolicy.initialize(address,UFragments,IOracle) (UFragmentsPolicy.sol#1006-1027):

External calls:

- oracle.update() (UFragmentsPolicy.sol#1025)
    - (baseYFIValue,None,None,None) = oracle.getData() (UFragmentsPolicy.sol#1026)

State variables written after the call(s):

- (baseYFIValue,None,None,None) = oracle.getData() (UFragmentsPolicy.sol#1026)

Do not update a variable after external call

Status: Not fixed Yet

Medium Severity Issues:-

1. Reuse of multiple contracts are found listing all in below table please remove the reused code and make it optimize

SafeMath is re-used:
- ExampleOracleSimple.sol#9-150
- UFragmentsPolicy.sol#14-72
- Orchestrator.sol#161-219
- UFragmentsPolicy.sol#14-72

IUniswapV2Pair is re-used:
- ExampleOracleSimple.sol#168-217
- UFragmentsPolicy.sol#746-795
- Orchestrator.sol#759-808
- UFragmentsPolicy.sol#746-795

Initializable is re-used:
- Orchestrator.sol#22-67
- UFragmentsPolicy.sol#91-136

Ownable is re-used:
- Orchestrator.sol#79-150
- UFragmentsPolicy.sol#148-219

As a result, the inherited contracts are not correctly analyzed:
- Orchestrator (Orchestrator.sol#1103-1247)

SafeMathInt is re-used:
- Orchestrator.sol#255-327
- UFragmentsPolicy.sol#255-327

UInt256Lib is re-used:
- Orchestrator.sol#337-352
- UFragmentsPolicy.sol#337-352

IERC20 is re-used:

- Orchestrator.sol#363-390
- UFragmentsPolicy.sol#359-386

ERC20Detailed is re-used:
- Orchestrator.sol#403-436
- UFragmentsPolicy.sol#395-428

UFragments is re-used:
- Orchestrator.sol#454-752
- UFragmentsPolicy.sol#443-741

IOracle is re-used:
- Orchestrator.sol#810-813
- UFragmentsPolicy.sol#797-800

UFragmentsPolicy is re-used:
- Orchestrator.sol#824-1090
- UFragmentsPolicy.sol#811-1077

Orchestrator (Orchestrator.sol#1103-1247) inherits from a contract for which the name is reused.
- Slither could not determine which contract has a duplicate name:
  -Ownable (UFragmentsPolicy.sol#148-219)
- Check if:
  - A inherited contract is missing from this list,
  - The contract are imported from the correct files.

UFragments (Orchestrator.sol#454-752) inherits from a contract for which the name is reused.
- Slither could not determine which contract has a duplicate name:
  -Ownable (UFragmentsPolicy.sol#148-219)
  -ERC20Detailed (UFragmentsPolicy.sol#395-428)
- Check if:
  - A inherited contract is missing from this list,
  - The contract are imported from the correct files.

UFragmentsPolicy (Orchestrator.sol#824-1090) inherits from a contract for which the name is reused.
- Slither could not determine which contract has a duplicate name:

-Ownable (UFragmentsPolicy.sol#148-219)
- Check if:
- A inherited contract is missing from this list,
- The contract are imported from the correct files.
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused

Status: Not fixed Yet

2. Uninitialised Local variable

UniswapV2Library.getAmountsOut(address,uint256,address[]).i
(ExampleOracleSimple.sol#397) is a local variable never initialized
FixedPoint.mul(FixedPoint.uq112x112,uint256).z
(ExampleOracleSimple.sol#274) is a local variable never initialized

Status: Not fixed Yet

Low Severity Issues:-

1. **Contract version should be locked pragma solidity ^0.4.24; Ufragment.sol**

Description:

An unlocked compiler version in the source code of the contract permits the user to compile it

at or above a particular version. This, in turn, leads to differences in the generated bytecode

between compilations due to differing compiler version numbers.

This can lead to an ambiguity when debugging as compiler specific bugs may occur in the

codebase that would be hard to identify over a span of multiple compiler versions rather than a

specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the full project can be compiled at.

Status: Not fixed Yet

2. **pragma solidity ^0.4.24; UfragmentPolicy.sol Line 356**

Description:

An unlocked compiler version in the source code of the contract permits the user to compile it

at or above a particular version. This, in turn, leads to differences in the generated bytecode

between compilations due to differing compiler version numbers.

This can lead to an ambiguity when debugging as compiler specific bugs may occur in the

codebase that would be hard to identify over a span of multiple compiler versions rather than a

specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the full project can be compiled at.

3. **Should use Address guard** , at **address uniswapV2Pair = 0xf72a9316620422f9921a1616c1934543e9e7e95e;**

Recommendation

Should be used as

“address(0xf72a9316620422f9921a1616c1934543e9e7e95e);”

And also use iscontract() function to check if address is of contract or not.

Status: Not fixed Yet

4. Should be declared as constant

   “address uniswapV2Pair = 0xf72a9316620422f9921a1616c1934543e9e7e95e;”

Recommendation

Should be used as

“Address unisswapV2Pair contant address(0xf72a9316620422f9921a1616c1934543e9e7e95e);”

5. Unused State variables

SafeMathInt.MAX_INT256 (UFragmentsPolicy.sol#257) is never used in SafeMathInt (UFragmentsPolicy.sol#255-327)

Ownable._____gap (UFragmentsPolicy.sol#218) is never used in UFragments (UFragmentsPolicy.sol#443-741)

Ownable._____gap (UFragmentsPolicy.sol#218) is never used in UFragmentsPolicy (UFragmentsPolicy.sol#811-1077)

Ownable._____gap (UFragmentsPolicy.sol#218) is never used in Orchestrator (Orchestrator.sol#1103-1247)

6. Functions should be declared External

renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (UFragmentsPolicy.sol#195-198)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (UFragmentsPolicy.sol#204-206)
name() should be declared external:
- ERC20Detailed.name() (UFragmentsPolicy.sol#409-411)
symbol() should be declared external:
- ERC20Detailed.symbol() (UFragmentsPolicy.sol#416-418)
decimals() should be declared external:
- ERC20Detailed.decimals() (UFragmentsPolicy.sol#423-425)
totalSupply() should be declared external:
- UFragments.totalSupply() (UFragmentsPolicy.sol#603-605)
balanceOf(address) should be declared external:
- UFragments.balanceOf(address) (UFragmentsPolicy.sol#611-613)
transfer(address,uint256) should be declared external:
- UFragments.transfer(address,uint256) (UFragmentsPolicy.sol#621-632)
allowance(address,address) should be declared external:
- UFragments.allowance(address,address) (UFragmentsPolicy.sol#640-646)
transferFrom(address,address,uint256) should be declared external:
- UFragments.transferFrom(address,address,uint256) (UFragmentsPolicy.sol#654-669)
approve(address,uint256) should be declared external:
- UFragments.approve(address,uint256) (UFragmentsPolicy.sol#682-690)
increaseAllowance(address,uint256) should be declared external:

- UFragments.increaseAllowance(address,uint256) (UFragmentsPolicy.sol#699-713)
decreaseAllowance(address,uint256) should be declared external:
- UFragments.decreaseAllowance(address,uint256) (UFragmentsPolicy.sol#721-740)
initialize(address,UFragments,IOracle) should be declared external:
- UFragmentsPolicy.initialize(address,UFragments,IOracle) (UFragmentsPolicy.sol#1006-1027)
inRebaseWindow() should be declared external:
- UFragmentsPolicy.inRebaseWindow() (UFragmentsPolicy.sol#1033-1038)
setCompleted(uint256) should be declared external:

Unit Testing

Test Suite

Contract: SYFI Token Contracts

✓ Should check a name of a token
✓ Should check a symbol of a token
✓ Should check a decimal of a token
✓ Should check a owner of a token (61ms)
✓ Should check a balance of a token contract
✓ Should check a balance of a owner (73ms)
✓ Should check the otc Token Sent when not sent using function
✓ Should check the total supply of a token contract
✓ Should check if contract is paused or not
✓ Should Not be able to pause the contract by non owner account (52ms)
✓ Should be able to pause the contract (40ms)
✓ Should check if contract is paused or not after pause
✓ Should Not be able to unpause the contract by non owner account (55ms)
✓ Should be able to unpause the contract from Owner Account
✓ Should check if contract is paused or not after unpaused
✓ Should check the otc Token Sent before sending otc tokens
✓ Should check a balance of a otc token receiver
✓ Should Not be able to send token to otc when value is wrong (43ms)
✓ Should Not be able to send token to otc by Non owner account (39ms)
✓ Should be able to send token to otc by owner only (50ms)

✓ Should check a balance of a otc token reciever
✓ Should check the otc Token Sent before sending otc tokens
✓ Should check a balance of a owner
✓ Should Not be able to send token to uniSwapLiquidity by Non owner Account (39ms)
✓ Should be able to send token to uniSwapLiquidity by owner only (54ms)
✓ Should check a balance of a Uniswap token receiver
✓ Should check the Uniswap Token Sent after sending Uniswap tokens
✓ Should check a balance of a Owner
✓ Should check a balance of a future token receiver
✓ Should check the future Token Sent before sending future tokens
✓ Should check a balance of a Owner
✓ Should Not be able to send token to ecosystem by Non owner Account (45ms)
✓ Should be able to send token to ecosystem by owner only (45ms)
✓ Should check a balance of a ecosystem token receiver
✓ Should check the ecosystem Token Sent after sending tokens
✓ Should check a balance of a owner
✓ Should check a balance of a accounts[1] whos tokens are locked
✓ should check a cycle of locked token Account[1]
✓ Should Not be able to transfer token when locking period is active (38ms)
✓ Should be able to increase time to get first cycle
✓ Should be able to check cycle
✓ Should be able to transfer token when locking period cycle is 1 (45ms)
✓ should check tokens tokens released by tokens token holder

accounts[1]

✓ Should check a balance of a beneficiary accounts[4]

✓ Should be able to transfer token when locking period cycle is 1 again as limit is not matched yet (51ms)

✓ should check tokens tokens released by tokens token holder accounts[1]

✓ Should check a balance of a beneficiary accounts[4]

✓ Should not be able to transfer token when locking period cycle is 1 again (47ms)

✓ Should be able to increase time to get second cycle

✓ Should be able to check cycle

✓ Should be able to increase time to get last cycle

✓ Should be able to check cycle

✓ Should be able to transfer token when locking period cycle is 34, last cycle (49ms)

✓ should check tokens tokens released by tokens token holder accounts[1]

✓ should check tokens tokens released and initial tokens are same now after Releasing all tokens

✓ Should check a balance of a beneficiary of accounts[4]

✓ Should be able to transfer tokens to locked account (51ms)

✓ Should check a balance of a sender of accounts[4] after sending all tokens

✓ Should check a balance of a beneficiary of accounts[1]

✓ Should be able to transfer tokens by previous locked account, non locking tokens only after locking over

✓ Should check a balance of a beneficiary of accounts[4] after sending all tokens

✓ Should check a balance of a sender accounts[1]

✓ should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4

✓ should Approve accounts[1] to spend specific tokens of

accounts[4]

✓ should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (42ms)

✓ should increase Approve accounts[4] to spend specific tokens of accounts[1]

✓ should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (38ms)

✓ should decrease Approve accounts[4] to spend specific tokens of accounts[1]

✓ should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (58ms)

✓ should decrease Approve accounts[4] to spend specific tokens of accounts[1]

✓ should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (50ms)

✓ should Approve accounts[1] to spend specific tokens of accounts[4] agin

✓ should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (46ms)

✓ should be able to transferfrom accounts[4] to accounts[1]

✓ should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (50ms)

✓ Should check a balance of a beneficiary of accounts[4] after sending all tokens

✓ Should check a balance of a receiver accounts[1]

✓ Should check a owner of a token before transferring ownership

✓ Should not be able to transfer ownership before

✓ Should be able to transfer ownership before

✓ Should be able to accept transfer ownership before (39ms)

✓ Should check a owner of a token after transferring ownership

✓ Should be able to transfer ownership again to accounts[0]

(39ms)
✓ Should be able to accept transfer ownership before (41ms)
✓ Should check a owner of a token after transferring ownership
✓ Should Not be able to send token to otc balance higher then user have (44ms)
✓ Should be able to send tokens token by owner only (46ms)
✓ Should check a balance of a owner after sending tokens tokens
✓ Should check the tokens Token Sent after releasing tokens tokens
✓ Should check a balance of a tokens token receiver
✓ Should check a balance of a accounts[6]
✓ should check approval by accounts 5 to accounts 6 to spend locked tokens on the behalf of accounts 5
✓ should Approve accounts[1] to spend specific tokens of accounts[4]
✓ should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (40ms)
✓ should check an address is tokens token holder
✓ should check a cycle of locked token Account[5]
✓ should check tokens sent initially tokens tokens to accounts[5]

Contract: rebase Token Contracts

✓ Should correctly initialize constructor of SYFI token Contract (62ms)
✓ Should correctly initialize constructor of Rebase token Contract (56ms)
✓ Should not be able to call rebaseFunction of token contract when rebase address is not initialised (50ms)
✓ Should not set the address of rebaseContract to token Contract from non owner account (44ms)

✓ Should set the address of rebaseContract to token Contract (43ms)
✓ Should check rebase address in token Contract
✓ Should not be able to call rebaseFunction of token contract by non rebase Contract or by wallet address (43ms)

Final Result of Test:

✓ 149 Passing (3s) PASSED

❌ 0 Failed

Final Result of Test:

✓ 67 Passing (3s) PASSED

❌ 0 Failed

Comments:

Use case of the smart contract is very well designed and Implemented. Overall, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. **SYFI** development Team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.

All the bugs, suggestions and recommends has been considered by SYFI tokens and some of the issues they will handle to their own as those issues or calls have been handle by the only owner,