SIGH Finance

Smart Contract Audit Report

IMMUNE BYTES

Audits

February 08, 2021

Introduction

1. About SIGH Finance

SIGH Finance is an initiative under LIFEORDREAM Venture Studio, whose Goal is to fasten the transition of human civilization into a multi-planetary species. SIGH Finance consists of an on-chain lending protocol designed to farm 'Volatility-Risk' of the supported instruments and an off-chain network intelligence infrastructure to study, develop and optimize crypto-econometric models supported by Digital Curation Markets (DCM), which shall later be deployed to support some of the space research projects and training data market for semi-autonomous robots deployed in space accessible via BMI.

Visit https://sigh.finance/ to know more.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit https://immunebytes.com/ to know more about the services.

Documentation Details

SIGH Finance team has provided documentation for the purpose of conducting the audit. The documents are:

1. https://docs.sigh.finance/
2. A brief overview of the contract over email.

Audit Process & Methodology

ImmueBytes team has performed thorough testing of the project starting with analyzing 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 team then performed a formal line by line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing 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. Analyzing 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. Analyzing the security of the on-chain data.

Audit Details

- Project Name: SIGH Finance
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit hash for audit: 9feee84e18cabb4015ca60dc016340f2c94af27a
- SighBoosters.sol - https://kovan.etherscan.io/address/0x0e4e1135eb9de57b48cd5119ca6900cbf2e23bab
- SignBoosterSale.sol - https://kovan.etherscan.io/address/0x4d89ac60b01e050c1af1de45a68ded14189d97ac

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 into the following three categories:

1. Security: Identifying security related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

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

Admin/Owner Privileges can be misused either intentionally or unintentionally.

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.

Issues	Admin/Owner Privileges	High	Medium	Low
Open	5	3	2	5
Closed	-	-	-	-

Admin/Owner Privileges

The **admin/owner** of **SIGHBoosters** and **SIGHBoostersSale** contract has various privileges over the smart contracts. These privileges can be misused either intentionally or unintentionally (in case admin's private key gets hacked). We assume that these extra rights will always be used appropriately. Some of these admin rights are listed below.

1. **Admin account can set any *uint256* value as *platformFeeDiscount* and *sighPayDiscount*.**
   In the **SIGHBoosters.sol**, the admin of smart contract can set any **uint256** value (ranging from 0 to 2^256 - 1) as the **_platformFeeDiscount** and **_sighPayDiscount** via the **updateDiscountMultiplier**() function.

2. **Admin has the right to pull out erc20 tokens from the SIGHBoostersSale contract.**
   a. Admin has the right to pull out any amount of **tokenAcceptedAsPayment** funds (the ERC20 token used for Boosters's Sale) from the **SIGHBoostersSale** contract anytime via the **transferBalance()** function.

   b. Admin has the right to pull out any amount of any ERC20 token from the **SIGHBoostersSale** contract anytime via the **transferTokens**() function.

3. **Admin has the right to change any Booster's price.**
   Admin has the right to set any **uint256** value as the sale price for any Sigh Booster anytime via the **updateSalePrice**() function.

4. **Admin has the right to change the accepted ERC20 token for Sigh Booster sale.**
   Admin has the right to set any ethereum address as the **tokenAcceptedAsPayment** anytime via the **updateAcceptedToken()** function.

5. **Admin has the right to change the sale start time.**
   Admin has the right to set any **timestamp** which is greater than **now**() as the **initiateTimestamp** via the **updateSaleTime()** function.

Recommendation:
Consider hardcoding a predefined range for variables. Also consider adding some governance for admin rights for smart contracts or atleast use a multi-sig wallet as admin/owner address.

High severity issues

1. **The setApprovalForAll() function does not function as intended.**
   The **setApprovalForAll()** function in **SIGHBoosters** smart contract is one of the standard ERC721 functions which is used by token holders to approve any other address to use their ERC721 tokens**.** However due to the **require** statements added in the function at **Line 236** and **237**, the function does not work as intended and reverts the approval transactions. The conditions in the **require** statements always return a **false** result.

   Recommendation:
   Consider removing the mentioned **require** statements or implement the conditions properly.

2. **SIGH Boosters can be bought at zero price.**
   In the **SIGHBoostersSale** contract the normal flow of a Booster's sale is this -
   The user sends/deposits the Booster he wants to sell to the **SIGHBoostersSale** contract, if that type of Booster is being sold for the first time then admin has to update the price of that respective Booster type, then any user can purchase that token.
   However anyone can purchase the Booster as soon as it is deposited to the **SIGHBoostersSale** contract, i.e., before the price is updated by the admin. This way an attacker can purchase the Booster at **zero** price.
   This attack is only possible for newly deposited Booster type.

   Recommendation:
   Consider adding some checks in the **buyBoosters**() function which prohibits attackers to purchase a token when its price is **zero** (not yet updated by admin).

3. **The onERC721Received() function is open for anyone to call anytime.**
   The **onERC721Received()** function of **SIGHBoostersSale** contract can be called by anyone with any **tokenId** as input. This can prevent the actual owner of a specific **tokenId** to deposit his Sigh Booster in **SIGHBoostersSale** contract for sale. This vulnerability can be used to stop few or all **tokenIds** from getting sold via the **SIGHBoostersSale** contract

   Recommendation:
   Consider restricting the **onERC721Received()** function call to only **SIGHBoosters** contract so that anyone else cannot call the **onERC721Received()** function directly.

Medium severity issues

1. **The SIGHBoostersSale contract might not work with non-standard ERC20 tokens.**
   The **SIGHBoostersSale** contract expects the **tokenAcceptedAsPayment** to be 100% ERC20 compatible, it expects the ERC20 functions to return a **boolean** value after execution. This can cause issue if any non-standard ERC20 token like USDT (which do not returns a **boolean** value) is set as the **tokenAcceptedAsPayment,** in this scenario all the ERC20 function calls like **transfer**() and **transferFrom**() will fail and get reverted.

   Recommendation:
   Consider making the **SIGHBoostersSale** contract non-standard ERC20 compatible so that it can be used with tokens like USDT.

2. **The Booster sale can happen even when the *initiate time* is not set.**
   The **Booster** sale functionality in **SIGHBoostersSale** contract is intended to be started after a certain period of time which is set by the admin. However the implementation of this functionality is not correct.
   Since the **default** value of every integer variable is Solidity is **zero (0)**, the **require** statement in **buyBooters**() function at **Line 96** can be bypassed in the case when **sale start time** is not set yet.

   Recommendation:
   Consider taking the sale start time as input in the contract's **constructor** so that its value is set at the time of contract deployment.

Low severity issues

1. **Failing test case for updateDiscountMultiplier() function.**
   The test case at **Line 121** in **SIGHBoosters_test.py** expects a revert when a zero value is set as the **_platformFeeDiscount.** However no logic is present in the smart contract to revert the transaction on zero as input. So the test case fails.

2. **Incorrect require statement in getTokenBalance().**
   In the **SIGHBoostersSale** contract the **require** statement present in **getTokenBalance()** function is incorrect. The function takes **token** address variable as input but checks the validity of **tokenAcceptedAsPayment** variable.

   Recommendation:
   Consider replacing the **tokenAcceptedAsPayment** variable with **token** at the above mentioned place.

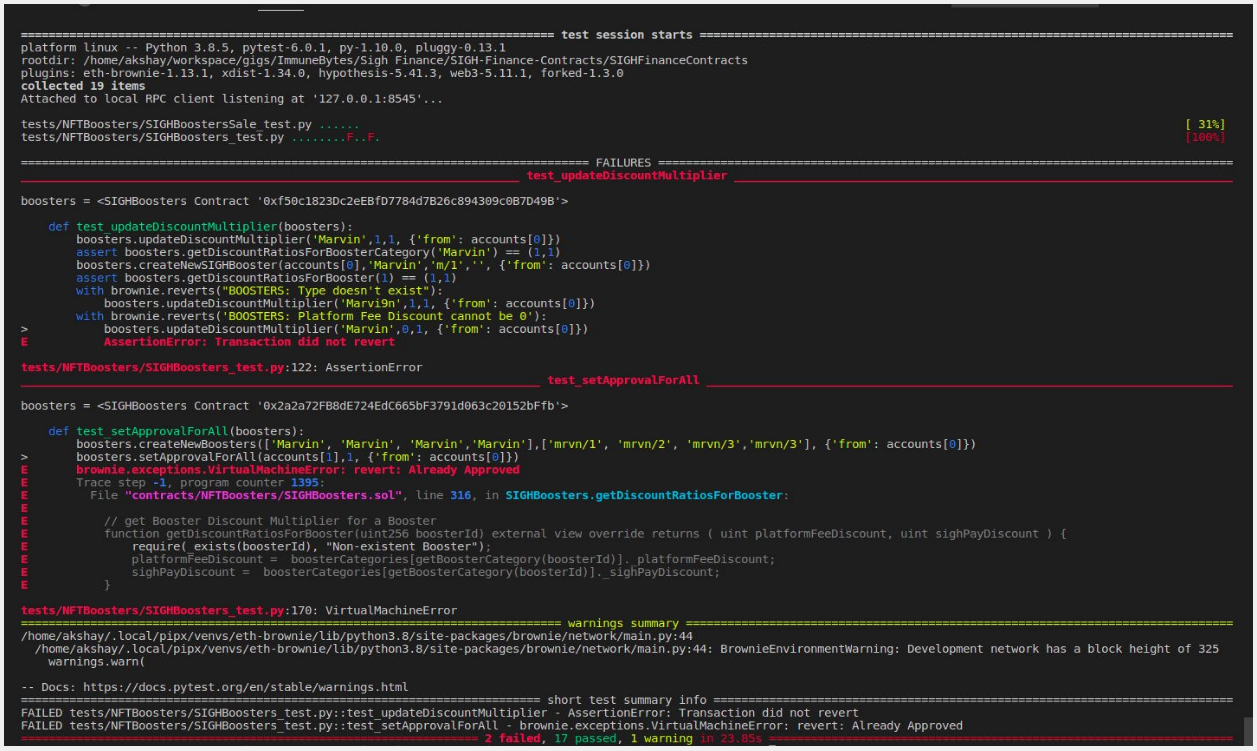
3. **Redundant check in safeTransferFrom() function.**
   In **SIGHBoosters** contract at **line 247** the **safeTransferFrom()** function contains a redundant **require** statement which is again repeated and checked at **line 252**. Both the statements check the **blacklisting** of a Booster.

   Recommendation:
   Consider removing the **require** statement at Line 247.

4. **Spelling mistake in the developer's comment.**
   The **line 92** of **SIGHBoostersSale** contract has a spelling mistake in the developer's comment. The comment spells *By* instead of *Buy.*

5. **Funds collected via Booster sale are not transferred to the Booster seller.**
   In **SIGHBoostersSale** contract, the funds collected via a Booster sale are not transferred to the Booster's seller/depositor. These funds remain inside the **SIGHBoostersSale** contract until they are claimed by the admin.
   This could be the intended business logic for the **SIGHBoostersSale** contract, so resolution of this point is upon the discretion of the SIGH Finance team.

Unit Test and Coverage



```
=========================================================================== test session starts ===========================================================================
platform linux -- Python 3.8.5, pytest-6.0.1, py-1.10.0, pluggy-0.13.1
rootdir: /home/akshay/workspace/gigs/ImmuneBytes/Sigh Finance/SIGH-Finance-Contracts/SIGHFinanceContracts
plugins: eth-brownie-1.13.1, xdist-1.34.0, hypothesis-5.41.3, web3-5.11.1, forked-1.3.0
collected 19 items
Attached to local RPC client listening at '127.0.0.1:8545'...

tests/NFTBoosters/SIGHBoostersSale_test.py ......                                                                                                                   [ 31%]
tests/NFTBoosters/SIGHBoosters_test.py ........F..F.                                                                                                                [100%]

================================================================================ FAILURES ================================================================================
______________________________________________________________________ test_updateDiscountMultiplier ______________________________________________________________________

boosters = <SIGHBoosters Contract '0xf50c1823Dc2eEBfD7784d7B26c894309c0B7D49B'>

    def test_updateDiscountMultiplier(boosters):
        boosters.updateDiscountMultiplier('Marvin',1,1, {'from': accounts[0]})
        assert boosters.getDiscountRatiosForBoosterCategory('Marvin') == (1,1)
        boosters.createNewSIGHBooster(accounts[0],'Marvin','m/1','', {'from': accounts[0]})
        assert boosters.getDiscountRatiosForBooster(1) == (1,1)
        with brownie.reverts("BOOSTERS: Type doesn't exist"):
            boosters.updateDiscountMultiplier('Marvi9n',1,1, {'from': accounts[0]})
        with brownie.reverts('BOOSTERS: Platform Fee Discount cannot be 0'):
>           boosters.updateDiscountMultiplier('Marvin',0,1, {'from': accounts[0]})
E           AssertionError: Transaction did not revert

tests/NFTBoosters/SIGHBoosters_test.py:122: AssertionError
_________________________________________________________________________ test_setApprovalForAll _________________________________________________________________________

boosters = <SIGHBoosters Contract '0x2a2a72FB8dE724EdC665bF3791d063c20152bFfb'>

    def test_setApprovalForAll(boosters):
        boosters.createNewBoosters(['Marvin', 'Marvin', 'Marvin','Marvin'],['mrvn/1', 'mrvn/2', 'mrvn/3','mrvn/3'], {'from': accounts[0]})
>       boosters.setApprovalForAll(accounts[1],1, {'from': accounts[0]})
E       brownie.exceptions.VirtualMachineError: revert: Already Approved
E       Trace step -1, program counter 1395:
E         File "contracts/NFTBoosters/SIGHBoosters.sol", line 316, in SIGHBoosters.getDiscountRatiosForBooster:
E
E           // get Booster Discount Multiplier for a Booster
E           function getDiscountRatiosForBooster(uint256 boosterId) external view override returns ( uint platformFeeDiscount, uint sighPayDiscount ) {
E               require(_exists(boosterId), "Non-existent Booster");
E               platformFeeDiscount =  boosterCategories[getBoosterCategory(boosterId)]._platformFeeDiscount;
E               sighPayDiscount =  boosterCategories[getBoosterCategory(boosterId)]._sighPayDiscount;
E           }

tests/NFTBoosters/SIGHBoosters_test.py:170: VirtualMachineError
============================================================================ warnings summary ============================================================================
/home/akshay/.local/pipx/venvs/eth-brownie/lib/python3.8/site-packages/brownie/network/main.py:44
  /home/akshay/.local/pipx/venvs/eth-brownie/lib/python3.8/site-packages/brownie/network/main.py:44: BrownieEnvironmentWarning: Development network has a block height of 325
    warnings.warn(

-- Docs: https://docs.pytest.org/en/stable/warnings.html
========================================================================= short test summary info =========================================================================
FAILED tests/NFTBoosters/SIGHBoosters_test.py::test_updateDiscountMultiplier - AssertionError: Transaction did not revert
FAILED tests/NFTBoosters/SIGHBoosters_test.py::test_setApprovalForAll - brownie.exceptions.VirtualMachineError: revert: Already Approved
================================================================ 2 failed, 17 passed, 1 warning in 23.85s ================================================================
```

- Some unit tests provided by SIGH Finance team are failing and are mentioned in detail in the **Issues** section above.
- Test coverage of smart contracts is not 100%.

Recommendation:
Our team recommends 100% line and branch coverage for unit test cases. So we suggest that the developers should write more extensive test cases for the contracts.

General Recommendations

1. **The contract should follow the proper Solidity Style Guide.**
   Consider following the Solidity guide. It is intended to provide coding/naming convention for writing solidity code.

   Recommendation:
   Solidity style guide can be found using the following link https://docs.soliditylang.org/en/v0.7.0/style-guide.html

Concluding Remarks

While conducting the audits of SIGH Finance smart contract, it was observed that the contracts contain Admin/Owner Privileges, High, Medium, and Low severity issues, along with a few areas of recommendations.

Our auditors suggest that High, Medium, Low severity issues should be resolved by SIGH Finance developers. Resolving the Admin/Owner Privileges and areas of recommendations are up to SIGH Finance's discretion. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the SIGH Finance platform or its product neither this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the one audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.