Yearn Gold Finance

Staking Platform

Smart Contract Audit Report

IMMUNE BYTES

Audits

October 27, 2020

Introduction

1. About Yearn Gold Finance

Yearn Gold Finance (YGF) is a protocol that unifies leading DeFi protocols and blockchain infrastructure by standardizing communication between them to create and execute complex financial transactions while championing Privacy, Anonymity, and Sovereignty. It’s a community project where only a small portion is allocated to Dev & the YGF team members.

YGF protocol will combine some of the best features of a decentralized finance market protocol, maximize its unique features, enabling users to enjoy the promise of a decentralized finance marketplace.

https://yearngoldfinance.com/

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 http://immunebytes.com/ to know more about the services.

Documentation Details

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

1. Short Description on Telegram
2. Medium Link
   https://info-ygf.medium.com/

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: Yearn Gold Finance
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit hash for audit: ceeb9be2ca7eb8e73c4e56ce85f77093ce1302d8

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:

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	High	Medium	Low
Open	2	1	4
Closed	-	-	-

High severity issues

1. **Fees are not getting distributed and can get stuck inside smart contract.**
   The **stake**() function in **YgfStake**.sol is intended to deduct fees from the staking amount, however these fees are not getting transferred to any wallet/address. These fee amounts stay inside the **YgfStaking** contract. Also there is no function to collect these fees except **transferToken**() function which itself possesses some issues (See High Severity Issue 2).

Recommendation:
Consider adding transfer of fee amount inside the **stake**() function to a predefined address or admin so that whenever anyone stakes any amount the fee gets transferred automatically.

2. **Implementation of transferToken() is not correct, and can cause loss of funds.**
The **transferToken**() function in **YgfStake**.sol is not implemented correctly and has few issues.
Firstly, the function is not admin protected hence can be called by anyone. This function seems to be used to collect fees and dust amounts of token from the YgfStake contract after all users unstake their funds. But anyone can create a bot to monitor the contract for the last unstaking transaction and then immediately drain out all the fees accumulated inside the contract.

Second, it must never be assumed that the **totalStackAmount** will be absolute zero. Since the smart contract has statements (Line 53) where it performs division of numbers, precision loss can occur which will result in a non zero **totalStackAmount**. This situation can also arise if any user who has staked his funds never comes back to unstake them either intentionally or unintentionally (loses his private key). In this case as well the totalStackAmount won't be **Zero**.

Recommendation:
Consider adding the automatic fee distribution mechanism as suggested in High Severity Issue 1. Also add the onlyOwner modifier to **transferToken**() function so that only admin can call it.

Medium severity issues

1. **setTime() function should not be there in the main contract.**
As mentioned in developer comments, the **setTime**() function in **YgfStaking** contract is intended for testing purposes and must not be deployed on Mainnet. But since this function is present in the **YgfStaking** contract, it creates a room for human error, i.e. unintentionally deploying the contract with this function. Human error has happened in the past and has caused loss of funds so this issue should be handled explicitly.

Recommendation:
Consider creating a **MockYgfStake** or **TestYgfStake** contract which inherits the **YgfStaking** contract and place the **setTime**() function inside this test contract.

Low severity issues

1. **unStake() can be called by anyone.**
   The **unStake**() function in **YgfStaking** smart contract is intended to be called by the users who have already staked funds and now want to unstake those funds. However there is no check in the function which verifies whether the caller has staked some funds or not. Anyone can call this function any number of times. However this does not cause any loss of funds but the scenario is unintentional.

   Recommendation:
   Consider adding a **require** statement which verifies that the caller must have staked some funds prior to calling this function.
   require(lastStack[msg.sender] != 0, “No funds Staked”)

2. **The Contract prohibits the users to stake again.**
   The **stake**() function has this statement r**equire(stakedAmount[msg.sender] == 0,"ERR_ALREADY_STACKED")**; which forces the users to stake only once. This implementation logic stops the user from staking his/her funds again.

3. **The contract only counts complete days for reward calculation.**
   As per the implementation of **YgfStaking** contract, it only accounts the complete days passed since the time users staked their funds. So if a user staked funds for 10 days and 23 hours, he will only get the rewards for staking for 10 days only (23 hours neglected).

4. **Ownable is inherited but not used.**
   As per the current implementation of **YgfStaking** contract there are no admin rights in the smart contract but still **Ownable** contract is used (inherited). This increases the deployment size and cost of **YgfStaking** smart contract.

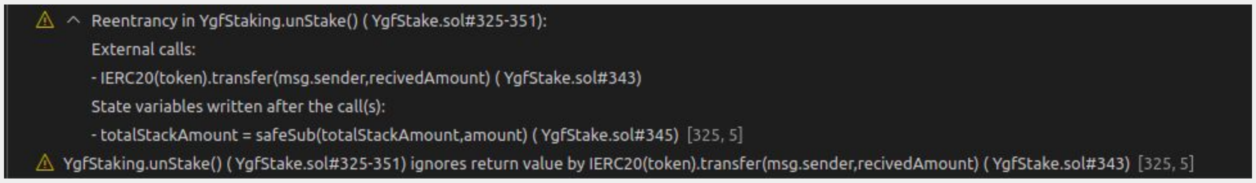
   Recommendation:
   Consider either removing the **Ownable** contract inheritance or implement admin restriction to functions as suggested in High Severity Issue 2.

Unit Test

No Test Cases has been written by the developers for Yearn Gold Finance.

Recommendation:
Our team suggests that the developer should write extensive test cases.

Slither Tool Result



```
⚠ ^ Reentrancy in YgfStaking.unStake() ( YgfStake.sol#325-351):
      External calls:
      - IERC20(token).transfer(msg.sender,recivedAmount) ( YgfStake.sol#343)
      State variables written after the call(s):
      - totalStackAmount = safeSub(totalStackAmount,amount) ( YgfStake.sol#345) [325, 5]
⚠ YgfStaking.unStake() ( YgfStake.sol#325-351) ignores return value by IERC20(token).transfer(msg.sender,recivedAmount) ( YgfStake.sol#343) [325, 5]
```

Recommendations

1. **All non changing variables should be marked as constant.**
   Most of the variables in the **YgfStaking** contract have a fixed value like **totalStackAmount**, **rewardBreakingPoint**, etc. These variables should be explicitly marked as **constant**.

2. **Spelling Mistakes**
   There are some spelling mistakes in **YgfStake.sol** at **Line - 23, 81, 92** and **Ownable.sol** at **Line - 35.** Some of these are public functions/variables which will be accessed from outside the **YgfStaking** contract via frontends or other smart contracts. It is recommended to have proper names to avoid confusion for users/developers.

3. **Unused Variable**
   The YgfStake contract contains **PERCENT_NOMINATOR** public variable which is never used in the contract. Unused variables increase the deployment size and cost.

Concluding Remarks

While conducting the audits of Yearn Gold Finance smart contracts, it was observed that the contracts contain High, Medium, and Low severity issues, along with several areas of recommendations.

Our auditors suggest that High, Medium, Low severity issues should be resolved by Yearn Gold Finance's developers. Resolving the areas of recommendations are up to Yearn Gold 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 Yearn Gold 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.