QuillAudits

Audit Report
August, 2022

For

YearnAgnostic

Table of Content

Executive Summary

Project Name	Yearn Agnostic
Overview	YearnAgnostic Finance is a blockchain agnostic DeFi aggregator platform that supports decentralized finance (DeFi) projects deployed on Ethereum blockchain, Binance smart chain etc., focusing on simplicity, user experience, privacy and global adoption.
Timeline	May 4th, 2022 to August 5th, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Yearn Agnostic's NFT marketplace codebase for quality, security, and correctness. GitHub - *YFIAG/NFT-Marketplace-Contracts*
Commit	574fe954896a2c222eec3ed93ac4524716f4dd3a
Fixed In(Review)	*https://github.com/YFIAG/NFT-Marketplace-Contracts/tree/audit_%232*
Fixed In (Extra Review)	*https://github.com/YFIAG/NFT-Marketplace-Contracts/tree/audit_%233*

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	2	4	4
Partially Resolved Issues	0	0	1	1
Resolved Issues	6	7	11	10

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - YFIAGLaunchPad

High Severity Issues

No issues found

Medium Severity Issues

A1. Possible absence of NFT with RootId

```
662     // adds a new launchpad
663     function addLaunchPad(
664         string calldata name,
665         ERC20 stakeToken,
666         uint24 _weightAccrualRate,
667         uint256 _rootId,
668         uint256 _startTime,
669         uint256 _endTime,
670         uint104 _minTotalStake
671     ) external onlyOwner {
672         require(_weightAccrualRate != 0, "accrual rate = 0");
673         require(_endTime > _startTime, "Invalid time");
674         require(_endTime > block.timestamp, "Invalid time");
675 
676         // add launchpad
677         launchpads.push(
678             LaunchpadInfo({
679                 name: name, // name of launchpad
680                 stakeToken: stakeToken, // token to stake (e.g., YFIAG)
681                 weightAccrualRate: _weightAccrualRate, // rate of stake weight accrual
682                 rootIdToken: _rootId, // root id token
```

Description

It is possible that the rootId passed in the addLaunchPad() function may not exist. mintFragment() , which is called in the claim() function will never work unless the NFT with rootid is minted.

Remediation

The NFT corresponding to the launchPad with the specific rootId must be minted during the execution of addLaunchPad() function itself via call to the mint() function in YFIAGNftMarketPlace

Status

Resolved

A.2 Centralization of setWinners()

Description

setWinners() function is used directly to decide the winners who can claim rewards. But this function is an onlyOwner function, allowing only the owner to call this function. As a result an owner can set his own addresses as winners, or he can deny the potential deserving winners their rewards for a long time. This can lead to unfair distribution of rewards for stakers.

Remediation

Consider a mathematical model that is fair to decide the winners from stakers and use a push and pull model to distribute the rewards for the same.

Status

Acknowledged

Low Severity Issues

A3. Handle error when staking to non existent pool

```
719     function stake(uint24 launchpadId, uint104 amount) external nonReentrant {
720         // stake amount must be greater than 0
721         require(amount > 0, "amount is 0");
722
723         // require msg.sender is wallet
724         require(!_msgSender().isContract(), "Sender == contr address");
725
726         // get launchpad info
727         LaunchpadInfo storage launchpad = launchpads[launchpadId];
728
729         //check expired startTime
730         require(launchpad.startTime < block.timestamp,"staking time has !started");
731
```

Description

If user stakes into non existent pool with [#L719] stake() then transaction will revert with reverted with panic code 0x32 error. This is because a line of code tries to access data on a non-existent index on the launchpads array.

Remediation

Consider adding a modifier which checks if the entered launchpad exists or not.

Status

Resolved

A4. Unsafe to rely on isContract()

Description

[#L719] stake(), [#L757] unstake(), [#L818] claim() shows use of isContract().
It is unsafe to assume that an address for which this function returns false is an externally-owned account (EOA) and not a contract.Among others, `isContract` will return false for the following types of addresses:
- an externally-owned account
- a contract in construction
- an address where a contract will be created
- an address where a contract lived, but was destroyed

Also quoting Openzeppelin's comments on isContract()
"Preventing calls from contracts is highly discouraged. It breaks composability, breaks support for smart wallets like Gnosis Safe, and does not provide security since it can be circumvented by calling from a contract in constructor"

Recommendation

We recommend reviewing logic. In the case if the address is known for which check is being made, use a conditional statement to check if the sender address is the one for which the check is being made.

Status

Resolved

A5. Redundant code block

```
251         if (numFinishedSalesDelta == 0) {
252             // calculate normally without rollover decay
253
254             uint80 elapsedBlocks = blockNumber -
255                 closestUserCheckpoint.blockNumber;
256
257             stakeWeight =
258                 closestUserCheckpoint.stakeWeight +
259                 (uint192(elapsedBlocks) *
260                     launchpad.weightAccrualRate *
261                     closestUserCheckpoint.staked) /
262                 10**18;
263
264             return stakeWeight;
265         } else {
266             // calculate with rollover decay
267
268             // starting stakeweight
269             stakeWeight = closestUserCheckpoint.stakeWeight;
```

Description

numFinishedSalesDelta will always be zero and else block on line [#L265] won't execute because in contract numFinishedSales is only going to increase when the _bumpSaleCounter argument passed in addLaunchpadCheckpoint() is true but its never happening in contract. Hence numFinishedSalesDelta which is calculated by calculating the difference in closestLaunchpadCp.numFinishedSales and closestUserCheckpoint.numFinishedSales is going to be zero.

Remediation

Review code logic and consider removing redundent code block from [#L265] to [#L312].

Status

Resolved

A6. Missing Zero Address Check

```
function setWinners(uint24 launchpadId, address[] memory _winners) public onlyOwner() launchpadNotFound(launchpadId) nonRee
    // require launchpad is disabled
    require(!launchpadDisabled[launchpadId], "launchpad disabled");

    //set launchpad disable
    launchpadDisabled[launchpadId] = true;

    for(uint256 i=0; i< _winners.length; ++i){
        require(isStakers[launchpadId][_winners[i]], "!stakers");

        winners[launchpadId][_winners[i]] = true;
    }

}
```

```
891       function setAddressMarketplace(address _YFIAGNftMarketplace) public onlyOwner(){
892           YFIAGNftMarketplace = _YFIAGNftMarketplace;
893       }
```

```
663       function addLaunchPad(
664           string calldata name,
665           ERC20 stakeToken,
666           uint24 _weightAccrualRate,
667           uint256 _rootId,
668           uint256 _startTime,
669           uint256 _endTime,
670           uint104 _minTotalStake
671       ) external onlyOwner {
672           require(_weightAccrualRate != 0, "accrual rate = 0");
673           require(_endTime > _startTime, "Invalid time");
674           require(_endTime > block.timestamp, "Invalid time");
675
676           // add launchpad
```

Description

Contracts lack zero address checks, hence are prone to be initialized with zero addresses.

[#L136] constructor() lacks zero address check for _YFIAGNftMarketplace
[#L803] setWinners() lacks zero address check for addresses in _winners[]
[#L891] setAddressMarketplace() lacks zero address check for _YFIAGNftMarketplace
[#L663] addLaunchPad lacks zero address check stakeToken.

Remediation

Consider adding zero address checks for the same.

Status

Resolved

A7. Denial of service to the user while claim NFT

```
804         //get amount refund
805         uint104 amountRefund = checkpoint.staked - launchpad.minTotalStake;
806
807         // check staked
808         require(amountRefund > 0, "staked < minTotalStake");
809
810         // set balance launchpad
811         balanceOfLaunchpad[launchpadId] += launchpad.minTotalStake;
812
813         // transfer amountRefund for sender
814         if(amountRefund > 0){
815             launchpad.stakeToken.safeTransfer(_msgSender(), amountRefund);
816         }
```

Description

The user can only stake an amount that is greater than or equal to launchpad.minTotalStake because of the check added in addUserCheckpoint() on line 449 . so if user stakes amount that is equal to minTotalStake then amountRefund would be 0 and this require check will fails because of which a user cant claim a NFT even after staking minimum amount required.

Recommendation

Review business logic and remove the require statement, doing this will allow user to claim NFT who has staked the token amount equal to minTotalStake for that launchpad.

Status

Resolved

Informational Issues

A8. Redundant Statements

```
847         //get amount refund
848         uint104 amountRefund = checkpoint.staked - launchpad.minTotalStake;
849
850         // check staked
851         require(amountRefund >= 0, "staked < minTotalStake");
852
853         // set balance launchpad
854         balanceOfLaunchpad[launchpadId] += launchpad.minTotalStake;
855
```

Description

[#L851] amountRefund is always going to be 0 or greater than 0 because it is declared as an unsigned integer. Highlighted line shows the use of the require statement to check the amountRefund is 0 or greater than 0.

Remediation

Remove redundant statements

Status

Resolved

A9. Modifier launchpadNotFound checks <= instead of only < sign

```
129     modifier launchpadNotFound(uint24 launchpadId){
130         require(launchpadId <= launchpads.length, "LP isn't exist");
131         _;
132     }
```

Description

launchpadNotFound modifier checks that entered launchpadId should be less than or equal to launchpads.length. In this case if launchpads.length is 5 and if we enter 5 then this check will get passed but in reality there doesn't exist any launchpad with id 5 because. There would be launchpads from id 0 to 4 (0 to 4 i.e 5 IDs).

Remediation

Review logic and remove = from condition statement.

Status

Resolved

A10. Unnecessary use of nonReentrant modifier

```
803     function setWinners(uint24 launchpadId, address[] memory _winners) public onlyOwner() launchpadNotFound(launchpadId) nonReentrant{
804         // require launchpad is disabled
805         require(!launchpadDisabled[launchpadId], "launchpad disabled");
806 
807         //set launchpad disable
808         launchpadDisabled[launchpadId] = true;
809 
810         for(uint256 i=0; i< _winners.length; ++i){
811             require(isStakers[launchpadId][_winners[i]], "!stakers");
812 
```

Description

[#L803]setWinners() is using nonReentrant modifier , In the current case of setWinners() function it seems unnecessary to use nonReentrant as this function doesn't have any external calls.

Remediation

Consider removing the nonReentrant modifier for setWinners().

Status

Resolved

A11. Unnecessary use of storage keyword

Description

Some functions use the storage keyword to store where memory can be used to save gas. Lines on these lines shows usage of storage keyword [#L480] [#L569] [#L707] [#L727] [#L765] [#L826] [#L878] [#L927]

Remediation

Use memory instead of storage when possible

Status

Resolved

A12. Incorrect comment and error message

```
446         // if this is first checkpoint
447         if (nCheckpointsUser == 0) {
448             // check if amount exceeds maximum
449             require(amount >= launchpad.minTotalStake, "exceeds staking cap");
450
```

Description

addUserCheckpoint() contains code for checking if the amount that the user is staking is greater than or equal to launchpad.minTotalStake, But the comment and error message for require statement don't matches the code, which can create confusion.

Remediation

Review business logic and change error message and comment which will match the code actions to avoid confusion.

Status

Resolved

A13. Incorrect launchpad count

Description

deleteLaunchpad() launchpad doesn't deletes launchpad from launchpads array and that's why launchpadCount() will return incorrect amount of current launchpad present when any launchpad is deleted.

Remediation

make sure that this count is not getting used for any critical operation and delete the launchpad id element from the launchpads array that is getting deleted by deleteLaunchpad() if required.

Status

Resolved

B. Contract - YFIAGNftMarketplace

High Severity Issues

B.1 Part A- User's Ether can get stuck in the contract

```
324         // buy native
325         if(tokenAddress[_tokenId] == address(0)) {
326             require(prices[_tokenId] == msg.value, "Value != price!");
327             bool flag;
328             payable(ownerOf(_tokenId)).transfer(_price.sub(_creatorRoyalty + _platformFee + _subOwnerFee));
329             if(_creatorRoyalty > 0){
330                 payable(tokenCreators[_tokenId]).transfer(_creatorRoyalty);
331             }
332             payable(platformFeeAddress).transfer(_platformFee);
333             _pause(_tokenId);
334             _transfer(ownerOf(_tokenId), msg.sender, _tokenId);
```

Description

The if statement on line: 325 executes only when the tokenAddress[_tokenId] == address(0) This means it will fail to execute for any NFT that has a non-zero tokenAddress which can be set using mint() function. As a result if NFTs with non-zero tokenaddresses are listed for sale by their owners, it will result in any ether getting stuck in the smart contract that is sent when calling the buy() function.
For example, Say the price of an NFT is set as 1 ETH and this NFT has a non zero tokenaddress. Then if a buyer calls the buy function by passing 1 ETH, the function will execute except for the if statement on line: 325 and the 1 ETH will get stuck in the YFIAGNftMarketplace contract. The buyer will lose his funds and also he will not get any ownership of the NFT. Moreover, it is possible that the buyer can also send ETH greater than 1 ETH such as 3 ETH and all the 3 ETH will get stuck as well.
The same problem as above is also in the buyAndBurn() function. The if statement on line: 379 executes only when the tokenAddress[_tokenId] == address(0) This also means it will fail to execute for any NFT that has a non-zero tokenAddress which can be set using mint() function. The only difference here is that the admin will be able to recover any of his funds that get stuck in the contract using the withdraw() function.

Remediation

Consider reviewing the smart contract and the business logic. It is also advised to add the code to refund the buyer all his funds in case the above scenario arises.

Status

Resolved

B.2 Part B- Malicious admin can withdraw all stuck ether

```
189     function withdraw() external override onlyOwner() {
190         payable(owner()).transfer(getBalance());
191     }
192
193     function withdraw(address _user, uint256 _amount) external override onlyOwner() {
194         uint256 _balance = getBalance();
195         require(_balance > 0, "Balance is null");
196         require(_balance >= _amount, "Balance < amount");
197
198         payable(_user).transfer(_amount);
199     }
200
```

Description

If there is any ether stuck in the contract such as in the partA of issue B.1, a malicious admin can use the withdraw() function to withdraw all the ether stuck in the contract without refunding any ether to the corresponding buyers whose funds were stuck.

Remediation

Consider adding required code and checks to avoid the above scenario and refund all the ether to its user in case this happens.

Status

Resolved

B.3 Non-owner can burn any NFT

```
229     function burnByLaunchpad(address account,uint256 _tokenId) external override tokenNotFound(_tokenId) onlyAdmin(){
230         require(_rootTokens[_tokenId], "isn`t root");
231         _burn(account,_tokenId);
232     }
233
234     function burn(uint256 _tokenId) external override tokenNotFound(_tokenId) isRootToken(_tokenId) isFragments(_tokenI
235         require(ownerOf(_tokenId) == msg.sender, "isn`t owner of token");
236         _burn(msg.sender, _tokenId);
237     }
238
```

Description

Admin can burn any NFT which is a root token and which the admin does not himself own. Which can be done using the burnByLaunchPad() function on line: 229 which an admin can call. This can be detrimental if the root tokens that the admin currently does not own have a significant value.

For example, Say the admin mints NFTs with tokenId 3 and 4 which are also root tokens. If tokens are directly minted to another address say wallet1 (or they are transferred by the admin to this account) and these tokens have 1 Ether value each in the secondary or third party market, then the admin can burn these NFTs resulting in 2 Ethers of loss to the person with wallet1(who is actually the real owner)

Remediation

Add a require check to allow burning of token only when the owner of the token is the same as the msg.sender

Status

Resolved

Note: Solution implemented for only Launchpad contract can burn root tokens

B.4 setPool will always revert

```
240     function setPool(address pool) public onlyOwner(){
241         require(yfiagPool != address(0),"Bad pool");
242         require(yfiagPool.isContract(),"Not contract");
243         // transfer all fund to new pool
244         IYFIAGNftPool(yfiagPool).migratePool(pool, address(0));
245         yfiagPool = pool;
246         emit SetNewPool(pool);
247     }
```

Description

setPool() function will always revert with “Bad pool” because setPool() is performing zero address check for yfiagPool variable and not for the pool argument , The yfiagPool would be zero address initially and condition on the line 241 will fail which will revert the transaction.

It will revert on line 244 also with error message function call to a non-contract account because yfiagPool variable would be storing zero address initially.

Remediation

For the 1st issue mentioned above - replace yfiagPool on the line 241 and 242 with pool parameter.
For the 2nd issue - initially while deploying contract yfiagPool address can be set in constructor so it can access migratePool() function inside yfiagPool contract while setting pool address with setPool() next time.
Ensure unit testing to avoid this type of issues.

Status

Resolved

Medium Severity Issues

B.5 Part A- Denial of Service via setDefaultAmountEarn()

```
239     function setDefaultAmountEarn(address _user, address _tokenAddress) external override{
240         require(msg.sender == YFIAGPool, "Isn`t Pool");
241         amountEarn[_user][_tokenAddress] = 0;
242     }
243
244     function setPlatformFeeAddress(address newPlatformFeeAddess) external override onlyAdmin(){
245         platformFeeAddress = newPlatformFeeAddess;
246     }
247
```

Description

In the setPool() function there is no check to see whether the pool address is a contract address or not. It is possible that an Externally Owned Account (EOA) is set as a pool address. A malicious EOA can exploit setDefaultAmountEarn() and set the amountEarn[_user][_tokenAddress] equal to zero for any user's tokens thus wiping out the rewards of the users and Denying them their rewards.
Also a malicious admin can change the address of the NFTPool contract to point to a contract which does not allow the subOwners to withdraw their subownerfees leading to another potential Denial of Service attack.

Remediation

It is advised to add a check to ensure that the pool address is not an EOA and is a contract. It should be noted that although isContract() can be used to check whether an address is a contract or not, it does not guarantee that an address is not a contract address if it returns false for the following types of addresses:
- an externally-owned account
- a contract in construction
- an address where a contract will be created
- an address where a contract lived, but was destroyed

Ideally hardcoding a pool address for the Marketplace contract using an immutable variable would be a best practice in terms of security.

Status

Resolved

B.6 Part B- Potential Denial of Service via setPool()

```
220     function setPool(address pool) public onlyOwner(){
221         YFIAGPool = pool;
222     }
223
224     function setLaunchPad(address launchPad) public onlyOwner(){
225         _launchPad = launchPad;
226         setAdmin(launchPad, true);
227     }
228
```

Description

Also an admin can change the addresses of the Pool before existing subowners can withdraw their fees from the pool which can lead to potential Denial of Service attacks for existing subowners.

For example, if the existing subOwners have 0.1 ETH in total as fees which they can withdraw from the NFTPool, and an admin changes the address of the pool, the existing subowners will not be able to access the previous NFTPool contract if they do not know the address of the contract old NFTPool contract (the YFIAGNftMarketplace's YFIAGPool will be pointing to the new pool's address instead of the old pool one.)

Remediation

It is advised to keep a track of all the NFTPool addresses that have been set till now or not allow the pool address to be changed unless all the subowners have withdrawn their fees from

Status

Resolved

B.7 Uninitialized YFIAGPool and _launchpad

```
42      address public YFIAGPool;
43
44
45
46      constructor() ERC721("YFIAG NFT", "YNFT") {
47          tokenId = 1;
48          platformFeeAddress = msg.sender;
49      }
```

Description

YFIAGPool and _launchpad will be zero addresses if they are not set manually or are forgotten to set using the setPool() and setLaunchPad() functions. For example, any fees sent to the pool on line: 360 in buy function will get burnt as it will be sent to zero address instead

Remediation

It is advised to set these addresses in the constructor itself to avoid such issues.

Status

Acknowledged

B.8 For loop over Dynamic array

```
351                 for(uint256 i=0; i< _subOwners[_rootId].length; ++i){
352                     if(_subOwners[_rootId][i] != _owner){
353                         address _subOwner = _subOwners[_rootId][i];
354                         amountEarn[_subOwner][tokenAddress[tokenId]] += _amountEarn;
355                         flag= true;
356                     }
357                 }
```

Description

The length of the _subOwners array can increase when admin or launchPad mints NFTs via the mintFragment() function.
As a result, the length of the _subOwners array for a particular rootId can grow. If it grows very large it can lead to out-of gas errors and consistently failed execution of the buy() function (for this particular rootId) due to the for loop that has to loop over a large _subOwners array

Remediation

Consider adding a require check on the maximum length allowed for the _subOwners array in the buy function or add a maximum limit allowed for _subOwners array in mintFragment itself.

Status

Resolved

B9. Missing Zero Address Check

```
220       function setPool(address pool) public onlyOwner(){
221           YFIAGPool = pool;
222       }
223
224       function setLaunchPad(address launchPad) public onlyOwner(){
225           _launchPad = launchPad;
226           setAdmin(launchPad, true);
227       }
```

```
244       function setPlatformFeeAddress(address newPlatformFeeAddess) external override onlyAdmin(){
245           platformFeeAddress = newPlatformFeeAddess;
246       }
247
```

```
268   function mintByCrosschain(address _to,address _token, string memory _uri, uint256 _royalty, address _creator) external override onlyAdmin(){
269       require(_token == address(0) || _token.isContract(), "Token isn`t a contract address");
270       require(_royalty <= maxRoyalties && _royalty >= minRoyalties, "Royalty wrong");
271
272       _safeMint(_to, tokenId, 0, _uri);
273
274       tokenAddress[tokenId] = _token;
275       royalties[tokenId] = _royalty;
276       tokenCreators[tokenId] = _creator;
277       creatorsTokens[_creator].push(tokenId);
278       tokenId++;
279   }
```

Description

Contracts lack zero address checks, hence are prone to be initialized with zero addresses.

[#L220] setPool lacks zero address check for pool
[#L224] setLaunchPad lacks zero address check for launchPad
[#L244] setPlatformFeeAddress lacks zero address check for newPlatformFeeAddess which can lead to burning of fees
[#L268] mintByCrosschain lacks zero address check for _creator which can be misleading as minted tokenIds of NFTs will be mapped to zero address
[#193] withdraw lacks zero address check for _user which can lead to burning of funds
[#201] withdraw lacks zero address check for _user which can lead to burning of funds

Remediation

Consider adding zero address checks.

Status

Resolved

B10. Misleading name/execution of burnByLaunchPad()

Description

The function burnByLaunchpad() can also be called by admin, not just Launchpad because of the onlyAdmin() modifier defined in the modified ERC721 contract. Thus the name of the function burnByLaunchpad() suggests that the function can only be called by the Launchpad contract but it is entirely possible for the admin himself to call this function, which can be misleading.

Remediation

It is suggested to add a different modifier for this function so that only the LaunchPad contract can call this function and not the admin directly to stay consistent with the intended naming and execution.

Status

Resolved

B11. Misleading name/execution of burnByLaunchPad()

```
229     function burnByLaunchpad(address account,uint256 _tokenId) external override tokenNotFound(_tokenId) onlyAdmin(){
230         require(_rootTokens[_tokenId], "isn't root");
231         _burn(account,_tokenId);
232     }
```

Description

After burning the Root NFT, although the NFT with the rootId gets burned, their fragments still remain. The allFragmentOf function still shows that the corresponding NFTs are still the fragments of the now non-existent rootId.
For example, if NFT with tokenId 9 is a root NFT, and it has fragments with tokenId as 10, 11 and 12. Now when the root NFT with tokenId 9 is burnt, its fragments still remain. If the allFragmentOf function is called by passing 9 as the tokenId, the array elements with tokenId 10, 11 and 12 will be returned. But this is contradictory to the fact that NFT with tokenId 9 does not exist and is not a root anymore. And logically the NFTs with tokenIds 10, 11 and 12 cannot be called as fragments of the now non-existent root token with tokenId 9.

Remediation

Clear the corresponding mapping of allFragmentOf as it can be misleading and review the business logic.

Status

Resolved

B12. Transaction Order Dependence in setPriceAndSell() and buy()

```
294     function setPriceAndSell(uint256 _tokenId, uint256 _price) public override tokenNotFound(_tokenId) isRootToken(_tokenId){
295         require(ownerOf(_tokenId) == msg.sender, "isn't owner of token");
296
297
298         prices[_tokenId] = _price;
299         _resume(_tokenId);
300
301         emit PriceChanged(_tokenId, _price, tokenAddress[_tokenId], msg.sender);
302     }
```

```
304     function buy(uint256 _tokenId) public payable override tokenNotFound(_tokenId) isRootToken(_tokenId){
305         require(tokenStatus[_tokenId], "Token not for sale");
306         require(ownerOf(_tokenId) != msg.sender, "already owner of token");
307         // require msg.sender is wallet
308         require(!_msgSender().isContract(), "Sender == contr address");
309
310         address _prevOwner = ownerOf(_tokenId);
311         uint256 _rootId = _rootIdOf[_tokenId];
312         uint256 _prevLenght = _subOwners[_rootId].length;
313         uint256 _price = prices[_tokenId];
314         uint256 _creatorRoyalty = 0;
315         uint256 _platformFee = (_price.mul(platformFee)).div(10000);
316         uint256 _subOwnerFee = 0;
317         if(_fragmentTokens[_tokenId] && _fragments[_rootId].length > 1){
318             _subOwnerFee = (_price.mul(royalties[_tokenId])).div(10000);
319         }
```

Description

When the seller sets the price of the NFT again (i.e. updates it), a buyer can frontrun his own transaction to buy the NFT at a possibly cheaper price.
Example for para 1-
- Seller sets price of nft to 1 ETH
- Seller decides to change its price to 2 ETH and send a transaction for it
- A Buyer monitoring the mempool sees the transaction of the seller
- Buyer frontruns his transaction to buy the NFT at 1ETH

Additionally if normal users have sent the transaction to buy the NFT at old price, but the transaction to update the price gets executed first then all the transactions of the normal users will fail regardless.

Remediation

Consider using a commit-reveal scheme in which the seller has to first commit the tokenId, his address and salt and then submit another transaction to change the price of the NFT and revealing his details. This will prevent any front-running scenarios as no one will be able to see the tokenId of the NFT being changed unless it is revealed which the commit-reveal scheme ensures.

Status

Acknowledged

Informational Issues

B13. Missing events for Critical functions

Description

Missing events for functions carrying out critical operations- setPlatformFee(), setDefaultRoyalties(), setPool(), setLaunchPad(), setDefaultAmountEarn(), setPlatformFeeAddress(), mint(), mintByCrosschain() - can emit an event that also tracks royalty and creator, mintFragment(), buy()

Remediation

Consider adding and emitting events for the above functions. For the mint() function, an event that also emits the royalty and isRoot value passed into the function can be added. For the mintByCrosschain() function, an event that also tracks royalty and creator can be added.

Status

Resolved

C. Contract - ERC721 (Modified)

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

C1. Misleading name of isRootToken() and isFragments()

```
100     modifier isRootToken(uint256 _tokenId) {
101         require(!_rootTokens[_tokenId], "is root");
102         _;
103     }
104
105     modifier isFragments(uint256 _tokenId){
106         require(!_fragmentTokens[_tokenId], "is fragments");
107         _;
108     }
109
```

Description

Ideally the modifier isRootToken() should be named as isNotRootToken(), beacause the require statement checks that the unavailability or falseness of _rootTokens mapping and not its presence.

Also the modifier isFragments() should be named as isNotFragments(), because the require statement checks that the unavailability or falseness of _fragmentTokens mapping and not its presence. Such incorrect naming could lead to improper usage and incorrect assumptions.

Remediation

It is advised to rename the modifiers as mentioned above.

Status

Resolved

Informational Issues

C2. Usage of ERC721Enumerable pattern is costly

Description

ERC721 Enumerable contract's pattern has been used in the Modified ERC721 contract. This contract is highly costly in terms of gas costs and the user has to pay a large amount of gas each time the token is minted or transferred.

Remediation

The contract can be optimised to make it more gas efficient. Checkout the ERC721A contract which allows multiple mints to a single address the same in gas fees as for minting a single NFT to the same address.
Refer- *https://blockmagnates.com/bored-ape-yacht-club-bayc-would-have-saved-users-over-80-million-in-gas-fees/*
https://medium.com/coinmonks/comparison-of-the-most-popular-erc721-templates-b3614353e31e

Status

Acknowledged

C3. Missing zero address check in setAdmin()

```
416     function setAdmin(address _user, bool _isAdmin) public onlyOwner() {
417         _admins[_user] = _isAdmin;
418
419         emit AdminSet(_user, _isAdmin);
420     }
421
```

Description

setAdmin() function's _user parameter lacks zero address check thus even a zero address can become an admin.

Remediation

Consider adding a zero address check for the same to avoid this issue.

Status

Resolved

D. Contract - YFIAGNftPool

High Severity Issues

D1. Admin can withdraw all funds and Deny subowner fees

Description

The function withdrawAdmin() on line: 58 allows the admin to withdraw all Ether at any point of time, thus withdrawing the funds that are meant only for the subowners(sent via MarketPlace contract) , leading to a potential denial of service.

Moreover, the function also allows the admin to withdraw any ERC20 accepted by this contract at any point of time.

Remediation

Consider reviewing the business logic and allowing withdrawal of funds to admin only in very specific scenarios, not otherwise. Remove the code to withdraw any ether from the admin if it is not really necessary.

Status

Resolved

D2. Broken Functionality

Description

setPool() in YFIAGNftMarketplace calls migratePool() function present in YFIAGNftPool contract, But migratePool() uses onlyOwner modifier and that's why call from any other address apart from owner’s address would be reverted. Hence a call from YFIAGNftMarketplace:setPool() to migratePool would revert.

Remediation

Consider removing onlyOwner modifier from migratePool() , and add a require check which will check the call is only from YFIAGNftMarketplace address to ensure security.

Status

Resolved

Medium Severity Issues

D3. Use of transferFrom instead of transfer

```
38      function withdraw(address _tokenAddress) external override nonReentrant {
39          uint256 subOwnerFee = YFIAGMKT.getAmountEarn(msg.sender, _tokenAddress);
40          if(_tokenAddress == address(0)){
41              require(subOwnerFee > 0, "Earn = 0");
42              require(address(this).balance >= subOwnerFee, "Balance invalid");
43              payable(msg.sender).transfer(subOwnerFee);
44              amountWithdrawn[msg.sender][_tokenAddress] += subOwnerFee;
45              YFIAGMKT.setDefaultAmountEarn(msg.sender, _tokenAddress);
46          }else{
47              require(subOwnerFee > 0, "Earn = 0");
48              require(IERC20(_tokenAddress).balanceOf(address(this)) >= subOwnerFee, "Balance invalid");
49              IERC20(_tokenAddress).transferFrom(address(this), msg.sender, subOwnerFee);
50              amountWithdrawn[msg.sender][_tokenAddress] += subOwnerFee;
51              YFIAGMKT.setDefaultAmountEarn(msg.sender, _tokenAddress);
52          }
53      }
```

Description

[#L38] withdraw() has functionality to withdraw ERC20 tokens. When withdraw() called with non zero token address then else code blocks executes to transfer ERC20 tokens to msg.sender

On [#L49] it uses transferFrom instead of transfer function, which will revert with the error ERC20: insufficient allowance.

Remediation

Consider using transfer insead of transferFrom.

Status

Resolved

D4. Use of transferFrom instead of transfer

```
65      function migratePool(address newPool, address _tokenAddress) public onlyOwner() {
66          require(newPool != address(0),"Bad address");
67          require(newPool.isContract(),"Not contract");
68          if(getBalance() > 0){
69              payable(newPool).transfer(getBalance());
70          }
71          if(IERC20(_tokenAddress).balanceOf(address(this)) > 0){
72              uint256 amount = IERC20(_tokenAddress).balanceOf(address(this));
73              IERC20(_tokenAddress).transferFrom(address(this), newPool, amount);
74          }
75      }
```

Description

migratePool() contains code to transfer ERC20 tokens in second if block but it uses ERC20 transferFrom() function instead of ERC20 transfer() , which will fail with insufficient balance error.

Remediation

Consider using ERC20 transfer in place of transferFrom.

Status

Resolved

D5. YFIAGNftPool can’t receive native tokens

```
65      function migratePool(address newPool, address _tokenAddress) public onlyOwner() {
66          require(newPool != address(0),"Bad address");
67          require(newPool.isContract(),"Not contract");
68          if(getBalance() > 0){
69              payable(newPool).transfer(getBalance());
70          }
71          if(IERC20(_tokenAddress).balanceOf(address(this)) > 0){
72              uint256 amount = IERC20(_tokenAddress).balanceOf(address(this));
73              IERC20(_tokenAddress).transferFrom(address(this), newPool, amount);
74          }
75      }
```

Description

migratePool() is sending native tokens to newPool contract , but newPool contract relies on subOwnerFeeBalance payable function to receive native tokens like BNB/ETH. The transaction to send native tokens to newPool will revert as it dont contains any fallback or receive external payable function.

Remediation

Add receive() external payable { } function in (new) YFIAGNftPool contract to receive native tokens or send them using subOwnerFeeBalance() payable.
Note: Currently pool contract is using subOwnerFeeBalance() payable function to receive native tokens (subowner fees) sent by YFIAGNftMarketplace:buy() , care needs to be taken if its removed after adding receive() external payable { }.

Status

Resolved

Low Severity Issues

No issues found

Informational Issues

D6. Missing Zero Address Check

Description

Contract lacks zero address checks, hence are prone to be initialized with zero addresses.

Constructor lacks a zero address check for _YFIAGNftMarketplace.
[#L67] setMarketplaceAddress lacks zero address check for marketPlaceAddress

Remediation

Consider adding zero address checks.

Status

Partially Resolved

Note: Constructor still lacks zero address check.

D7. Redundant Function

Description

[#L30] getAmountEarn() is defined in this YFIAGNftPool contract which interacts with getAmountEarn() present in YFIAGNftMarketplace contract.
Instead of defining new getAmountEarn() in YFIAGNftPool, getAmountEarn present in YFIAGNftMarketplace contract can be used.

Remediation

Consider removing redundant function.

Status

Acknowledged

D8. Unused libraries

```
4   import "./utils/Address.sol";
5   import "./utils/SafeMath.sol";
6   import "./interfaces/IYFIAGNftMarketplace.sol";
7   import "./interfaces/IERC20.sol";
8   import "./utils/Ownable.sol";
9   import "./interfaces/IYFIAGNftPool.sol";
10  import "./utils/ReentrancyGuard.sol";
11
12  contract YFIAGNftPool is IYFIAGNftPool, Ownable, ReentrancyGuard{
13      IYFIAGNftMarketplace public YFIAGMKT;
14      using Address for address;
15      using SafeMath for uint256;
16
```

Description

SafeMath and Address libraries are imported on the [#L4], [#L5] and using directive is used to attach functions from these libraries to address and uint256 type on [#L14],[#L15] but functions from these contracts are never used.

Remediation

Consider removing import statements from the contract if libraries are not getting used.

Status

Resolved

Note: Current contract is using a function from the Address library.

D9. Unnecessary use of nonReentrant modifier

```
58      function withdrawAdmin(address _tokenAddress) external override onlyOwner() nonReentrant {
59          if(_tokenAddress == address(0)){
60              payable(owner()).transfer(getBalance());
61          }else{
62              uint256 amount = IERC20(_tokenAddress).balanceOf(address(this));
63              IERC20(_tokenAddress).transferFrom(address(this), msg.sender, amount);
64          }
65      }
```

Description

[#L58] withdrawAdmin uses nonReentrant modifier. This function is owner only function and can be called by only the owner which eliminates risk of reentrancy.

Remediation

Consider not using nonReentrant modifier for withdrawAdmin.

Status

Resolved

Note: withdrawAdmin() removed.

D10. Redundant statements

```
65      function migratePool(address newPool, address _tokenAddress) public onlyOwner() {
66          require(newPool != address(0),"Bad address");
67          require(newPool.isContract(),"Not contract");
68          if(getBalance() > 0){
69              payable(newPool).transfer(getBalance());
70          }
71          if(IERC20(_tokenAddress).balanceOf(address(this)) > 0){
72              uint256 amount = IERC20(_tokenAddress).balanceOf(address(this));
73              IERC20(_tokenAddress).transferFrom(address(this), newPool, amount);
74          }
75      }
```

Description

setPool() in YFIAGNftMarketplace contract calls migratePool() in YFIAGNftPool contract, setPool() checks for zero addres check and if address is contract or not. The migratePool() in YFIAGNftPool contract also contains these checks which can be removed as these checks are gettig performed in setPool().

Remediation

Remove check to see if address is contract or not and Zero address check.

Status

Acknowledged

E. Common issues

High Severity Issues

E1. Private key visible

Description

The initial code that the YFIAG team had provided at commit 574fe95 contained private key in one of its test files and was visible due to the repo being public

Remediation

It is advised to remove the private key and use a new account altogether instead of the account corresponding to the private key

Status

Resolved

Auditor's Comment: Although this issue is outside of its intended scope, we quickly reported it to the Yearn Agnostic team after discovering it during our audit, and the Yearn Agnostic Team indicated that the private key is getting used only for testing purpose.

Medium Severity Issues

No issues found

Low Severity Issues

E2. Renounce Ownership

Description

The modified ERC721 contract, YFIAGLaunchPad, YFIAGNftBridgeTreasury and YFIAGNftPool inherit the Ownable contract which contains the renounceOwnership() function. The renounceOwnership function can be called accidentally by the owner leading to immediate renouncement of ownership to zero address after which any onlyOwner functions will not be callable which can be risky.

Remediation

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Refer this post for additional info- *https://www.linkedin.com/posts/razzor_github-razzorsecrazzorsec-contracts-activity-6873251560864968705-HOS8*

Status

Acknowledged

E3. Poor Test coverage

Description

There is poor test coverage by the team for the contracts which can lead to a substantial number of undiscovered bugs.

Remediation

It is advised to do thorough unit testing of the contracts and do a good test coverage.

Status

Acknowledged

E4. Redundant code for handling ERC20 tokens

Description

Contracts contain code for handling ERC20 tokens in some functions
Such as:
- YFIAGNftPool: migratePool() contains code in the if block to handle erc20 token transfer.
- YFIAGNftPool: withdraw() takes the _tokenAddress parameter and contains code in the else block to handle erc20 token transfer.
- YFIAGNftMarketplace:mint() takes erc20 _token parameter

but the project doesn't support ERC20 tokens for buying NFTs.

Remediation

The redundant code that is not getting used should be removed. Care needs to be taken while removing redundant code to check if it is not breaking other functionality.

Status

Acknowledged

Informational Issues

E5. Unlocked pragma (pragma solidity ^0.8.0)

Description

YFIAGLaunchPad uses solidity ^0.8.0 , YFIAGNftMarketplace and YFIAGNftPool uses >=0.8.4 <=0.8.6 version.
Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version get selected while deploying contract which has higher chances of having bugs in it.

Remediation

Lock the pragma version for the compiler version that is chosen while testing and debugging to ensure that the same compiler version is used during deployment.

Status

Resolved

Functional Testing

YFIAGLaunchPad

- ✓ setAddressMarketplace reverts if not called by owner
- ✓ setAddressMarketplace sets marketplace address
- ✓ setWinners reverts if not a owner
- ✓ setWinners reverts if launchpad not found
- ✓ setWinners reverts if launchpad is disabled
- ✓ Should be able to set winner addresses
- ✓ addLaunchPad adds launchpad
- ✓ addLaunchPad reverts if accrual rate is 0
- ✓ addLaunchPad reverts if endtime is less than starttime
- ✓ addLaunchPad reverts if endtime is less than current timestamp
- ✓ Owner should be able to disable launchpad
- ✓ stake reverts if the amount is zero
- ✓ stake reverts if the launchpad is disabled
- ✓ Should be able to stake tokens
- ✓ Should be able to unstake tokens if launchpad is disabled
- ✓ Winner should be able claim nft
- ✓ Owner should be able to delete launchpad
- ✓ Owner should be able to withdraw

YFIAGNftMarketPlace

- ✓ Should be able to mint NFT with corresponding RootId
- ✓ Should be able to mintFragments of RootIds
- ✓ User should be able to buy NFT for sale using buy()
- ✓ Subowners should be added when calling mintFragment()
- ✓ Subowners should get fees after user buying NFT
- ✓ NFTPool should be able to receive Ether as fees
- ✓ buyAndBurn should work as intended
- ✓ Should not allow paused NFT to be bought
- ✓ Modifiers should work as intended
- ✓ Bypassing require checks should not be possible

YFIAGNftPool

- ✓ Should be able to withdraw ETH
- ✓ Should be able to get the amount an address earned by getAmountEarn
- ✓ Should be able to get the amount withdrawn by address
- ✓ Should be able to set marketplace address
- ✓ Reverts if unauthorized address tries to withdraw ETH
- ✓ Reverts if caller reenters while withdrawing

YFIAGNftBridgeTreasury

- ✓ Should be able to pay
- ✓ Owner should be able to withdraw ETH
- ✓ Reverts if unauthorized address tries to withdraw
- ✓ Should be able to set marketplace address
- ✓ Reverts if an unauthorized address tries set address
- ✓ Should be able to refund ETH
- ✓ Reverts if amount entered is greater than current contract balance
- ✓ Reverts if unauthorized address tries to refund

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Yearn Agnostic. We performed our audit according to the procedure described above.

Issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Yearn Agnostic Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Yearn Agnostic Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
August, 2022

For

YearnAgnostic

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com