hexens × Fungify.

JUNE.24

SECURITY REVIEW REPORT FOR FUNGIFY

CONTENTS

- The fee calculation in NftIndex.redeem() is not accurate
- Missing limit checks in fee and buffer setting functions
- Unnecessary receive function in NftIndex
- No event is emmitted on Upgradeable.replaceImplementation()
- An incorrect assertion about totalVaultLiq in the function removeAsset()
- Proxies duplicate the implementation functionality
- No storage gaps in the inherited contracts
- Proxies use the standard storage layout instead of the namespaced
- Consider utilizing internal calls to retrieve the total supply and user balances
- Unused errors
- Unused weth variable in NftIndex

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

EXECUTIVE SUMMARY

OVERVIEW

This audit reviewed a specific commit for Fungify, a lending protocol built on Compound that facilitates the lending and borrowing of NFTs. The commit concentrated on the NFT Index, a token that is backed by and can be redeemed for NFT collections. The pricing is denominated in NFT Token, based on the floor value of the collections as determined by the Oracle. Through arbitrage, this ensures that the price of $NFT is pegged to the floor values of the respective NFTs.

Our security assessment was a review of the smart contracts changes in this commit, spanning a total of 2 weeks

During our audit, we identified four high-severity vulnerabilities that could put the protocol and user funds at risk. Additionally, we found five medium-severity issues, nine low-severity issues, and three informational issues.

Finally, all of our reported issues were fixed or acknowledged by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

SCOPE

The analyzed resources are located on:

https://github.com/fungify-dao/fungify-protocol-refactor/tree/383957ea22a920b488171df28da5e0eb0d2541a2

The issues described in this report were fixed. Corresponding commits are mentioned in the description.

AUDITING DETAILS

STARTED
10.06.2024

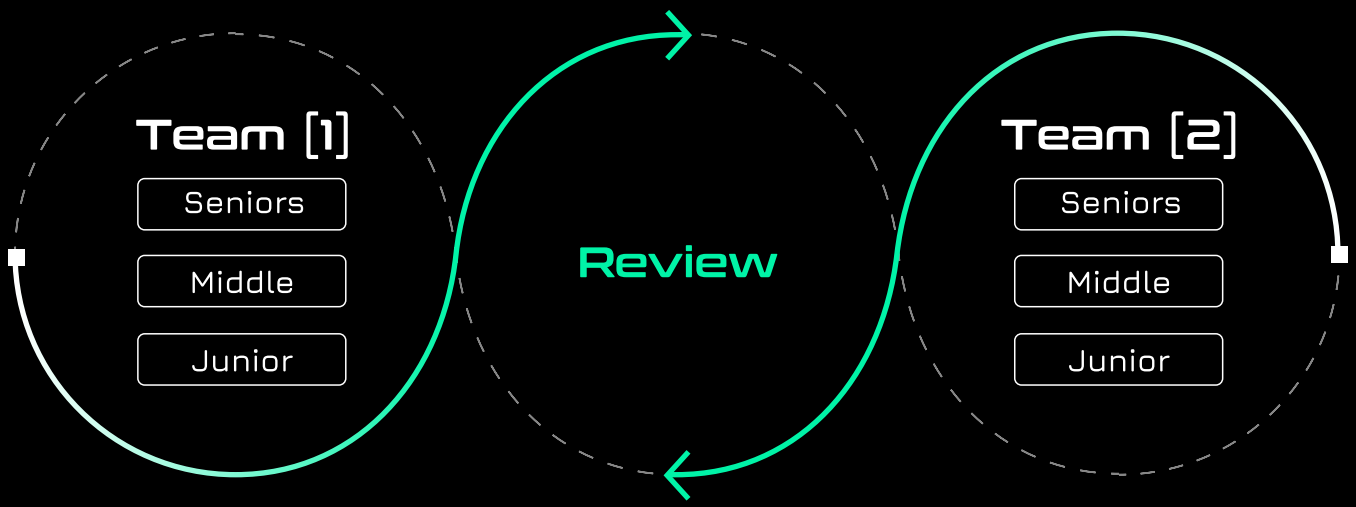
DELIVERED
25.06.2024

Review Led by

TRUNG DINH
Senior Security Researcher | Hexens

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

Impact	Probability			
	rare	unlikely	likely	very likely
Low/Info	Low/Info	Low/Info	Medium	Medium
Medium	Low/Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

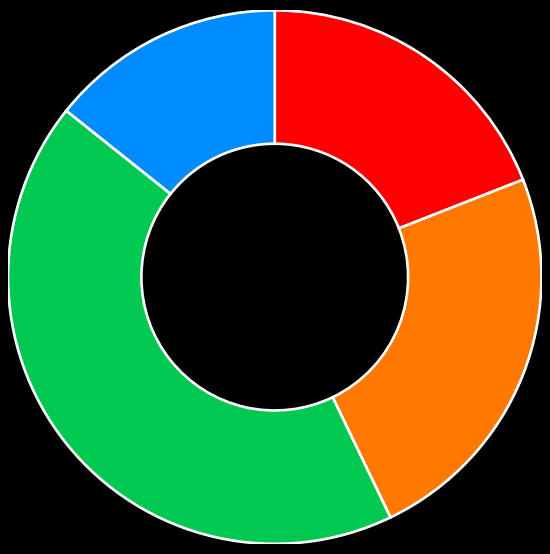
ISSUE SYMBOLIC CODES

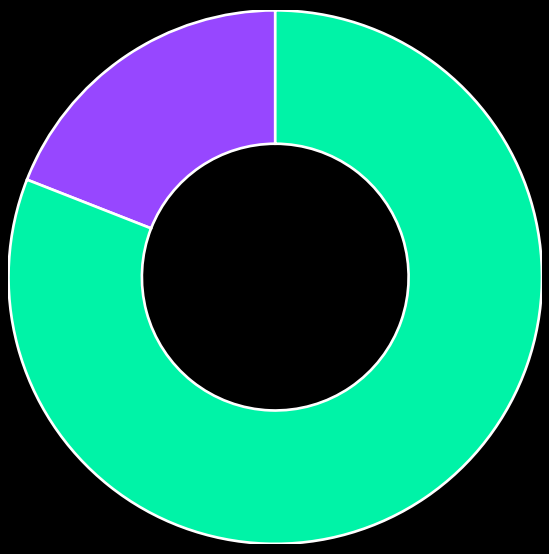
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	4
Medium	5
Low	9
Informational	3

Total: 21





WEAKNESSES

This section contains the list of discovered weaknesses.

FUNG-2

VRF RANDOM NFT REDEMPTION CAN BE MANIPULATED

SEVERITY: High

PATH:

protocol/NftIndex.sol:selectNft#L709-717

REMEDIATION:

To mitigate this issue, it is crucial that the random selection of the NFT is only dependent on the variable randomly generated word from Chainlink VRF and any values that are fixed from the request creation (such as i and callprice in this case) and not on variables that can change between the request creation and fulfilment (such as vaultSize).

A possible solution could be to fix vaultSize in the request as well, similar to callprice. This way there is no manipulation possible unless the calculated ID is greater than the actual vault size during fulfilment, for which an adjustment would have to be made.

STATUS: Fixed

DESCRIPTION:

A user can redeem their NFT LIQ tokens for random NFTs by first calling **redeem**, which initiates a redemption request for a random number through Chainlink VRF.

Eventually, VRF will call **fulfillRandomWords** with a random number that is used to calculate the index for the random NFT. On line 648 this index is calculated:

```
uint256 randomIndex = uint256(keccak256(abi.encodePacked(randomWords[0], i,
callPrice)));
(nft, stickerPrice) = selectNft(randomIndex, vaultSize);
```

It consists of the random word, the enumerator and the NFT price at the point of request creation. Then **selectNft** is called with this index to choose an actual NFT from the **vault**, which is an **EnumerableSet** that holds all of the NFT keys.

In there, the random key is taken by doing a modulo operation against the **vaultSize**. This **vaultSize** is the set's length and it can be manipulated prior to fulfilment of the random number (e.g. by front-running).

By increasing the vault's size, the randomly chosen NFT would shift up and another one would be taken. Since the vault holds all NFTs, it can be arbitrarily manipulated by depositing cheap NFTs.

Utilising this, an attacker could ensure that they receive an NFT that is much higher than their floor price for the price of the floor price, leading to profits at the cost of the users and the protocol.

```
    function selectNft(uint256 randomNum, uint256 vaultSize) internal view
returns(SharedObjs.Nft memory nft, uint256 stickerPrice) {

        uint256 randomIdx = randomNum % vaultSize;
        nft = nfts[EnumerableSet.at(vault, randomIdx)];
        assert(nft.nftContract != address(0));

        stickerPrice = getBaseRedemptionPrice(nft.nftContract);
        assert(stickerPrice != 0);
    }
```

REDEEMING REQUIRES MORE TOKENS THAN WILL EVER BE MINTED

SEVERITY: High

PATH:

contracts/protocol/NftIndex.sol#L575-L577

contracts/protocol/NftIndex.sol#L699

REMEDIATION:

Consider charging the redemption fee in another token, such as ETH. To implement this, the protocol will need an oracle to determine the exchange rate from the NftIndex token to the ETH token.

STATUS: Acknowledged

DESCRIPTION:

The deposit and redeem functionality is incomplete because the **redeem()** function requires more tokens than **deposit()** produces due to the fee and the buffer.

This will always lead to one unredeemable NFT in the pool.

For example, on the first deposit to the pool the contract will mint 10 tokens if the price of the deposited NFT is 100 USD. Some part of these tokens will be sent to the depositor, some will be left on the contract as a fee. Even if the depositor gets all 10 tokens, to redeem the NFT, they would need 10 tokens + redemption buffer (2%) + redemption fee (0.5%), which is impossible because the token's total supply is only 10 tokens.

The same situation would occur if there were other tokens, but they were withdrawn. The only way to withdraw the NFT is to lock another one or

disable the fee and the buffer.

```
    uint256 basePrice = baseRedeemPrice();
    uint256 redeemFee = basePrice * redemptionFeeRate / RATE;
    uint256 callPrice = basePrice + redeemFee;
```

```
      basePrice = basePrice * redemptionBuffer / RATE;
```

FUNG-19

THE INCENTIVE SHOULD NOT BE APPLIED TO UNINCENTIVIZED TOKENS

SEVERITY: High

PATH:

contracts/protocol/NftIndex.sol#L350-L352

contracts/protocol/NftIndex.sol#L450

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

The **NftIndex.nftAssetsNeeded()** function calculates the number of NFTs required from a given contract to reach a target weight. This calculation is based on the following formula:

```
*                      (assetVaultCount + nftsNeeded) * floorPrice
*   targetWeight = ---------------------------------------------------
*                    totalVaultFloorValue + (nftsNeeded * floorPrice)
```

where:

- **totalVaultFloorValue** is the combined value of all vaults listed as STANDARD type.
- **assetVaultCount** is the value of the deposited NFT's vault.

The denominator represents the NEW total value of STANDARD vaults after **nftsNeeded** tokens are deposited. The issue arises if the deposited NFT is not a STANDARD token, but an UNINCENTIVIZED one. In this case, the denominator does not increase post-deposit, breaking the formula's assumption. As a result, this leads to an incorrect calculation of the incentive amount for users depositing an unincentive token.

```
*                      (assetVaultCount + nftsNeeded) * floorPrice
*   targetWeight = ----------------------------------------------------
*                    totalVaultFloorValue + (nftsNeeded * floorPrice)
```

```
            uint256 incentiveRate =
nftsNeededToIncentiveRate(nftAssetsNeeded(address(nftContract)));
```

Consider setting the incentiveRate = 0 in the function NftIndex.deposit() if the deposited token is a UNINCENTIVIZED token.

```
- uint256 incentiveRate =
nftsNeededToIncentiveRate(nftAssetsNeeded(address(nftContract)));
+ uint256 incentiveRate;
+ if (asset.listedType == SharedObjs.ListedType.STANDARD) {
+   incentiveRate =
nftsNeededToIncentiveRate(nftAssetsNeeded(address(nftContract)));
+ }
```

WPUNKS HAVE DYNAMIC TOTALSUPPLY

SEVERITY: High

PATH:

contracts/protocol/NftIndex.sol#L237-L244

contracts/protocol/NftIndex.sol#L323-L376

REMEDIATION:

You can make an exception in the code for WPUNKS and call the original contract instead or generalize and add another parameter to assets that will contain the supply of an original contract and a wrapper.

It is also possible to create another wrapper for WPUNKS, which will call the original contract on totalSupply() but the WPUNKS contract on transfers and other functions.

STATUS: Fixed

DESCRIPTION:

The Cryptopunks ERC721 wrapper contract, WPUNKS, which is planned to be used in Fungify, has a dynamic **totalSupply()** function. The return value depends on the number of punks wrapped through the contract. Any punk holder can change it by minting or burning.

Currently, the total supply of WPUNKS is 798, 12 times smaller than the real size of the collection, 10000.

The **NftIndex** contract uses total supplies to calculate a collection's market capitalization using the **nftContractMarketCap()** function.

The lesser total supply of WPUNKS leads to overincentivization of the collection because the **nftAssetsNeeded()** function will output more NFTs than it should, and the ratio between Cryptopunks and other collections in the vault will be broken.

A higher percentage of one collection in the vault will lead to a higher influence of this collection on the vault's value, which will, thus, increase instability and put under question the main purpose of the index token: diversification of assets.

Another possible impact is the total supply manipulation that will affect the calculated market capitalization. If attackers have several Cryptopunks, they can mint WPUNKS to increase the market capitalization and get more incentives and steal from the incentives pool, or burn to decrease it and frontrun other users to disincentivize them.

```
    function nftContractMarketCap(address nftContract) public view
returns(uint256) {

        // Get latestFloor from chainlink
        uint256 latestFloor = oracle.getAssetPrice(nftContract);

        // Calculate nftContractMarketCap by multiplying floor price by
totalSupply of nfts in that contract
        return latestFloor * IERC721Enumerable(nftContract).totalSupply();
    }
```

```
function nftAssetsNeeded(address nftContract) public view returns (int) {

    /* Gets values needed for the calculation */

    uint256 floorPrice = oracle.getAssetPrice(nftContract);

    if (floorPrice == 0) {
        return 0;
    }

    uint256 targetWeight = nftContractTargetWeight(nftContract);

    if (targetWeight == RATE) {
        return 0;
    }

    // Multiplies that price by the vault count to get the sum.
    uint256 currentValue = floorPrice * indexAssets[nftContract].workingSize;

    // Naive target value, only considering current totalVaultFloorValue
    uint256 totalVaultFloorValue = totalStandardVaultValue();
    uint256 targetValue = totalVaultFloorValue * targetWeight / RATE;

    /**
     * Step 1:
     * ASCII Art for the original equation:
     *
     *                      (assetVaultCount + nftsNeeded) * floorPrice
     *   targetWeight =
-------------------------------------------------------------
     *                   totalVaultFloorValue + (nftsNeeded * floorPrice)
     *
     *
     * Step 2:
     * Solving for nftsNeeded.
     *
     * ASCII Art of intermediate equation:
     *
     *                (floorPrice * assetVaultCount) - (totalVaultFloorValue *
targetWeight)
```

```
     *   nftsNeeded =
-------------------------------------------------------------------------------
-----
     *                              (floorPrice * targetWeight) - floorPrice
     *
     * Step 3:
     * Substitute currentValue = floorPrice * assetVaultCount
     * Substutute targetValue = totalVaultFloorValue * targetWeight
     *
     * ASCII Art for the implemented equation:
     *
     *                       currentValue - targetValue
     *   nftsNeeded = ---------------------------------------
     *                (floorPrice * targetWeight) - floorPrice
     */
    return (int(currentValue) - int(targetValue)) /
        ((int(floorPrice * targetWeight / RATE)) - int(floorPrice));
}
```

SIGNATURE REUSE ACROSS FORKS

SEVERITY: Medium

PATH:

contracts/protocol/NftIndex.sol::permit()#L112-L133

REMEDIATION:

Add the chainID in the schema of the signature passed to the permit() function. Consider following the EIP712 implementation when calculating the domain separator: openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol at 83c7e45092dac350b070c421cd2bf7105616cf1a · OpenZeppelin/openzeppelin-contracts

STATUS: Fixed

DESCRIPTION:

The **NftIndex.sol** contract computes the domain separator using the network's chainID, which is fixed at the time of initialization. In the event of a post-deployment chain fork, the **chainID** cannot be updated, and the signatures may be replayed across both versions of the chain.

```
    function initialize(PriceOracle _oracle, int[] calldata nftsNeeded,
uint[] calldata incentives) public onlyOwner {
        if (_initialized == true) {
            revert AlreadyInitialized();
        }

        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256("EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)"),
                keccak256(bytes(name())),
                keccak256(bytes("1")),
                block.chainid,
                address(this)
            )
        );

  [...]
```

In the context of a post-deployment chain hard fork, where a subset of the community does not implement the upgrade, two parallel chains with the same **chainID** variable can exist. This creates an opportunity for malicious actors to replay valid signatures across both versions of the chain. Aa result, an attacker could reuse signatures on both chains. If a change in the **chainID** is detected, the domain separator can be cached and regenerated.

```
    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external {
        if (deadline < block.timestamp) {
            revert ExpiredPermit();
        }
        if (DOMAIN_SEPARATOR == bytes32(0)) {
            revert UninitializedDomainSeparator();
        }
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01",
DOMAIN_SEPARATOR, keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender,
value, nonces[owner]++, deadline))));
        address recoveredAddress = ecrecover(digest, v, r, s);
        if (recoveredAddress == address(0) || recoveredAddress != owner) {
            revert InvalidPermitSignature();
        }
        _approve(owner, spender, value);
    }
```

INCORRECT COMPARISON BETWEEN CALLBALANCE AND STICKERPRICE CAN PREVENT THE FULFILLMENT OF THE REDEEM REQUEST

SEVERITY: Medium

PATH:

contracts/protocol/NftIndex.sol#L653-L661

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

The function **NftIndex.fulfillRandomWords()** handles the actual redemption process after the random number is retrieved from the VRF. It uses a for loop to redeem an NFT in each iteration. Within this loop, the function checks if the **callBalance** is sufficient by comparing it with the **stickerPrice** before proceeding with the redemption to ensure the user's balance is adequate.

The issue arises when the **redemptionFeeRate** is factored in. This fee rate is charged in each iteration, increasing the total amount the user needs to pay for each NFT redemption to **stickerPrice + stakerFees**.

Due to this fee mechanism, the comparison on line 653 does not account for the **redemptionFeeRate**, leading to a reversion when **stickerPrice <= callBalance < stickerPrice + stakerFees** due to an underflow error on line 658 in the else block.

```
if (stickerPrice > callBalance) {
    redeemed = false;


} else {
    redeemNft(nft, redeemer, recipient, stickerPrice);
    callBalance -= stickerPrice + stakerFees;
    vaultSize--;
    redeemed = true;
}
```

Consider modifying the code from line 653 to line 661 to:

```
- if (stickerPrice > callBalance) {
+ if (stickerPrice + stakerFees > callBalance) {
    redeemed = false;


} else {
    redeemNft(nft, redeemer, recipient, stickerPrice);
    callBalance -= stickerPrice + stakerFees;
    vaultSize--;
    redeemed = true;
}
```

MISSING SLIPPAGE PARAMETER IN THE DEPOSIT() FUNCTION

SEVERITY: Medium

PATH:

contracts/protocol/NftIndex.sol#L410-L484

REMEDIATION:

We recommend adding a minimal token output parameter to the function and reverting if the actual amount transferred to the user is less than the parameter.

STATUS: Fixed

DESCRIPTION:

Market conditions can change after a user sends a **deposit()** transaction. It is necessary to give the user the choice to revert the transaction if the output token amount is less than expected.

Another user can frontrun (intentionally or not) the first user's transaction with another **deposit()** transaction that will decrease the incentive rate for the first user's transaction, leading to an unexpected loss.

```
function deposit(SharedObjs.Nft [] calldata assets) external {
    uint256 totalLiqToMint;
    uint256 totalSellerMintCut;
    uint256 totalStakersCut;


    // Loops through all assets and deposits all respective nftIds
    for(uint256 i; i < assets.length;) {
        IERC721 nftContract = IERC721(assets[i].nftContract);
        uint256 tokenId = assets[i].tokenId;
        bytes32 key = _getNftKey(address(nftContract), tokenId);
        SharedObjs.IndexAsset storage asset =
indexAssets[address(nftContract)];
        if (asset.assetType != SharedObjs.AssetType.ERC721 ||
asset.listedType == SharedObjs.ListedType.REMOVED || asset.listedType ==
SharedObjs.ListedType.UNLISTED) {
            revert InvalidAsset();
        }


        // Gets the Liq value of the deposited Nfts
        uint256 callPrice = oracle.getAssetPrice(address(nftContract));
        uint256 assetLiqPrice = usdToLiq(callPrice);
        asset.totalVaultLiq += assetLiqPrice;


        // Transfers in the nfts and verifies balances
        {
            uint256 balanceBefore = nftContract.balanceOf(address(this));


            assert(!EnumerableSet.contains(vault, key)); // Vault must not
contain this key/asset
            if(EnumerableSet.add(vault, key)) {
                // Note: asset.size is incremented below, alongside LIQ
mint, to avoid inaccurate prices during the batch call
                if(nfts[key].nftContract == address(0)){
                    nfts[key] = SharedObjs.Nft(address(nftContract),
tokenId);
                }
            }
```

```
            nftContract.safeTransferFrom(msg.sender, address(this),
tokenId);
            // Calculate the amount that was *actually* transferred
            if (nftContract.balanceOf(address(this)) - balanceBefore != 1) {
                revert BalanceMismatch();
            }
        }


        // Get nftsNeeded for the specified contract.
        uint256 incentiveRate =
nftsNeededToIncentiveRate(nftAssetsNeeded(address(nftContract)));
        ++asset.workingSize;


        // Calculates the fee distrbution of each nft being transferred
        (uint256 stakersCut, uint256 sellerMintCut, uint256 sellerPoolCut,
uint256 protocolCut) = calculateDepositFeeDistribution(assetLiqPrice,
incentiveRate);


        totalLiqToMint += protocolCut + stakersCut + sellerMintCut;
        totalStakersCut += stakersCut;
        totalSellerMintCut += sellerMintCut + sellerPoolCut;


        // Updates Incentives Pool
        if(protocolCut != 0) {
            incentivesPool += protocolCut;
        }
        if (sellerPoolCut != 0) {
            if (incentivesPool >= sellerPoolCut) {
                incentivesPool -= sellerPoolCut;
            } else {
                incentivesPool = 0;
            }
        }
        emit DepositNFT(msg.sender, msg.sender, key, address(nftContract),
tokenId, callPrice, stakersCut, sellerMintCut, sellerPoolCut, protocolCut);
        unchecked { ++i; }
    }
```

```
    // Handles all variables related to LIQ Price
    for(uint256 i; i < assets.length;) {
        ++indexAssets[assets[i].nftContract].size; // Increments collection
size which affects totalVaultValue and hence LIQ price
        unchecked { ++i; }
    }


    stakerAllocatedFees += totalStakersCut;
    _mint(address(this), totalLiqToMint);
    _transfer(address(this), msg.sender, totalSellerMintCut);
}
```

FUNG-17

THE RATE EMA IN THE FUNGIFYPRICEFEED CONTRACT IS NOT EFFECTIVE

SEVERITY: Medium

PATH:

contracts/protocol/NftIndex.sol#L410-L484

REMEDIATION:

Check if the rate has been updated in the current block or other time period.

We also recommend updating the rates in a decentralized manner with a committee of several independent entities that should agree on the price in the time period.

STATUS: Acknowledged

DESCRIPTION:

The EMA restriction **overMaxPriceDeviation** on the rate updates in **FungifyPriceFeed.updateRate()** function can be bypassed by sending several **updateRate()** calls in a row. In each call, the attacker can pass a new rate that falls into the allowed deviation, which consequently allows the attacker to reach any value of the rate they want to apply through a series of calls. The issue happens because there are no checks if the rate has been updated in the current block.

The **wards** addresses with permission to call the function are partially trusted (the admins gave the permission but additional checks are intended) but centralized entities, and they should be restricted to give the protocol participants time to react in case of a malicious action.

```
function overMaxPriceDeviation(int256 newRate, int256 oldRate) public view
returns(bool) {


    int allowedDeviation = oldRate * maxPriceDeviationBip / ONE_IN_BIP;


    if (newRate > oldRate + allowedDeviation || newRate < oldRate -
allowedDeviation) {
        return true;
    } else {
        return false;
    }
}


function updateRate(int256 newRate_) public auth {


    int256 oldEMA = latestAnswer;


    if (overMaxPriceDeviation(newRate_, oldEMA)) {
        revert OverPriceDeviationMaximum();
    }


    int256 newEMA = updateEMA(oldEMA, newRate_);


    latestAnswer = newEMA;
    latestUpdateTime = block.timestamp;
    emit RateUpdated(description, oldEMA, newEMA, newRate_);
}


function updateEMA(int256 currentEMA, int256 newValue) internal view returns
(int256) {
```

```
    /*
        Multiplier: 2 / (emaPeriods + 1)
        EMA: (LastestValue - PreviousEMA) * Multiplier + PreviousEMA
    */


    int256 emaPeriods_ = emaPeriods;


    int256 newEMA =
        sub(
            add(
                newValue * 2 / (emaPeriods_ + 1),
                currentEMA
            ),
            currentEMA * 2 / (emaPeriods_ + 1)
        );


    return newEMA;
}
```

FUNG-18

ATTACKER CAN DRAIN CONTRACT'S VRF BALANCE USING ERC721 CALLBACK

SEVERITY: Medium

PATH:

contracts/protocol/NftIndex.sol#L765

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

The NFT redemption process consists of two steps, each executed in separate transactions:

1. **Request Creation**: This step involves transferring LIQ tokens from the sender to the contract and sending a request to the Chainlink VRF coordinator to obtain a random word.
2. **Request Fulfillment**: This step completes the actual NFT redemption using the random word provided by the VRF coordinator.

The flow of the second step starts with the **VRFCoordinatorV2.fulfillRandomWords()** function in the chainlink/contracts/src/v0.8/dev/VRFCoordinatorV2.sol at 9ae2b4aafa22d60bde18337d502893f36748605a · smartcontractkit/chainlink.

```
function fulfillRandomWords(Proof memory proof, RequestCommitment memory rc)
external nonReentrant returns (uint96) {
    ...

    bytes memory resp =
abi.encodeWithSelector(v.rawFulfillRandomWords.selector, requestId,
randomWords);

    ...

    uint96 payment = calculatePaymentAmount(
      startGas,
      s_config.gasAfterPaymentCalculation,
      getFeeTier(reqCount),
      tx.gasprice
    );
    if (s_subscriptions[rc.subId].balance < payment) {
      revert InsufficientBalance();
    }
    s_subscriptions[rc.subId].balance -= payment;

    ...
}
```

This function, in turn, calls the **rawFulfillRandomWords()** function of the **NftIndex** contract, with a gas limit defined by **callbackGasLimit**. The equivalent value of the gas consumed in **NftIndex.rawFulfillRandomWords()** is then deducted from the VRF balance of the contract.

An attacker can exploit this by manipulating the **rawFulfillRandomWords()** function to maximize gas consumption up to the **callbackGasLimit**, thereby draining the VRF balance of the contract. This can be done by leveraging the callback of the **ERC721.safeTransferFrom()** invocation in the **NftIndex.redeemNft()** function. The attacker can consume as much gas as possible and then revert the transaction in the callback, consuming up to 63/64 of the gas as per EIP-150.

By repeatedly triggering the redemption process and reverting the transaction in the ERC721 callback, the attacker can deplete the VRF balance

of the **NftIndex** contract, ultimately preventing users from redeeming their NFTs.

To execute the attack, the attacker must lock their LIQ tokens into the NftIndex contract for each call to redeem the NFT. The protocol's admin will refund this amount to the attacker using the **nftIndex.refundRedeemer()** function. However, the refund will only be processed if the protocol fails to detect the attacker's malicious actions.

```
nftContract.safeTransferFrom(msg.sender, address(this), tokenId);
```

Consider using the ERC721.transferFrom() function instead of ERC721.safeTransferFrom() to transfer the NFT to the redeemer on line 765 to avoid the callback.

```
- IERC721(nft.nftContract).safeTransferFrom(address(this), recipient,
nft.tokenId);
+ IERC721(nft.nftContract).transferFrom(address(this), recipient,
nft.tokenId);
```

Although this remediation deviates from OZ's recommendation, from a security standpoint, it's preferable not to initiate the callback from the user to avoid potential attacks. If we proceed with this approach, we should provide a documented warning to users.

Alternatively, we could consider restricting the gas limit for the invocation of safeTransferFrom().

This section will discuss the cost and possibility of the exploit for the attacker.

To drain the VRF balance of the contract, attackers need to execute the redemption process multiple times, which requires them to have a significant amount of LIQ tokens to send within the **NftIndex.redeemNft()** function.

The question is whether the LIQ tokens can be refunded to the attackers. The answer depends on the sophistication of the backend (BE) infrastructure. If the BE infrastructure is advanced enough to detect whether a user is attempting to manipulate the redemption process to drain the VRF balance, the attack would result in a loss for the attackers. However, it is challenging to ensure the infrastructure can consistently detect such attacks. Attackers could split their LIQ tokens to carry out the exploit, making it difficult for the system to determine whether the user is intentionally reverting the transaction to drain funds or if it is an unintended revert.

Bonus: Currently, the **callbackGasLimit** value can be set to 25e5, which translates to approximately $50-$60 on the Ethereum network at present. Therefore, with each call to the redeem function, the attacker can reduce the VRF balance by $50.

FUNG-10

STANDARD VERSIONS OF LIBRARIES ARE USED IN THE UPGRADEABLE CONTRACTS

SEVERITY: Low

REMEDIATION:

Replace the libraries with the upgradeable versions.

STATUS: Fixed

DESCRIPTION:

The OpenZeppelin and Chainlink libraries used in **NftIndex**, **NftIndexProxy**, **Upgradeable**, **Upgradeable2Step**, and **VrfV2PlusConsumer** are not upgradeable. In the case of an upgrade in the system, if the dependencies versions have changed, there can be a shift in the storage slots, leading to a broken storage layout.

The mentioned upgradeable versions of the libraries implement various techniques to prevent this, such as storage gaps, namespaced layout, and initializers instead of constructors.

contracts/proxies/Upgradeable.sol#L4

```
import "@openzeppelin/access/Ownable.sol";
```

contracts/proxies/Upgradeable2Step.sol#L4

```
import "@openzeppelin/access/Ownable2Step.sol";
```

contracts/proxies/NftIndexProxy.sol#L5

```
import  "@openzeppelin/access/AccessControl.sol";
```

contracts/protocol/NftIndex.sol#L4-L8

```
import "@openzeppelin/token/ERC20/ERC20.sol";
import "@openzeppelin/token/ERC20/utils/SafeERC20.sol";
import "@chainlink/vrf/dev/libraries/VRFV2PlusClient.sol";
import "@openzeppelin/utils/structs/EnumerableSet.sol";
import "@openzeppelin/access/AccessControl.sol";
```

contracts/vrfUtils/VrfV2PlusConsumer.sol#L5-L6

```
import "@openzeppelin/access/Ownable.sol";
import "@chainlink/vrf/dev/VRFConsumerBaseV2Plus.sol";
```

THE FEE CALCULATION IN NFTINDEX.REDEEM() IS NOT ACCURATE

SEVERITY: Low

PATH:

contracts/protocol/NftIndex.sol#L677-L700

contracts/protocol/NftIndex.sol#L575-L576

REMEDIATION:

Add the buffer after the fee calculation.

STATUS: Fixed

DESCRIPTION:

The fee calculated in the **NftIndex.redeem()** function will be unnecessary higher by 2% in the default configuration because the **basePrice** calculated in the **baseRedeemPrice()** includes the **redemptionBuffer**.

It will make the redemption less efficient and will increase the leftover funds.

```
    function baseRedeemPrice() public view returns(uint256 basePrice) {

        address nftContract;
        uint256 averageFloor;

        uint256 listedAssetsLength = listedAssets.length;

        for (uint256 i; i < listedAssetsLength;) {
            nftContract = listedAssets[i];

            // Avoids division by zero when collection not in vault
            averageFloor = getBaseRedemptionPrice(nftContract);

            // Tracks maximum
            if (averageFloor > basePrice) {
                basePrice = averageFloor;
            }

            unchecked { ++i; }
        }

        // Applies the buffer, helps prevent failed redemptions during high
traffic
        basePrice = basePrice * redemptionBuffer / RATE;
    }
```

```
        // Fetches the price of a redemption
        uint256 basePrice = baseRedeemPrice();
        uint256 redeemFee = basePrice * redemptionFeeRate / RATE;
```

MISSING LIMIT CHECKS IN FEE AND BUFFER SETTING FUNCTIONS

SEVERITY: Low

PATH:

contracts/protocol/NftIndex.sol::setRedemptionBuffer()#L1003-1005

contracts/protocol/NftIndex.sol::setDepositFeeRate()#L1021-1023

contracts/protocol/NftIndex.sol::setRedemptionFeeRate()#L1030-1032

contracts/protocol/NftIndex.sol::setVeFungSpreadRate()#L1039-1041

REMEDIATION:

Consider adding appropriate limits to these functions to prevent setting excessively high values.

STATUS: Fixed

DESCRIPTION:

While the mentioned functions are restricted to being called only by accounts with the **INTERNAL_ROLE**, there is still a risk that they could be set to unreasonably high values either by mistake or maliciously, due to the absence of limit checks. This could result in excessively high fees that could be detrimental to users.

```
    function setRedemptionBuffer(uint256 newRedemptionBuffer) external
onlyRole(INTERNAL_ROLE) {
        redemptionBuffer = newRedemptionBuffer;
    }
```

```
    function setDepositFeeRate(uint256 newDepositFeeRate) external
onlyRole(INTERNAL_ROLE) {
        depositFeeRate = newDepositFeeRate;
    }
```

```
    function setRedemptionFeeRate(uint256 newRedemptionFeeRate) external
onlyRole(INTERNAL_ROLE) {
        redemptionFeeRate = newRedemptionFeeRate;
    }
```

```
    function setVeFungSpreadRate(uint256 newVeFungSpreadRate) external
onlyRole(INTERNAL_ROLE) {
        veFungSpreadRate = newVeFungSpreadRate;
    }
```

UNNECESSARY RECEIVE FUNCTION IN NFTINDEX

SEVERITY: Low

PATH:

contracts/protocol/NftIndex.sol#L56

REMEDIATION:

Remove the function.

STATUS: Fixed

DESCRIPTION:

The contract doesn't use native currency. The function creates unnecessary exposure to accidental losses of native funds.

```
    receive() external payable {}
```

FUNG-11

NO EVENT IS EMMITTED ON UPGRADEABLE.REPLACEIMPLEMENTATION()

SEVERITY: Low

PATH:

contracts/proxies/Upgradeable.sol#L11-L13

contracts/proxies/Upgradeable2Step.sol#L23-L30

REMEDIATION:

Add the event emission.

STATUS: Fixed

DESCRIPTION:

There's no **ReplaceImplementation** event emission in the **Upgradeable.replaceImplementation()** function in contrast to the **Upgradeable2Step.replaceImplementation()** function.

```
    function replaceImplementation(address impl_) public onlyOwner {
        implementation = impl_;
    }
```

```
    function acceptImplementation() public {
        if (msg.sender != pendingImplementation) {
            revert OwnableUnauthorizedAccount(msg.sender);
        }
        emit ReplaceImplementation(implementation, msg.sender);
        delete pendingImplementation;
        implementation = msg.sender;
    }
```

AN INCORRECT ASSERTION ABOUT TOTALVAULTLIQ IN THE FUNCTION REMOVEASSET()

SEVERITY: Low

PATH:

contracts/protocol/NftIndex.sol#L199-L202

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

In the function **NftIndex.removeAsset()**, when **asset.size != 0** or **asset.amount != 0**, the function incorrectly asserts that **asset.totalVaultLiq** always receives the value **0**. This assertion is mistaken because if **asset.size** is not **0**, it indicates the existence of an NFT in the contract. Therefore, **totalVaultLiq** must be greater than **0**, unless the NFT has no value, which is impossible.

```
if (asset.size != 0 || asset.amount != 0) {
    assert(asset.totalVaultLiq == 0);
    revert NonEmptyAsset();
}
```

Consider modifying lines 199 to 202 of the contract NftIndex to:

```
if (asset.size != 0 || asset.amount != 0) {
-     assert(asset.totalVaultLiq == 0);
+     assert(asset.totalVaultLiq != 0);
      revert NonEmptyAsset();
}
```

FUNG-9

PROXIES DUPLICATE THE IMPLEMENTATION FUNCTIONALITY

SEVERITY: Low

PATH:

contracts/proxies/NftIndexProxy.sol#L7

contracts/proxies/Upgradeable.sol#L6

contracts/proxies/Upgradeable2Step.sol#L10

REMEDIATION:

Use the common Transparent Proxy from the OpenZeppelin library instead.

STATUS: Acknowledged

DESCRIPTION:

The **NftIndexProxy** and **FungifyPriceFeedProxy** contain multiple duplicate functions from the **Ownable**, **Ownable2Step**, and **AccessControl** libraries which are the parent contracts of the implementation contract **NftIndex** and **FungifyPriceFeed**. If these functions are upgraded on the implementation contracts, they cannot be reached because the proxies will always call their own.

```
contract NftIndexProxy is Proxy2Step, AccessControl {
```

```
contract Upgradeable is Ownable {
```

```
contract Upgradeable2Step is Ownable2Step {
```

FUNG-12

NO STORAGE GAPS IN THE INHERITED CONTRACTS

SEVERITY: Low

REMEDIATION:

Add the storage gaps.

Read more here: Writing Upgradeable Contracts - OpenZeppelin Docs

STATUS: Fixed

DESCRIPTION:

The dependencies of **NftIndex**:

- **Upgradeable2Step**
- **VrfV2PlusConsumer**
- **NftIndexState**

And **FungifyPriceFeed**:

- **Upgradeable**

Don't implement storage gaps, complicating possible upgrades of the system.

FUNG-13

PROXIES USE THE STANDARD STORAGE LAYOUT INSTEAD OF THE NAMESPACED

SEVERITY: Low

PATH:

contracts/proxies/Upgradeable.sol#L7

contracts/proxies/Upgradeable2Step.sol#L11-L12

REMEDIATION:

Replace the storage variables accordingly to ERC-7201.

See the OpenZeppelin implementation for an example.

STATUS: Acknowledged

DESCRIPTION:

Using the standard storage layout creates additional exposure to broken storage slots during an upgrade because the slots become reserved on the implementation, and it's easy to use them by accident.

```
    address public implementation;
```

```
    address public pendingImplementation;
    address public implementation;
```

CONSIDER UTILIZING INTERNAL CALLS TO RETRIEVE THE TOTAL SUPPLY AND USER BALANCES

SEVERITY: Informational

PATH:

contracts/protocol/NftIndex.sol#L499

contracts/protocol/NftIndex.sol#L795

contracts/protocol/NftIndex.sol#L818

contracts/protocol/NftIndex.sol#L841

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

The **ERC20** contract provides the **totalSupply()** and **balanceOf()** functions as **public**, allowing them to be invoked internally within the contract. However, in the **NftIndex** contract, which inherits from **ERC20**, these functions are called using external calls, resulting in higher gas consumption.

```
assert(this.balanceOf(address(this)) >= incentivesPool);
```

```
assert(this.balanceOf(address(this)) >= incentivesPool +
stakerAllocatedFees);
```

```
uint256 liqSupply = this.totalSupply();
```

Consider calling totalSupply() and balanceOf() function internally.

```
- this.balanceOf(address(this))
+ balanceOf(address(this))
```

```
- this.totalSupply();
+ totalSupply();
```

UNUSED ERRORS

SEVERITY: Informational

PATH:

ChainlinkPriceOracle.sol#L13

REMEDIATION:

If the error is redundant and there is no missing logic where it can be used, it should be removed.

STATUS: Fixed

DESCRIPTION:

The **ChainlinkPriceOracle** contract declares the **Unauthorized** error, but it is never used.

```
error Unauthorized();
```

FUNG-6

UNUSED WETH VARIABLE IN NFTINDEX

SEVERITY: Informational

PATH:

contracts/protocol/NftIndex.sol#L48

REMEDIATION:

Consider removing it.

STATUS: Fixed

DESCRIPTION:

The **weth** variable is not used anywhere in the contract.

```
    address public immutable weth;
```


hexens
×
Fungify.