SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	openq
Prepared by:	Sherlock
Lead Security Expert:	0x52
Dates Audited:	February 1 - February 15, 2023
Prepared on:	March 7, 2023

Introduction

OpenQ provides a codebase with the world's most intuitive Web3-powered toolkit for discovering, managing, and incentivizing software engineers, anywhere on planet Earth.

Scope

Includes:

- All contracts in `/contracts`, **EXCLUDING** the `Mocks` directory

Excludes:

- Any off-chain services, like our oracles which are all running in Node.js on OpenZeppelinDefenderAutotask.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
9	7

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0x52
ctf_sec
ltyu
Robert
TrungOre
0xdeadbeef
bin2chen
XKET
ck

cccz
yongkiws
rvierdiiev
tsvetanovv
yixxas
GimelSec
clems4ever
KingNFT
libratus
cryptostellar5
Ruhum
unforgiven
HonorLt
HollaDieWaldfee
Jeiwan
joestakey
carrot
seyni
8olidity
0xbepresent
CodeFoxInc
holyhansss
ast3ros
kiki_dev
chaduke
Breeje
ADM
StErMi
usmannk
ak1
__141345__
Avci
caventa
cergyk
dipp
jkoppel
0xmuxyz
Tricko
csanuragjain
hake
eyexploit
ArcAnya
imare
Atarpara
chainNue
MyFDsYours
RaymondFam
Bauer
nicobevi
Aymen0909
oot2k
slowfi
sinh3ck
whiteh4t9527
peanuts

Issue H-1: Adversary can lock every deposit forever by making a deposit with _expiration = type(uint256).max

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/362

Found by

Robert, GimelSec, Ityu, cryptostellar5, bin2chen, yixxas, KingNFT, 0xdeadbeef, clems4ever, TrungOre, 0x52

Summary

DepositMangerV1 allows the caller to specify _expiration which specifies how long the deposit is locked. An adversary can specify a deposit with _expiration = type(uint256).max which will cause an overflow in the BountyCore#getLockedFunds subcall and permanently break refunds.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/DepositManager/Implementations/DepositManagerV1.sol#L54-L56

DepositManagerV1#fundBountyToken allows the depositor to specify an _expiration which is passed directly to BountyCore#receiveFunds.

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L47-L52

BountyCore stores the _expiration in the expiration mapping.

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L339-L349

When requesting a refund, getLockedFunds returns the amount of funds currently locked. The line to focus on is `depositTime[depList[i]] + expiration[depList[i]]`

An adversary can cause `getLockedFunds` to always revert by making a deposit in which `depositTime[depList[i]] + expiration[depList[i]] > type(uint256).max` causing an overflow. To exploit this the user would make a deposit with `_expiration = type(uint256).max` which will cause a guaranteed overflow. This causes DepositMangerV1#refundDeposit to always revert breaking all refunds.

Submitting as high risk because when combined with payout breaking methods it will result in all deposited tokens being stuck forever.

Impact

Adversary can permanently break refunds

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/DepositManager/Implementations/DepositManagerV1.sol#L152-L195

Tool used

Manual Review

Recommendation

Add the following check to DepositMangerV1#fundBountyToken:

```
+   require(_expiration <= type(uint128).max)
```

Discussion

FlacoJones

Overflow check: https://github.com/OpenQDev/OpenQ-Contracts/pull/129

Funder == Issuer: https://github.com/OpenQDev/OpenQ-Contracts/pull/116

Issue H-2: Adversary can brick bounty payouts by calling fundBountyToken but funding it with an ERC721 token instead

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/352

Found by

Robert, Ityu, XKET, ctf_sec, 0x52, ck

Summary

recieveFunds is only meant to be called with an ERC20 token but _receiveERC20 is generic enough to work with an ERC721 token. An adversary could call this with an ERC721 token to add it as a bounty reward. The problem is that the payout functions would completely break when trying to send it as a payout. The result is that afterwards the bounty would be completely broken.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L197-L209

BountyCore#_receiveERC20 makes two calls to the underlying token contract. The first is the transferFrom method which exists functions identical to the ERC20 variant using token id instead of amount.

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L291-L299

The other call that's made is balanceOf which is also present in ERC721

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L221-L228

On the flipside, when withdrawing and ERC20 it uses the transfer method which doesn't exist in ERC721. The result is that ERC721 tokens can be deposited as ERC20 tokens but can't be withdrawn. Since the contract will revert when trying to payout the ER721 token all payouts from the bounty will no longer work.

Submitting as high risk because when combined with refund locking methods it will result in all deposited tokens being stuck forever.

Impact

Bounty will be permanently unclaimable

Code Snippet

Tool used

Manual Review

Recommendation

Split the whitelist into an NFT whitelist and an ERC20 whitelist, to prevent a whitelisted NFT being deposited as an ERC20 token.

Discussion

FlacoJones

Will fix with an explicity OpenQTokenWhitelist, no arbitrary token addresses

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/113 and https://github.com/OpenQDev/OpenQ-Contracts/pull/116 and https://github.com/OpenQDev/OpenQ-Contracts/pull/114 effectively remove the possibility of this exploit

Issue H-3: Tier winner can steal excess funds from tiered percentage bounty if any deposits are expired

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/275

Found by

0x52

Summary

Total funds used to calculate the percentage payment is set when the bounty is closed. If the main deposit is refundable by the time when the bounty is closed, a tier winner could exploit the bounty to steal all the funds in the contract.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredPercentageBountyV1.sol#L123-L136

When a tiered bounty is closed, the balances of each payout token is snapshot.

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredPercentageBountyV1.sol#L104-L120

These balance snapshots are used to calculate the amount to pay of each payout token. The balances are only snapshot once and don't change when deposits are refunded. A tier winner can abuse this structure to steal excess funds from the contract if there are any expired deposits. This would be accomplished by using a very short-lived deposit to artificially inflate the prize pool before claiming and refunding to drain all available funds.

Example: User A creates a competition with 10,000 USDC in prizes. The contest goes longer than expected and their deposit becomes available for refund. User B wins 3rd place which entitles them to 10% of the pool. User A makes a call adding User B as the winner of tier 2 (3rd place). User B calls DepositManagerV1#fundBountyToken to fund the bounty with 90,000 USDC and a _expiration of 1 (second). They then call ClaimManagerV1#permissionedClaimTieredBounty which snapshots the total bounty at 100,000 USDC and entitles User B to 10,000 USDC (10%) of winnings. The next block User B calls DepositManagerV1#refundDeposit and refunds their deposit. Since the initial 10,000 USDC deposit has expired, User B withdraws the other 90,000 USDC in the contract. This leaves the bounty with no funds.

Impact

Tier winner can steal all funds in the percentage bounty during claim if initial deposit is expired

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredPercentageBountyV1.sol#L104-L120

Tool used

Manual Review

Recommendation

All deposits should be locked for some minimum amount of time after a tiered bounty has been closed

Discussion

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/112 and https://github.com/OpenQDev/OpenQ-Contracts/pull/117

Issue H-4: Adversary can permanently break percentage tier bounties by funding certain ERC20 tokens then refunding

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/267

Found by

cccz, rvierdiiev, 0x52, bin2chen, ctf_sec, TrungOre, tsvetanovv

Summary

Some ERC20 tokens don't support 0 value transfers. An adversary can abuse this by adding it to a percentage tier bounty then refunding it. This is because after the refund the token will still be on the list of tokens to distribute but it will have a value saved of 0. This means that no matter what it will always try to transfer 0 token and this will always revert because the specified token doesn't support zero transfers.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredPercentageBountyV1.sol#L123-L136

TieredPercentageBountyV1#closeCompetition set the final fundingTotals for each token. If a token has no balance then the fundingTotals for that token will be zero.

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredPercentageBountyV1.sol#L104-L120

For each token in tokenAddresses it will send the claimedBalance to the claimant. If fundingTotal is 0 then it will attempt to call transfer with an amount of 0. Some ERC20 tokens will revert on transfers like this.

An adversary can purposefully trigger these conditions by making a deposit with ERC20 token that has this problem. This will add the ERC20 token to tokenAddresses and cause the contract to try to send 0 when making a payout. Payouts will become completely bricked, with no way to recover since fundingTotals can't be set anywhere else.

Submitting as high because it can be used in conjunction with methods of breaking refunds to permanently trap user funds.

Impact

Payouts are permanently bricked

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredPercentageBountyV1.sol#L104-L120

Tool used

Manual Review

Recommendation

Add two fixes:

1) If a deposit is refunded and the contract has no tokens left then remove that token from the list of tokens
2) Add a condition to _transferERC20 that only transfers if _volume != 0

Discussion

FlacoJones

Valid. Will fix with an explicity token whitelist and requiring funder == issuer

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/112

and

https://github.com/OpenQDev/OpenQ-Contracts/pull/113

and

https://github.com/OpenQDev/OpenQ-Contracts/pull/116

Issue H-5: Adversary can permanently break reward distribution for percentage tier bounties by funding bounty then refunding after competition closes

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/266

Found by

Robert, cccz, CodeFoxInc, libratus, 8olidity, Ityu, XKET, seyni, HonorLt, yixxas, Ruhum, holyhansss, unforgiven, ctf_sec, ast3ros, TrungOre, 0xbepresent, 0x52

Summary

When closeCompetition is called for TieredPercentageBountyV1 it takes a snapshot of the current token balance. Afterwards it uses this number to calculate the payouts. When a deposit is refunded after the competition is closed then the contract won't have enough funds to pay users.

To exploit this an adversary can make a deposit for a token that has a current balance of zero using a expiration of 1 second. After the competition closes they refund their deposit. Now when the contract tries to give payouts to the winners it will try to payout a token that it no longer has any of, causing it to revert anytime someone tries to claim a payout.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredPercentageBountyV1.sol#L123-L136

TieredPercentageBountyV1#closeCompetition set the final fundingTotals for each token.

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredPercentageBountyV1.sol#L104-L120

For each token in tokenAddresses it will send the claimedBalance to the claimant. If a deposit is refunded after the competition is closed then the contract won't have enough funds to pay users.

An adversary can exploit this by making a deposit of 100 for some token with and _expiration of 1 (second). After the competition has been closed they can refund their deposit causing the contract to be short on funds. If the user makes a deposit with an ERC20 token payouts can be re-enabled by donating to make the contract whole. The user can permanently break payouts by using native MATIC as the deposit. All bounty contracts have their receive function disabled which means the contract can't just be funded, which permanently breaks payouts.

Submitting as high because it can be combined with methods for breaking refunds to lock user funds permanently.

Impact

Adversary can permanently break TieredPercentageBounty payouts

Code Snippet

Tool used

Manual Review

Recommendation

Since TieredPercentageBounty is designed to distribute all deposits present at closing, refunds should be disabled after the competition is closed.

Discussion

FlacoJones

Valid. Will fix by removing this contract for now.

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/112

Issue H-6: Refunds can be bricked by triggering OOG (out of gas) in DepositManager

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/77

Found by

Jeiwan, bin2chen, joestakey, KingNFT, carrot, clems4ever, jkoppel, chainNue, MyFDsYours, rvierdiiev, GimelSec, ak1, HonorLt, yixxas, HollaDieWaldfee, holyhansss, hake, Robert, eyexploit, Ityu, kiki_dev, unforgiven, ctf_sec, imare, TrungOre, 0x52, Atarpara, seyni, 0xdeadbeef

Summary

The `DepositManager` contract is in charge of refunding tokens from the individual bounties. This function ends up running a for loop over an unbounded array. This array can be made to be sufficiently large to exceed the block gas limit and cause out-of-gas errors and stop the processing of any refunds.

Vulnerability Detail

The function `refundDeposit()` in `DepositManager.sol` is responsible for handling refunds, through the following snippet of code, https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/DepositManager/Implementations/DepositManagerV1.sol#L152-L181 We are here interested in the line https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/DepositManager/Implementations/DepositManagerV1.sol#L171-L172 which calculates available funds. If we check the function `getLockedFunds()`, we see it run a for loop https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L333-L352 This loop is running over the list of ALL deposits. The deposit list is unbounded, since there are no checks for such limits in the `receiveFunds()` function. This can result in a very long list, causing out-of-gas errors when making refund calls.

Impact

Inability to withdraw funds. Can be forever locked into the contract.

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L333-L352

Tool used

Manual Review

Recommendation

Put a bound on the function `receiveFunds` to limit the number of deposits allowed.

Discussion

FlacoJones

Confirmed. Will fix by requiring funder == issuer and implementing a simple deposit volume greater than/less than token balance after claims

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/117 and https://github.com/OpenQDev/OpenQ-Contracts/pull/116

Issue H-7: Bounties can be broken by funding them with malicious ERC20 tokens

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/62

Found by

Tricko, Jeiwan, bin2chen, joestakey, KingNFT, carrot, usmannk, clems4ever, jkoppel, rvierdiiev, GimelSec, HonorLt, yixxas, HollaDieWaldfee, hake, slowfi, Robert, cccz, CodeFoxInc, libratus, kiki_dev, sinh3ck, unforgiven, oot2k, ctf_sec, imare, 0xbepresent, 0x52, TrungOre, ck, whiteh4t9527, dipp, XKET, 0xdeadbeef, csanuragjain, tsvetanovv

Summary

Any malicious user can fund a bounty contract with a malicious ERC20 contract and prevent winners from withdrawing their rewards.

Vulnerability Detail

The `DepositManagerV1` contract allows any user to fund any bounty with any token, as long as the following check passes in `fundBountyToken` function https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/DepositManager/Implementations/DepositManagerV1.sol#L45-L50 This check always passes as long as `tokenAddressLimitReacehd` is false, which will be the case for most cases and is a reasonable assumption to make.

The claims are handled by the `claimManagerV1` contract, which relies on every single deposited token to be successfully transferred to the target address, as can be seen with the snippet below, taken from the function `_claimAtomicBounty` https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/ClaimManager/Implementations/ClaimManagerV1.sol#L123-L134 and the `claimBalance` function eventually ends up using the `safeTransfer` function from the Address library to carry out the transfer as seen in `BountyCore.sol:221` https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L221-L228 However, if any of the tokens end up reverting during the `safeTransfer` call, the entire claiming call is reverted. It is quite easy for an attacker to fund the bounty with such a malicious contract, and this would essentially brick the bounty preventing the claims from being processed. This will also lock up the funds in the contract until expiry, which can be an indefinite amount of time.

This attack can be carried out in the following bounty types:

1. AtomicBountyV1
2. TieredFixedBounty

3. TieredPercentageBounty

by using malicious ERC20 tokens following the same attack vector.

Impact

Bricking of bounty and lock up of funds until expiry timestamp

Code Snippet

The attack can be recreated using the following test snippet

```
it('should revert when holding malicious token', async () => {
          // ARRANGE
          const volume = 100
          await openQProxy.mintBounty(
            Constants.bountyId,
            Constants.organization,
            atomicBountyInitOperation
          )
          const bountyAddress = await openQProxy.bountyIdToAddress(
            Constants.bountyId
          )
          await depositManager.fundBountyToken(
            bountyAddress,
            ethers.constants.AddressZero,
            volume,
            1,
            Constants.funderUuid,
            { value: volume }
          )

          // Attacker funds with bad token
          const attacker = claimantThirdPlace
          const MockBAD = await ethers.getContractFactory('MockBAD')
          const mockBAD = await MockBAD.connect(attacker).deploy()
          await mockBAD.deployed()
          await mockBAD.connect(attacker).approve(bountyAddress, 10000000)
          await depositManager
            .connect(attacker)
            .fundBountyToken(
              bountyAddress,
              mockBAD.address,
              volume,
              1,
              Constants.funderUuid
            )
          // Blacklist malicious token
          await mockBAD.connect(attacker).setBlacklist(bountyAddress)

          // ACT
          await expect(
            claimManager
              .connect(oracle)
              .claimBounty(
                bountyAddress,
                claimant.address,
                abiEncodedSingleCloserData
              )
          ).to.be.revertedWith('blacklisted')
        })
```

Where the `mockBAD` token is defined as follows, with a malicious blacklist:

```
contract MockBAD is ERC20 {
    address public admin;
    address public blacklist;

    constructor() ERC20('Mock DAI', 'mDAI') {
        blacklist = address(1);
        _mint(msg.sender, 10000 * 10 ** 18);
        admin = msg.sender;
    }

    function setBlacklist(address _blacklist) external {
        require(msg.sender == admin, 'only admin');
        blacklist = _blacklist;
    }

    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        require(from != blacklist, 'blacklisted');
        super._beforeTokenTransfer(from, to, amount);
    }
}
```

Tool used

Hardhat Manual Review

Recommendation

Use a try-catch block when transferring out tokens with `claimManager.sol`. This will ensure that even if there are malicious tokens, they wont revert the entire transaction.

Discussion

FlacoJones

Will fix by implementing an explicity token whitelist

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/113 and https://github.com/OpenQDev/OpenQ-Contracts/pull/116

Issue M-1: Non-whitelisted tokens cannot be added if the limit of token addresses is filled with whitelisted ones

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/530

Found by

Jeiwan, bin2chen, RaymondFam, carrot, ast3ros, rvierdiiev, yixxas, cergyk, Breeje, HollaDieWaldfee, hake, CodeFoxInc, libratus, kiki_dev, unforgiven, 0xbepresent, XKET, Ruhum, 0xdeadbeef, csanuragjain

Summary

Non-whitelisted tokens cannot be deposited to a bounty contract if too many whitelisted contracts were deposited.

Vulnerability Detail

The DepositManagerV1.fundBountyToken function allows depositing both whitelisted and non-whitelisted tokens by implementing the following check:

1. if a token is whitelisted, it canbedepositedwithoutrestrictions;
2. if a token is not whitelisted, it cannotbedepositedifopenQTokenWhitelist.TOKEN_ADDRESS_LIMITtokenshavealreadybeendeposited.

However, while the token addresses limit requirement is only applied to non-whitelisted tokens, whitelisted tokens also increase the counter of token addresses: both non-whitelisted and whitelisted token addresses are addedtothetokenAddressesset.

Impact

Bounty minters may not be able to deposit non-whitelisted tokens after they have deposited multiple whitelisted ones.

Code Snippet

DepositManagerV1.sol#L45-L50 BountyCore.sol#L326-L328 BountyCore.sol#L55

Tool used

Manual Review

Recommendation

Consider excluding whitelisted token addresses when checking the number of deposited tokens against the limit.

Discussion

FlacoJones

We are going to remove the ability to fund with an arbitrary ERC20 - removing the TOKEN_ADDRESS_LIMIT and simply reverting if a token, erc721 or erc20, is not whitelisted

This will effectively cap the number of ERC20 or ERC721 tokens to the total number of whitelisted tokens (which the protocol controls)

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/113

Issue M-2: Null characters are not checked in string values

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/439

Found by

0xdeadbeef, yongkiws

Summary

OpenQ consumes data from the on-chain contracts using even monitoring with subgraph. Strings with null characters that are emitted as part of an on-chain event will be recognized off-chain as if they were without null character.

This can corrupt data off-chain and possibly lead to impersonation. Some examples of strings that are emitted that include strings: `TokenDepositReceived` - contains issue id and organization name (ORG name ZZ\x00Z will be the same as ZZZ) `BountyClosed` - contains issue id and organization name (ORG name ZZ\x00Z will be the same as ZZZ), closer data.

Vulnerability Detail

Events are emitted without validation that the string does not contain any null characters, Example can be found: https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/ClaimManager/Implementations/ClaimManagerV1.sol#L294

Due to a known limitation on subgraph: https://github.com/graphprotocol/graph-tooling/issues/1098

Events that contain strings with null characters will be transformed to the same string without the null characterss

Impact

The OpenQ platform is heavily dependent on the on-chain events execute the business.

Events are emitted during critical operations such as funding, claiming, refunding, etc..

An attacker can create strings that will corrupt the data and possibly impersonate himself to a user/org

- ORG name ZZ\x00Z will be the same as ZZZ

Code Snippet

In the description

Tool used

Manual Review

Recommendation

Consider adding a validator function that checks every index of the string to see if it is the ASCII range. This will also help restrict to specific encoding.

Discussion

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/128

hrishibhat

Sponsor comment:

> I believe it could allow for user impersonation if they used escape characters to pose as someone else and attempt to claim

Issue M-3: Claimed amount of tokens will be computed incorrectly if rebasing token is used

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/305

Found by

Avci, libratus, GimelSec, ak1, yixxas, Breeje, carrot, __141345__, TrungOre, ADM, tsvetanovv

Summary

Protocol wants to support all kinds of tokens, including rebasing tokens as noted in Sherlock's On-chain context guidelines. If rebasing tokens are used as payoutTokens for TieredPercentageBounty, users may either underclaim or overclaim dependant on whether token rebased up or down. One of the more popular rebasing token is stETH, with a 5 billion market cap currently.

Vulnerability Detail

The first time a claim is made in a bounty, `closeCompetition()` is called to prevent any further funding of the bounty contract. `closeCompetition()` keeps track of the total balances of each token in the `fundingTotals[]` array at this point in time. However, claims are not all made in the same timestamp. A rebasing token can make subsequent claims that are made to be computed wrongly.

```
function closeCompetition() external onlyClaimManager {
    require(
        status == OpenQDefinitions.OPEN,
        Errors.CONTRACT_ALREADY_CLOSED
    );

    status = OpenQDefinitions.CLOSED;
    bountyClosedTime = block.timestamp;

    for (uint256 i = 0; i < getTokenAddresses().length; i++) {
        address _tokenAddress = getTokenAddresses()[i];
        fundingTotals[_tokenAddress] = getTokenBalance(_tokenAddress);
    }
}
```

`claimedBalance()` is calculated in this way. It uses the snapshot of the balance of the token and take a percentage based on `payoutSchedule`. Because rebasing token balances can adjust up or down, but `fundingTotals[]` remain constant, the amount claimed by users will be more than expected if token rebases down, and less than expected if token rebases up.

```
uint256 claimedBalance = (payoutSchedule[_tier] *
            fundingTotals[_tokenAddress]) / 100;
```

Impact

User will either overclaim, which results in some other user not being able to claim their rightful amount, or underclaim tokens themselves.

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredPercentageBountyV1.sol#L116
https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredPercentageBountyV1.sol#L134

Tool used

Manual Review

Recommendation

In order to add support for rebasing tokens, consider checking the new balance right before a claim is made

Discussion

FlacoJones

Will fix with explicit token whitelist

FlacoJones

and removing crowdfunding

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/112

Issue M-4: [Medium] - Funds locked if bounty is funded with both ether and erc20 tokens at the same time

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/288

Found by

Tricko, Jeiwan, joestakey, caventa, nicobevi, carrot, usmannk, clems4ever, rvierdiiev, HonorLt, yixxas, HollaDieWaldfee, eyexploit, 8olidity, ctf_sec, 0xbepresent, Aymen0909, 0x52, Bauer, Ruhum, peanuts, ADM

Summary

When a bounty is created. Next step is to fund such bounty with the rewards by the creator. To do that, a call to `fundBountyToken()` in DepositManagerV1.sol must be done. This function allows both kind of funds, native tokens (eth) sending value to the transaction, or any allowed token through `_tokenAddress` (previously whitelisted by the protocol). Internally, the function `receiveFunds()` is called for that particular bounty.

Vulnerability Detail

If for some reason the bounty creator calls this method sending both ether and a valid `_tokenAddress` then the ether sent will be lost. In the `recieveFunds()` function body we find this condition:

```
uint256 volumeReceived;
if (_tokenAddress == address(0)) {
    volumeReceived = msg.value;
} else {
    volumeReceived = _receiveERC20(_tokenAddress, _funder, _volume);
}
```

In our case, `_tokenAddress` is a valid token so the if condition is not fullfilled and the volume received will be only the token amount. Locking the ether since the deposit will be recorded just for the tokens sent.

There's a function `getLockedFunds()` that looks like a way to get such locked ether. But because these funds are not being recorded as part of a deposit, this function won't work.

Impact

Ether locked on the bounty contract.

Code Snippet

DepositManagerV1.sol#L36-L74

BountyCore.sol#L21-L58

Tool used

Manual Review

Recommendation

The recommendation is to validate that, in case that a `_tokenAddress` is sent, then there is no ether sent too.

```
        uint256 volumeReceived;
        if (_tokenAddress == address(0)) {
            volumeReceived = msg.value;
        } else {
+            if (msg.value != 0) {
+                  revert EtherSent();
+            }
            volumeReceived = _receiveERC20(_tokenAddress, _funder, _volume);
        }
```

Discussion

FlacoJones

Good find! will fix

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/123

Issue M-5: Adversary can break any bounty they wish by depositing an NFT then refunding it

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/263

Found by

Robert, Tricko, cccz, libratus, 8olidity, rvierdiiev, bin2chen, HonorLt, cergyk, Ruhum, HollaDieWaldfee, usmannk, unforgiven, TrungOre, StErMi, 0x52

Summary

Refunding an NFT doesn't remove the nftDeposit so the contract will try to payout an NFT it doesn't have if the NFT has been refunded.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L64-L93

All bounties use BountyCore#refundDeposit to process refunds to user. This simply transfers the NFT back to the funder but leaves the nftDeposit.

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/ClaimManager/Implementations/ClaimManagerV1.sol#L150-L165

When any bounty is claimed it loops through nftDeposits and attempts to transfer an NFT for each one. The problem is that if an NFT has been refunded the deposit receipt will still exist but the contract won't have the NFT. The result is that all payouts will be permanently broken.

Submitting as high risk because when combined with refund locking methods it will result in all deposited tokens being stuck forever.

Impact

Adversary can permanently break payouts on any bounty

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/ClaimManager/Implementations/ClaimManagerV1.sol#L31-L67

Tool used

Manual Review

Recommendation

Remove the deposit receipt from nftDeposits when refunding NFT deposits

Discussion

FlacoJones

Valid. Will fix by removing nfts for now

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/113

and

https://github.com/OpenQDev/OpenQ-Contracts/pull/114

Issue M-6: Adversary can break NFT distribution by depositing up to max then refunding all of them

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/262

Found by

libratus, GimelSec, Jeiwan, dipp, bin2chen, caventa, HonorLt, kiki_dev, Ruhum, HollaDieWaldfee, unforgiven, ctf_sec, clems4ever, jkoppel, 0x52, 0xmuxyz

Summary

Bounties limit the number of NFT deposits to five. An adversary can block adding NFTs by repeatedly depositing and withdrawing an NFT.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L64-L93

All bounties use BountyCore#refundDeposit to process refunds to user. This simply transfers the NFT back to the funder but leaves the nftDeposit. This uses up the deposit limit which is current set to 5. Since the deposit cap is used up by deposits that have been refunded the slots can't be used to distribute legitimate NFTs to the bounty claimant.

Impact

Adversary can block legitimate NFT distribution

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/BountyCore.sol#L64-L93

Tool used

Manual Review

Recommendation

When an NFT deposit is refunded it should remove the depositID from nftDeposits

Discussion

FlacoJones

Valid. Will fix by removing nft funding and disbaling crowdfunding

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/113

and

https://github.com/OpenQDev/OpenQ-Contracts/pull/116

and

https://github.com/OpenQDev/OpenQ-Contracts/pull/114

Issue M-7: OngoingBountyV1 is incompatible with NFTs but still accepts NFT deposits

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/261

Found by

cccz, libratus, dipp, GimelSec, Jeiwan, seyni, joestakey, kiki_dev, cergyk, Ruhum, carrot, HollaDieWaldfee, csanuragjain, ctf_sec, StErMi, 0x52, 0xmuxyz

Summary

OngoingBountyV1 is incompatible with NFTs but still accepts NFT deposits

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/OngoingBountyV1.sol#L133-L160

OngoingBountyV1 is designed to receive NFTs and NFTs can be deposited to it via DepositManager#fundBountyNFT

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/ClaimManager/Implementations/ClaimManagerV1.sol#L173-L197

However when ongoing bounties are claimed they have no method to distribute the NFTs that are deposited.

Impact

OngoingBountyV1 has no way to distribute NFTs

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/ClaimManager/Implementations/ClaimManagerV1.sol#L173-L197

Tool used

Manual Review

Recommendation

Change _claimOngoingBounty to allow it to distribute NFTs

Discussion

FlacoJones

Will remove Ongoing for now

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/112

Issue M-8: Refunding logic with multiple deposits is first mover take all

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/257

Found by

libratus, Jeiwan, Ityu, joestakey, yixxas, chaduke, Ruhum, HollaDieWaldfee, ctf_sec, 0x52

Summary

There is no accounting to keep track of which deposits are being used to pay bounties. This means that after all deposits have expired, the first user to refund their deposit will get more funds.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/DepositManager/Implementations/DepositManagerV1.sol#L171-L179

Available funds are simply calculated by taking the current balance of the bounty contract and subtracting the locked balance. This works well for single deposits but doesn't when there are multiple deposits from multiple funders. When this occurs it will cause a first mover take all situation.

Example: User A and User B both makes 10,000 USDC deposits to an Ongoing bounty, bringing the total funds of the bounty to 20,000 USDC. After some time the bounty is closed. By the time it's closed, it has paid out 10,000 USDC in awards leaving it with 10,000 USDC. User A and B both wait until their deposits are expired. Now User A requests a refund from his deposit. This refunds all 10,000 USDC to User A leaving none for User B. As a result, User A has effectively forced User B to pay all the awards.

Impact

Reward logic is first mover take all

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/DepositManager/Implementations/DepositManagerV1.sol#L152-L195

Tool used

Manual Review

Recommendation

Bounties should track how much has been paid and how much has been funded. When an ongoing bounty is refunded it should even split remaining funds over depositors.

Discussion

FlacoJones

Will fix by requiring funder == issuer

FlacoJones

https://github.com/OpenQDev/OpenQ-Contracts/pull/117

https://github.com/OpenQDev/OpenQ-Contracts/pull/116

Issue M-9: Resizing the payout schedule with less items might revert

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/244

Found by

rvierdiiev, Jeiwan, GimelSec, XKET, bin2chen, ak1, HonorLt, caventa, usmannk, unforgiven, clems4ever, TrungOre, ArcAnya, StErMi, 0x52, ck

Summary

According to some comments in `setPayoutScheduleFixed`, reducing the number of items in the schedule is a supported use case. However in that case, the function will revert because we are iterating over as many items as there was in the previous version of the three arrays making the function revert since the new arrays have less items.

Vulnerability Detail

Let say they were 4 items in the arrays `tierWinners`, `invoiceComplete` and `supportingDocumentsComplete` and we are resizing the schedule to 3 items. Then the following function would revert because we use the length of the previous arrays instead of the new ones in the for loops.

```
function setPayoutScheduleFixed(
        uint256[] calldata _payoutSchedule,
        address _payoutTokenAddress
    ) external onlyOpenQ {
        require(
            bountyType == OpenQDefinitions.TIERED_FIXED,
            Errors.NOT_A_FIXED_TIERED_BOUNTY
        );
        payoutSchedule = _payoutSchedule;
        payoutTokenAddress = _payoutTokenAddress;

        // Resize metadata arrays and copy current members to new array
        // NOTE: If resizing to fewer tiers than previously, the final indexes
↪   will be removed
        string[] memory newTierWinners = new string[](payoutSchedule.length);
        bool[] memory newInvoiceComplete = new bool[](payoutSchedule.length);
        bool[] memory newSupportingDocumentsCompleted = new bool[](
            payoutSchedule.length
        );
        for (uint256 i = 0; i < tierWinners.length; i++) {
↪   <=====================================================
            newTierWinners[i] = tierWinners[i];
        }
        tierWinners = newTierWinners;

        for (uint256 i = 0; i < invoiceComplete.length; i++) {
↪   <=====================================================
            newInvoiceComplete[i] = invoiceComplete[i];
        }
        invoiceComplete = newInvoiceComplete;

        for (uint256 i = 0; i < supportingDocumentsComplete.length; i++) {
↪   <=====================================================
            newSupportingDocumentsCompleted[i] = supportingDocumentsComplete[i];
        }
        supportingDocumentsComplete = newSupportingDocumentsCompleted;
    }
```

The same issue exists on TieredPercentageBounty too.

Impact

Unable to resize the payout schedule to less items than the previous state.

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredFixedBountyV1.sol#L157

Tool used

Manual Review

Recommendation

```
for (uint256 i = 0; i < newTierWinners.length; i++) {
    newTierWinners[i] = tierWinners[i];
}
tierWinners = newTierWinners;

for (uint256 i = 0; i < newInvoiceComplete.length; i++) {
    newInvoiceComplete[i] = invoiceComplete[i];
}
invoiceComplete = newInvoiceComplete;
```

```
for (uint256 i = 0; i < newSupportingDocumentsCompleted.length; i++) {
    newSupportingDocumentsCompleted[i] = supportingDocumentsComplete[i];
}
supportingDocumentsComplete = newSupportingDocumentsCompleted;
```

Note this won't work if increasing the number of items compared to previous state must also be supported. In that case you must use the length of the smallest of the two arrays in each for loop.

Discussion

FlacoJones

A valid issue but an invalid approach. Because if one is INCREASING the size of the array, you will still get an array out of bounds exception.

This suggestion, combined with adding a `i >= previousArray.length` will do the trick

https://github.com/OpenQDev/OpenQ-Contracts/pull/126