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 22, 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 Open Zeppelin Defender Autotask.

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
6	8

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0x52
unforgiven
Robert

bin2chen
ctf_sec
TrungOre

jkoppel
Ityu
XKET

ck
ast3ros
cccz
rvierdiiev
GimelSec
clems4ever
KingNFT
tsvetanovv
0xdeadbeef
HonorLt
libratus
Jeiwan
Ruhum
HollaDieWaldfee
0xbepresent
yixxas
8olidity
usmannk
CodeFoxInc
seyni
chainNue
holyhansss
StErMi
cergyk
Tricko
joestakey
caventa
MyFDsYours
carrot
hake
dipp
chaduke
0xmuxyz
imare
ArcAnya
kiki_dev
csanuragjain
Atarpara
eyexploit
RaymondFam
Breeje
oot2k
slowfi
sinh3ck
whiteh4t9527
ak1

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

0xdeadbeef, Robert, GimelSec, bin2chen, TrungOre, KingNFT, clems4ever, 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

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

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

pauliax

Escalate for 52 USDC. I think this issue is essentially the same as #62. It does not matter if the token is ERC20, ERC721, ERC1155 or ERC777, or whatever, what matters is that this impostor contract pretends to comply with the funding function only later to reveal its dark side. Based on my interpretation, all the issues that are talking about feeding the bounty with 'wrong' tokens should be grouped together under one issue, because all of them are giving basically the same effect.

sherlock-admin

> Escalate for 52 USDC. I think this issue is essentially the same as #62. It does not matter if the token is ERC20, ERC721, ERC1155 or ERC777, or whatever, what matters is that this impostor contract pretends to comply with the funding function only later to reveal its dark side. Based on my interpretation, all the issues that are talking about feeding the bounty with 'wrong' tokens should be grouped together under one issue, because all of them are giving basically the same effect.

You've created a valid escalation for 52 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

Evert0x

Escalation rejected. OpenQ used the same method of locking down everything to a whitelist as a means to fix all of them but they are not all the same and would have needed different fixes for each one if OpenQ had wanted to keep the funding process open.

sherlock-admin

> Escalation rejected. OpenQ used the same method of locking down everything to a whitelist as a means to fix all of them but they are not all the same and would have needed different fixes for each one if OpenQ had wanted to keep the funding process open.

This issue's escalations have been rejected!

Watsons who escalated this issue will have their escalation amount deducted from their next payout.

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

jkoppel, Robert, bin2chen, unforgiven, ast3ros, 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

sherlock-admin

> N/A

You've deleted an escalation for this issue.

ctf-sec

Escalate for 50 USDC. I would love to make a few arguments: this exploits path is feasible because first of all, user can claim the refund after the competition is closed and the snapshot of the tierPercentageContract balance is taken. secondly, the expiration time can be gamed. then the exploit path is formulated:

> 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.

However, the issue "user can claim the refund after the competition is closed" is already rewarded, then either 266 should be considered as a duplicate of this issue or this issue can be considered as a duplicate of the 266

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

the issue " This would be accomplished by using a very short-lived deposit to artificially inflate the prize pool before claiming" because the expiration can be gamed, adversary can set short expiration time.

which is the issue: https://github.com/sherlock-audit/2023-02-openq-judging/issues/229, but is it not rewarded.

my issue https://github.com/sherlock-audit/2023-02-openq-judging/issues/187 also mentioned that the expiration time can be gamed as well and I did articulate the impact:

> Developer B failed to claim the entitled bounty because User A set expiration time too short and claim the refund.

I think https://github.com/sherlock-audit/2023-02-openq-judging/issues/229 can be considered as the duplicate of this issue as well because the short expiration is crucial to this attack. this leads to my next point.

I think this bug report should be a medium instead of high, according to the judging guide given the risk of the attacker is taking.

https://docs.sherlock.xyz/audits/watsons/judging

> Criteria for Issues: Medium: There is a viable scenario (even if unlikely) that could cause the protocol to enter a state where a material amount of funds can be lost. The attack path is possible with assumptions that either mimic on-chain conditions or reflect conditions that have a reasonable chance of becoming true in the future. The more expensive the attack is for an attacker, the less likely it will be included as a Medium (holding all other factors constant). The vulnerability must be something that is not considered an acceptable risk by a reasonable protocol team.
>
> High: This vulnerability would result in a material loss of funds and the cost of the attack is low (relative to the amount of funds lost). The attack path is possible with reasonable assumptions that mimic on-chain conditions. The vulnerability must be something that is not considered an acceptable risk by a reasonable protocol team.

I argue that the attack cost is not low at all, and the attack is risky and the attack can lose money.

Just using the POC provided in this bug report:

> 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.

Even though the expiration time is set to 1 second, before the malicious user B call refundDeposit to claim the 90000 USDC, it is very possible that user C, another tier winner (for example, user C is the tier 1 winner and is eligible for winning 50% of the reward), user C call permissionedClaimTieredBounty to claim 50% of the 100,000 (10000 USDC provided by User A and the rest provided by User B), user C unexpectedly getting 5,0000 but the malicious user B is losing a lot of money.

The lower tier the malicious winner, the more difficult the attack is and the attack is not profitable at all if the attacker lose a lot of money nd athe other tier winner claims the tiered rewards before the attacker refund the token, thus I think the cost of the attack is high and the issue is a medium issue instead of a HIGH issue.

To summarize my point:

1. the root cause: the deposit expiration time is gamed and should be marked as a duplicate of this issue.
2. the issue should be a medium issue given the cost of the attack and the potential heavy loss of the attacker.

sherlock-admin

> Escalate for 50 USDC. I would love to make a few arguments: this exploits path is feasible because first of all, user can claim the refund after the competition is closed and the snapshot of the tierPercentageContract balance is taken. secondly, the expiration time can be gamed. then the exploit path is formulated:
>
> > 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.
>
> However, the issue "user can claim the refund after the competition is

closed" is already rewarded, then either 266 should be considered as a duplicate of this issue or this issue can be considered as a duplicate of the 266

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

the issue " This would be accomplished by using a very short-lived deposit to artificially inflate the prize pool before claiming" because the expiration can be gamed, adversary can set short expiration time.

which is the issue: https://github.com/sherlock-audit/2023-02-openq-judging/issues/229, but is it not rewarded.

my issue https://github.com/sherlock-audit/2023-02-openq-judging/issues/187 also mentioned that the expiration time can be gamed as well and I did articulate the impact:

> Developer B failed to claim the entitled bounty because User A set expiration time too short and claim the refund.

I think https://github.com/sherlock-audit/2023-02-openq-judging/issues/229 can be considered as the duplicate of this issue as well because the short expiration is crucial to this attack. this leads to my next point.

I think this bug report should be a medium instead of high, according to the judging guide given the risk of the attacker is taking.

https://docs.sherlock.xyz/audits/watsons/judging

> Criteria for Issues: Medium: There is a viable scenario (even if unlikely) that could cause the protocol to enter a state where a material amount of funds can be lost. The attack path is possible with assumptions that either mimic on-chain conditions or reflect conditions that have a reasonable chance of becoming true in the future. The more expensive the attack is for an attacker, the less likely it will be included as a Medium (holding all other factors constant). The vulnerability must be something that is not considered an acceptable risk by a reasonable protocol team.
>
> High: This vulnerability would result in a material loss of funds and the cost of the attack is low (relative to the amount of funds lost). The attack path is possible with reasonable assumptions that mimic on-chain conditions. The vulnerability must be something that is not considered an acceptable risk by a reasonable protocol team.

I argue that the attack cost is not low at all, and the attack is risky and the attack can lose money.

Just using the POC provided in this bug report:

>> 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.
>
> Even though the expiration time is set to 1 second, before the malicious user B call refundDeposit to claim the 90000 USDC, it is very possible that user C, another tier winner (for example, user C is the tier 1 winner and is eligible for winning 50% of the reward), user C call permissionedClaimTieredBounty to claim 50% of the 100,000 (10000 USDC provided by User A and the rest provided by User B), user C unexpectedly getting 5,0000 but the malicious user B is losing a lot of money.
>
> The lower tier the malicious winner, the more difficult the attack is and the attack is not profitable at all if the attacker lose a lot of money nd athe other tier winner claims the tiered rewards before the attacker refund the token, thus I think the cost of the attack is high and the issue is a medium issue instead of a HIGH issue.
>
> To summarize my point:
>
> 1. the root cause: the deposit expiration time is gamed and should be marked as a duplicate of this issue.
> 2. the issue should be a medium issue given the cost of the attack and the potential heavy loss of the attacker.

You've created a valid escalation for 50 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

hrishibhat

Escalation rejected

Issue #275 and #266 are different as 275 is about a malicious user inflating the prize pool to steal extra funds using refund deposit and short expiration time. While 266 is user breaking the payouts by first depositing, setting fundingTotals & using refund.

Both issues #229 and #187 do not talk of the above attacks. and mentions only the part of the short expiration time that the Sponsor has addressed in the respective issue.

sherlock-admin

> Escalation rejected
>
> Issue #275 and #266 are different as 275 is about a malicious user inflating the prize pool to steal extra funds using refund deposit and short expiration time. While 266 is user breaking the payouts by first depositing, setting fundingTotals & using refund.
>
> Both issues #229 and #187 do not talk of the above attacks. and mentions only the part of the short expiration time that the Sponsor has addressed in the respective issue.

This issue's escalations have been rejected!

Watsons who escalated this issue will have their escalation amount deducted from their next payout.

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

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

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

kiseln

Escalate for 27 USDC

> Some ERC20 tokens don't support 0 value transfers

There are no relevant tokens that revert on 0 value transfers. `LEND` is often provided as an example, however, it was discontinued in 2020 and is supposed to be migrated to AAVE https://docs.aave.com/faq/migration-and-staking. I'd say there is 0 chance this token is whitelisted as a valid bounty token by the protocol owners.

I would consider `LEND` in the same category as any invalid/malicious token that can be added in a permissionless way, in which case this group of issues is a duplicate of #62

sherlock-admin

> Escalate for 27 USDC
>
>> Some ERC20 tokens don't support 0 value transfers
>
> There are no relevant tokens that revert on 0 value transfers. `LEND` is often provided as an example, however, it was discontinued in 2020 and is supposed to be migrated to AAVE https://docs.aave.com/faq/migration-and-staking. I'd say there is 0 chance this token is whitelisted as a valid bounty token by the protocol owners.
>
> I would consider `LEND` in the same category as any invalid/malicious token that can be added in a permissionless way, in which case this group of issues is a duplicate of #62

You've created a valid escalation for 27 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

Evert0x

Escalation rejected.

Protocol signaled they were planning to use `any` token in this protocol, reverting on 0 transfer is uncommon but not unlikely to happen on a legit token.

sherlock-admin

> Escalation rejected.
>
> Protocol signaled they were planning to use `any` token in this protocol, reverting on 0 transfer is uncommon but not unlikely to happen on a legit token.

This issue's escalations have been rejected!

Watsons who escalated this issue will have their escalation amount deducted from their next payout.

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

CodeFoxInc, yixxas, HonorLt, cccz, 0x52, 0xbepresent, ctf_sec, Robert, unforgiven, libratus, jkoppel, TrungOre, Ityu, seyni, chainNue, holyhanss, Ruhum, XKET, 8olidity

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: 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

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

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

kiseln

Escalate for 20 USDC

This completely bricks payouts just like #62. Shouldn't it be high?

sherlock-admin

> Escalate for 20 USDC
>
> This completely bricks payouts just like #62. Shouldn't it be high?

You've created a valid escalation for 20 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

hrishibhat

Escalation accepted

This issue is a valid high

sherlock-admin

> Escalation accepted
>
> This issue is a valid high

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

sherlock-admin

> Escalation accepted
>
> This issue is a valid high

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

Issue H-7: 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

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

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-8: 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, GimelSec, yixxas, CodeFoxInc, HonorLt, joestakey, KingNFT, dipp, Jeiwan, tsvetanovv, 0x52, cccz, 0xbepresent, ctf_sec, hake, Robert, csanuragjain, libratus, slowfi, HollaDieWaldfee, jkoppel, whiteh4t9527, bin2chen, TrungOre, rvierdiiev, imare, carrot, clems4ever, 0xdeadbeef, usmannk, oot2k, sinh3ck, XKET, kiki_dev

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

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

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 can be deposited without restrictions;
2. if a token is not whitelisted, it cannot be deposited if openQTokenWhitelist.TOKEN_ADDRESS_LIMIT tokens have already been deposited.

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 added to the tokenAddresses set.

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: when issuer set new winner by calling setTierWinner() code should reset invoice and supporting documents for that tier

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/297

Found by

unforgiven

Summary

if invoice or supporting documents are required to receive the winning prize then tier winner should provide them. bounty issuer or oracle would set invoice and supporting document status of a tier by calling `setInvoiceComplete()` and `setSupportingDocumentsComplete()`. bounty issuer can set tier winners by calling `setTierWinner()` but code won't reset the status of the invoice and supporting documents when tier winner changes. a malicious winner can bypass invoice and supporting document check by this issue.

Vulnerability Detail

if bounty issuer set invoice and supporting documents as required for the bounty winners in the tiered bounty, then tier winner should provide those and bounty issuer or off-chain oracle would set the status of the invoice and documents for that tier. but if issuer wants to change a tier winner and calls `setTierWinner()` code would changes the tier winner but won't reset the status of the invoice and supporting documents for the new winner. This is the `setTierWinner()` code in OpenQV1 and TieredBountyCore:

```
function setTierWinner(
    string calldata _bountyId,
    uint256 _tier,
    string calldata _winner
) external {
    IBounty bounty = getBounty(_bountyId);
    require(msg.sender == bounty.issuer(), Errors.CALLER_NOT_ISSUER);
    bounty.setTierWinner(_winner, _tier);

    emit TierWinnerSelected(
        address(bounty),
        bounty.getTierWinners(),
        new bytes(0),
        VERSION_1
    );
}

function setTierWinner(string memory _winner, uint256 _tier)
    external
    onlyOpenQ
{
    tierWinners[_tier] = _winner;
}
```

As you can see code only sets the `tierWinner[tier]` and won't reset `invoiceComplete[tier]` or `supportingDocumentsComplete[tier]` to false. This would cause an issue when issuer wants to change the tier winner. these are the steps that makes the issue:

1. UserA creates tiered Bounty1 and set invoice and supporting documents as required for winners to claim their funds.
2. UserA would set User1 as winner of tier 1 and User1 completed the invoice and oracle would set `invoiceComplete[1] = true`.
3. UserA would change tier winner to User2 because User1 didn't complete supporting documents phase. now User2 is winner of tier 1 and `invoiceComplete[1]` is true and User2 only required to complete supporting documents and User2 would receive the win prize without completing the invoice phase.

Impact

malicious winner can bypass invoice and supporting document check when they are required if he is replace as another person to be winner of a tier.

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredBountyCore.sol#L59-L64

Tool used

Manual Review

Recommendation

set status of the `invoiceComplete[tier]` or `supportingDocumentsComplete[tier]` to false in `setTierWinner()` function.

Discussion

0xunforgiven

Escalate for 31 USDC

my issue is not duplicate of #425.

issue #425 is: "array lists `invoiceCompleteClaimIds[]` and `supportingDocumentsCompleteClaimIds[]` can contain claimIds that are incomplete". that is about those array list containing extra values and the issue has no serious impact because code checks mappings to check invoice and docuements status.

my issue explains that when a tier winner gets changed (Bounty issuer change tier1 winner from User1 to User2 for any reason) code doesn't reset the values in the mappings `invoiceComplete[tier1]` or `supportingDocumentsComplete[tier1]`. so values in those mappings would shoe wrong values for the new tier1 winner. the POC shows how the issue happens:

1. UserA creates tiered Bounty1 and set invoice and supporting documents as required for winners to claim their funds.
2. UserA would set User1 as winner of tier 1 and User1 completed the invoice and oracle would set invoiceComplete[1] = true.
3. UserA would change tier winner to User2 because User1 didn't complete supporting documents phase. now User2 is winner of tier 1 and invoiceComplete[1] is true and User2 only required to complete supporting documents and User2 would receive the win prize without completing the invoice phase.

sherlock-admin

> Escalate for 31 USDC
>
> my issue is not duplicate of #425.
>
> issue #425 is: "array lists `invoiceCompleteClaimIds[]` and `supportingDocumentsCompleteClaimIds[]` can contain claimIds that are incomplete". that is about those array list containing extra values and the issue has no serious impact because code checks mappings to check invoice and docuements status.
>
> my issue explains that when a tier winner gets changed (Bounty issuer change tier1 winner from User1 to User2 for any reason) code doesn't reset the values in the mappings `invoiceComplete[tier1]` or `supportingDocumentsComplete[tier1]`. so values in those mappings would shoe wrong values for the new tier1 winner. the POC shows how the issue happens:
>
> 1. UserA creates tiered Bounty1 and set invoice and supporting documents as required for winners to claim their

funds.

2. UserA would set User1 as winner of tier 1 and User1 completed the invoice and oracle would set invoiceComplete[1] = true.
3. UserA would change tier winner to User2 because User1 didn't complete supporting documents phase. now User2 is winner of tier 1 and invoiceComplete[1] is true and User2 only required to complete supporting documents and User2 would receive the win prize without completing the invoice phase.

You've created a valid escalation for 31 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

hrishibhat

Escalation accepted

Not a valid duplicate of #425 and a valid issue `_eligibleToClaimAtomicBounty` would end up validating a user without completing the invoice phase as shown.

sherlock-admin

> Escalation accepted
>
> Not a valid duplicate of #425 and a valid issue `_eligibleToClaimAtomicBounty` would end up validating a user without completing the invoice phase as shown.

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

Issue M-3: Adversary can block NFT distribution on tiered bounties by assigning the NFTs to unused tiers

Source: https://github.com/sherlock-audit/2023-02-openq-judging/issues/264

Found by

0x52

Summary

Bounties limit the number of NFT deposits to five. An adversary can block adding NFTs by assigning NFTs to tiers that don't exist.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/DepositManager/Implementations/DepositManagerV1.sol#L113-L131

DepositMangerV1#fundBountyNFT passes the user supplied _data to TieredBountyCore#receiveNft

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredBountyCore.sol#L41-L42

_data is decoded and stored in _tier allowing the user to specify any tier they wish.

An adversary can abuse this fill all eligible nft deposit slots with nfts that can't be claimed by any tier. This allows them to effectively disable nft prizes for any tiered bounty. Using a large enough tier will make it impossible to ever claim the nfts because it would cost too much gas to set a large enough payout schedule

Impact

Adversary can effectively disable nft prizes for any tiered bounty

Code Snippet

https://github.com/sherlock-audit/2023-02-openq/blob/main/contracts/Bounty/Implementations/TieredBountyCore.sol#L18-L48

Tool used

Manual Review

Recommendation

TieredBountyCore#receiveNFT should check that specified tier is a within bound (i.e. by comparing it to the length of tierWinners)

Discussion

IAm0x52

Not a dupe of #261 and a separate issue. This discusses supplying NFTs to the incorrect tiers of tiered bounties

hrishibhat

This issue was incorrectly duped with #261 and was missed during the processing of the initial results. This issue is a valid medium, and also unique. As mentioned above, the attacker can fill tiers large enough so that it is not possible to set the payout schedule for any tier due to out of gas.

Issue M-4: 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

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

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-5: Refunding logic with multiple deposits is first mover take all

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

Found by

ctf_sec, yixxas, HonorLt, joestakey, unforgiven, TrungOre, chaduke, MyFDsYours, Ruhum, libratus, Ityu, Jeiwan, 0x52, HollaDieWaldfee

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-6: Resizing the payout schedule with less items might revert

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

Found by

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

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