SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	**Public Best Efforts**
Prepared for:	**Winnables Raffle**
Prepared by:	**Sherlock**
Lead Security Expert:	**0x73696d616f**
Dates Audited:	**August 16 - August 20, 2024**
Prepared on:	**October 2, 2024**

Introduction

Winnables is a cutting-edge decentralized raffle platform (transparent and fair) that offers exciting prizes, including NFTs and crypto, all on the Ethereum network.

Scope

Repository: Winnables/public-contracts

Branch: main

Audited Commit: 9474451539b7081f5b2e246c68b90a16e7c55b31

Final Commit: be8ad9ada69cf341d619f9022c94e372194f179d

For the detailed scope, see the contest details.

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
5	3

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

neko_nyaa
Oblivionis
S3v3ru5
0x73696d616f
kuprum
TessKimy
dimulski
0x0bserver
jennifer37
0xbrivan
0xShahilHussain
Offensive021
p0wd3r
aslanbek
Silvermist
anonymousjoe
almurhasan
0x0x0xw3
jsmi
rbserver
Paradox
iamnmt
durov
0rpse
shaflow01
philmnds
charles__cheerful
Oxsadeeq
tjonair
pashap9990
IvanFitro
dimah7
utsav
araj
shikhar
CatchEmAll
sakshamguruji
PTolev
0xAadi
ogKapten
PNS
rsam_eth
dinkras_
Penaldo
dany.armstrong90
PeterSR
Galturok
matejdb
irresponsible
gajiknownnothing
Waydou
dimi6oni
vinica_boy
y4y
dobrevaleri
shui
AllTooWell
phoenixv110
Trooper
DrasticWatermelon
joshuajee
0xarno
denzi_
KaligoAudits
SadBase
dy
tofunmi
turvec
KungFuPanda
IMAFVCKINSTARRRRRR
Feder
SovaSlava
0xrex
akiro
Trident-Audits
aman
ZC002
eeshenggoh
BitcoinEason
Bluedragon
frndz0ne
Afriaudit
ironside
oxwhite
MSK
oxelmiguel
BlocSoc_Audits
ni8mare
GenevaRoc
gkrastenov
pandasec
0x6a70
AuditorPraise
0xjarix
gerd
ihtishamsudo
4gontuk
Drynooo
UAARRR
sharonphiliplima
MrPotatoMagic
PratRed
korok
roguereggiant
ydlee
petro1912
mahmud
chaduke
ke1caM
KingNFT
dwaipayan01
nilay27
MaanVader
Saurabh_Singh
azanux
thisvishalsingh
boringslav
casper
0xnolo
unnamed

Issue H-1: Users will lock raffle prizes on the `WinnablesPrizeManager` contract by calling `WinnablesTicketManager::propagateRaffleWinner` with wrong CCIP inputs

Source: https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/50

Found by

0rpse, 0x0bserver, 0x73696d616f, 0xAadi, 0xbrivan, 0xrex, CatchEmAll, DrasticWatermelon, Feder, Galturok, IMAFVCKINSTARRRRRR, KungFuPanda, Oblivionis, Offensive021, Oxsadeeq, PNS, PTolev, Paradox, Penaldo, PeterSR, S3v3ru5, SadBase, SovaSlava, Trooper, Waydou, akiro, araj, dany.armstrong90, dimulski, dinkras_, durov, dy, gajiknownnothing, iamnmt, irresponsible, jennifer37, joshuajee, matejdb, neko_nyaa, ogKapten, philmnds, rsam_eth, sakshamguruji, shaflow01, shikhar, tofunmi, turvec, utsav

Summary

The WinnablesTicketManager::propagateRaffleWinner function is vulnerable to misuse, where incorrect CCIP inputs can lead to assets being permanently locked in the `WinnablesPrizeManager` contract. The function does not have input validation for the `address prizeManager` and `uint64 chainSelector` parameters. If called with incorrect values, it will fail to send the message to `WinnablesPrizeManager`, resulting in the assets not being unlocked.

Root Cause

The root cause of the issue lies in the design of the `propagateRaffleWinner` function:

1. The function is responsible for sending a message to WinnablesPrizeManager to unlock the raffle assets.
2. The function is marked as external, so anyone can call it.
3. The function receives `address prizeManager` and `uint64 chainSelector` as inputs, which are responsible for sending the message to the `WinnablesPrizeManager` contract for it to unlock the assets previously locked for the raffle.
4. The inputs forementioned are not validated, meaning users can call the function with wrong values.

5. This cannot be undone, as the function changes the state of the raffle in a way that prevents the function from being called again.

Internal pre-conditions

A raffle must have been won by a player.

External pre-conditions

A user must call `WinnablesTicketManager::propagateRaffleWinner` with incorrect input values.

Attack Path

1. A user wins the raffle.
2. Some user calls `WinnablesTicketManager::propagateRaffleWinner` and provides incorrect inputs for prizeManager and chainSelector.
3. The propagateRaffleWinner function fails to send the correct message to WinnablesPrizeManager due to the parameter mismatch.
4. As a result, the assets associated with the raffle remain locked and cannot be retrieved by the raffle winner.

Impact

This vulnerability completely disrupts the protocol, as it becomes impossible to retrieve the reward of the raffle.

PoC

The test below, which is an edited version of this existing test, shows that the function call will be successful with a random chainSelector

```
    it('Should be able to propagate when the winner is drawn', async () => {
@>    const { events } = await (await
↪   manager.propagateRaffleWinner(counterpartContractAddress, 9846, 1)).wait();
      expect(events).to.have.lengthOf(3);
      const ccipEvent = ccipRouter.interface.parseLog(events[0]);
      expect(ccipEvent.args.receiver).to.eq('0x' +
↪   counterpartContractAddress.toLowerCase().slice(-40).padStart(64, '0'));
      expect(ccipEvent.args.data).to.have.lengthOf(108);
      const drawnWinner = ethers.utils.getAddress('0x' +
↪   ccipEvent.args.data.slice(-40));
      expect(buyers.find(b => b.address === drawnWinner)).to.not.be.undefined;
```

```
      expect(ccipEvent.args.data.slice(0, 68)).to.eq('0x0100000000000000000000000
↪ 0000000000000000000000000000000000000000000001');
    });
```

Mitigation

Implement input validation to ensure that `prizeManager` and `chainSelector` are correct before proceeding with the propagation.

Discussion

matejdrazic

It does not make any sense to group lack of access control on `cancelRaffle` issues with this issue. It should be grouped with #57 .

Brivan-26

@matejdrazic I believe it does make sense. The same root cause (lack of the **same** inputs validation) and the same impact

matejdrazic

@Brivan-26 hey - but its 2 different functions. By following that logic you can group all same type issues into one. So if there were more access control issues on the contract they would be grouped here? That is not right.

Also they do not have the same root? They have the same type of root cause and thats lack of access control.

Brivan-26

@matejdrazic

> So if there were more access control issues on the contract they would be grouped here?

Actually, yes, there were many contests before that had the same access control across multiple contracts even and they are grouped into one report because submitting multiple reports about the same break across different functions is kind of redundant.

Concerning this issue:

- The root cause is a lack of input validation for `prizeManager` and `chainSelector`, not access control as anyone can call this function
- the impact is the message will not reach the destination chain and loss of funds will occur

Concerning `cancelRaffle`:

- The root cause is the same as the previous one, lack of input validation for `prizeManager` and `chainSelector`
- The impact is the same, the message will not reach the destination chain and loss of funds will occur

It is still kind of subjective, it can be a separate report(and I'm okay with that) but it makes sense to group two issues into one single report given the above facts.

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/Winnables/public-contracts/pull/22

Issue H-2: Attacker will prevent any raffles by calling `WinnablesTicketManager::cancelRaffle` before admin starts raffle

Source: https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/57

Found by

0rpse, 0x0bserver, 0x73696d616f, 0xAadi, 0xShahilHussain, 0xarno, 0xbrivan, AllTooWell, BitcoinEason, Bluedragon, CatchEmAll, KaligoAudits, Oblivionis, Offensive021, PNS, PTolev, Paradox, S3v3ru5, Silvermist, TessKimy, Trident-Audits, ZC002, aman, araj, charles__cheerful, denzi_, dimi6oni, dimulski, dinkras_, dobrevaleri, durov, eeshenggoh, frndz0ne, iamnmt, jennifer37, neko_nyaa, ogKapten, p0wd3r, philmnds, phoenixv110, rsam_eth, sakshamguruji, shaflow01, shikhar, shui, tjonair, utsav, vinica_boy, y4y

Summary

The WinnablesTicketManager::cancelRaffle function is vulnerable to abuse because it is an external function that allows anyone to cancel a raffle if its status is set to PRIZE_LOCKED. An attacker could exploit this by repeatedly calling `cancelRaffle` whenever a new raffle is available to be started, effectively preventing any raffles from ever being initiated.

Root Cause

The root cause of this issue lies in the design of the function:

1. The function is external, meaning it can be called by anyone.
2. When called, it checks the underlying function `WinnablesTicketManager::_checkShouldCancel`, which allows cancellation of a raffle if the status is PRIZE_LOCKED, which is a temporary state before the admin calls `WinnablesTicketManager::createRaffle`.
3. This opens up a window of opportunity for an attacker to cancel the raffle before it transitions to an active state.

Internal pre-conditions

There must be a raffleId with raffleStatus == PRIZE_LOCKED

External pre-conditions

The attacker must monitor the contract to identify when a raffle is in the PRIZE_LOCKED state, which occurs after the admin locks a prize in the `WinnablesPrizeManager` contract. The attacker must call the `WinnablesTicketManager::cancelRaffle` before the admin calls `WinnablesTicketManager::createRaffle`.

Attack Path

1. An attacker monitors the contract to detect when a new raffle enters the PRIZE_LOCKED state.
2. As soon as the raffle reaches this state, the attacker calls the cancelRaffle function.
3. The raffle is canceled before it can transition to an active state, preventing it from starting.
4. The attacker can repeat this process for each new raffle, effectively blocking the initiation of all raffles.

Impact

This vulnerability allows a malicious actor to disrupt the entire raffle system. By preventing any raffles from starting, the attacker can undermine the functionality of the whole protocol.

PoC

The test below, which can be added to the hardhat test suite, shows that a random user can cancel the raffle if it hasn't yet been started

```
describe('Buyer can cancel raffle', () => {
  before(async () => {
    snapshot = await helpers.takeSnapshot();
  });

  after(async () => {
    await snapshot.restore();
  });
  const buyers = [];

  it('Should be able to cancel a raffle', async () => {
    const now = await blockTime();
    const buyer = await getWalletWithEthers();
```

```
    await (await link.mint(manager.address,
↪   ethers.utils.parseEther('100'))).wait();
    const tx = await
↪   manager.connect(buyer).cancelRaffle(counterpartContractAddress, 1, 1);
    const { events } = await tx.wait();
    expect(events).to.have.lengthOf(3);
    const ccipMessageEvent = ccipRouter.interface.parseLog(events[0]);
    expect(ccipMessageEvent.name).to.eq('MockCCIPMessageEvent');
    expect(ccipMessageEvent.args.data).to.eq('0x0000000000000000000000000000000000
↪   0000000000000000000000000000000001');
    await expect(manager.getWinner(1)).to.be.revertedWithCustomError(manager,
↪   'RaffleNotFulfilled');
  });
});
```

Mitigation

This vulnerability can be mitigated by updating the underlying function `WinnablesTicketManager::_checkShouldCancel` to only allow the admin to cancel a raffle that hasn't started yet.

Discussion

neko-nyaa

Escalate. Per Sherlock rules, this is a grief-DoS for one block. Because the only impact is denial of a raffle without any actual loss, as well as this function is not time-sensitive, this is a Low.

sherlock-admin3

> Escalate. Per Sherlock rules, this is a grief-DoS for one block. Because the only impact is denial of a raffle without any actual loss, as well as this function is not time-sensitive, this is a Low.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

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

Brivan-26

@neko-nyaa You are incorrect.

Malicious users can repeatedly cancel raffles, preventing the creation of new raffles, stopping users from purchasing tickets, and forcing administrators to constantly create new raffles. This behavior renders the protocol unusable.

matejdrazic

I think my issue #158 is an dup of this issue and not #50 because the root cause is anyone can call cancel function. Can the judge please check this out?

Basicly the root is lack of access control on cancelRaffle. We're just showing 2 different exploits.

0xsimao

There is loss, chainlink tokens to send the cross chain messages, hence high is appropriate.

0rpse

The issue that exists is not only DoS for one block, because of the cross chain messaging the protocol will incur further losses on LINK tokens, though this report and many others do not mention that, some of them do(#391 and others).

The reports that fail to identify at least a Medium impact and only suggest DoS for a block should be invalidated as per Sherlock docs.

MarioBozhikov

Considering this is valid high there is a similar issue #498 that would also render the protocol unusable.

As the intent would be absolutely the same where instead of canceling the raffles all of the players will be refunded and will always need to reenter the raffle.

ArnieSec

In my opinion this should be invalid, from sherlock docs:

> Griefing for gas (frontrunning a transaction to fail, even if can be done perpetually) is considered a DoS of a single block

In this example the LINK token is essentially the gas token that pays for a crosschain transaction, the same way eth is gas for a transaction on mainnet, hence the repeated canceling of the raffle even if done perpetually does not qualify the issue to be valid as can be confirmed from the snippet of the docs above.

The loss is constrained to essentially the gas token of ccip, and is a dos of a single block. This should not be valid according to sherlock rules.

furthermore, also from sherlock docs

> DoS has two separate scores on which it can become an issue: The issue causes locking of funds for users for more than a week. The issue impacts the availability of time-sensitive functions (cutoff functions are not considered time-sensitive).
>
> If both apply, the issue can be considered of High severity.

The issue does not cause locking of user funds for more than a week, and the issue does not impact time-sensitive function

Therefore the issue is clearly invalid, to expand even further, from the readme

> Admins will always keep LINK in the contracts (to pay for CCIP Measages) and in the VRF subscription (to pay for the random numbers)

Brivan-26

@ArnieSec You did not understand the issue. The issue is NOT with the LINK amount not enough to cover cross-chain messages. Check my reply to the waston here

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/Winnables/public-contracts/pull/3

ArnieSec

@Brivan-26

> Malicious users can repeatedly cancel raffles, preventing the creation of new raffles, stopping users from purchasing tickets, and forcing administrators to constantly create new raffles. This behavior renders the protocol unusable.

Your comment is above

> frontrunning a transaction to fail, even if can be done perpetually) is considered a DoS of a single block

From Sherlock docs.

You are describing DOS of single block, the issue is invalid. Not to mention the DOS does not lock user funds at all, a requirement needed for this to even be a high as shown by my earlier comment.

Brivan-26

@ArnieSec

> You are describing DOS of single block

Single block? Anyone can prevent raffle creation at any time no matter which block, this makes the protocol unusable. Read the issue carefully before elaborating further

ArnieSec

Yes that is the definition of perpetually, even if you can dos a block every time, this still does not qualify as valid under Sherlock rules

mystery0x

This finding concerns a scenario where Public Cancel function can DOS the protocol whereas #50 deals with CCIP Parameters in `cancelRaffle` and `propagateWinner` being never validated. They are different from each other and separately valid highs.

WangSecurity

As I understand, it's indeed a DOS of one-block and to have a larger impact, it has to be called repeatedly. But, as was quoted correctly above, the following rule applies:

> Griefing for gas (frontrunning a transaction to fail, even if can be done perpetually) is considered a DoS of a single block, hence only if the function is clearly time-sensitive, it can be a Medium severity issue

But, as I see there's a LINK token loss and as I understand, it's not similar to gas fees, since you still have to pay gas for calling `createRaffle`, and paying LINK is a fee for sending CCIP cross-chain messages. Hence, I agree it's a loss of funds. Of course, feel free to correct me.

@mystery0x could you identify which duplicates of this report identify the loss of LINK token? For example, this report doesn't identify it, and only reports a one-block DOS of non-time-sensitive function.

ArnieSec

@WangSecurity

Maybe I agree there is a loss of funds but the issue cannot be high and should be downgraded to medium, duplicates who do not mention the Loss of link token funds should be de duplicated and made invalid aswell.

> DoS has two separate scores on which it can become an issue: The issue causes locking of funds for users for more than a week. The issue impacts the availability of time-sensitive functions (cutoff functions are not considered time-sensitive).
>
> If both apply, the issue can be considered of High severity.

Due to the fact that this bug only fits 1 of the 2 criteria, the bug cannot qualify for high severity.

Brivan-26

@WangSecurity I still don't understand how this this a DOS of one-block and why the LINK loss is the only impact. The issue described here is that anyone can prevent **any** raffle from being started by calling `cacnelRaffle` which leads to the following impacts:

- No raffle can be started
- Admins should keep creating new raffles and keep locking new prizes
- Loss of LINK

This makes the protocol unusable.

AtanasDimulski

Hey @WangSecurity, the loss of the LINK token comes in second place, with this attack possible the only thing that the protocol can achieve is take space on the network, nothing more.

WangSecurity

> duplicates who do not mention the Loss of link token funds should be de duplicated and made invalid aswell

Yes, I agree with that and that's why I asked the lead judge if the current duplicates are sufficient.

> Due to the fact that this bug only fits 1 of the 2 criteria, the bug cannot qualify for high severity

The reason why I view it as a high-severity finding is not satisfying the DOS requirements but due to LINK token loss. As I understand, this issue has a definite loss of funds without extensive limitations.

> I still don't understand how this this a DOS of one-block

Let me re-iterate. Let's consider only one instance of the attack, the attacker calls `cancelRaffle` front-running the admin's call to start a raffle. But, the admin can start a raffle in the very next block, no? I understand it should be a raffle with another id, but they still can use the same funds to start a new raffle? In this case, it's the one-block DOS. Therefore, for this issue to have a larger impact, the attack has to repeated indefinitely. Therefore, the following from the DOS rules applies:

> Griefing for gas (frontrunning a transaction to fail, even if can be done perpetually) is considered a DoS of a single block, hence only if the function is clearly time-sensitive, it can be a Medium severity issue.

> the loss of the LINK token comes in second place, with this attack possible the only thing that the protocol can achieve is take space on the network, nothing more

Not sure what you mean by that @AtanasDimulski could you please re-iterate? Do you mean the LINK loss is not important and doesn't pose such an impact?

Note: thank you to the Watsons who elaborated on duplicates, I will reply to them once the above question is resolved

Brivan-26

@WangSecurity

> Let me re-iterate. Let's consider only one instance of the attack, the attacker calls cancelRaffle front-running the admin's call to start a raffle. But, the admin can start a raffle in the very next block, no?

In the next block, the same attack can re-occur. Attacker can front-run any start raffle to cancel it. As a consequence, no raffle can be started

> Therefore, for this issue to have a larger impact, the attack has to repeated indefinitely

That's what we are describing here ser. Please check #51 if you think this submission does not outline that clearly

AtanasDimulski

Hey @WangSecurity, first of all, I want to state that I am astonished by the fact we are still discussing whether this issue is valid or not. The whole purpose of the protocol is to create raffles, clearly, it can't. The fact that losing some LINK and ETH fees is considered to be more severe than that is, to put it mildly - illogical. What I mean by the protocol will only consume space is that a piece of code that is unusable, pointless, meaningless, useless, and so on, will be deployed and just exist on the chain. A whole protocol won't work because an extremely cheap attack can be performed multiple times - the cancelation of the transaction is performed on contracts that will be deployed on Avalance. The contract that sends the CCIP message will be deployed on Ethereum. This whole shenanigans is based on one stupid sentence from the Sherloc rules. However, the fact that this is extremely important admin function is disregarded. If we use a bit of logic we will clearly see that this is much more severe than some fees that are lost(fees are just an addition to the severity). Finally when we are going to abandon all sense and logic and simply argue about wording - from the readme: *The protocol working as expected relies on having an admin creating raffles.* - this clearly is not the case. From Sherlock docs: *The protocol team can use the README (and only the README) to define language that indicates the codebase's restrictions and/or expected functionality. Issues that break these statements, irrespective of whether the impact is low/unknown, will be assigned Medium severity. High severity will be applied only if the issue falls into the High severity category in the judging guidelines.* Lets please use some common sense, and stop with this pointless discussion. Thanks for your time.

ArnieSec

@WangSecurity

> The reason why I view it as a high-severity finding is not satisfying the DOS requirements but due to LINK token loss. As I understand, this issue

> has a definite loss of funds without extensive limitations.

The network fee in ccip is .45$ + the destination gas fee, this will be cents since the destination is L2. So in total the entire link lost per tx will be 50 cents. The attacker pays the gas fee associated with calling the function on mainnet which will certainly be alot higher than 50 cents. Given that simply sending Usdc is costing 70 cents right now.

Now given that the impact is dos of one block

And knowing the attack will cost the attacker more than the loss to the protocol.

I think medium severity is appropriate.

AtanasDimulski

@ArnieSec you would have better spent your time writing this report instead of trying to come up with reasons why it is invalid. I suggest you go back to the readme, notice that the WinnablesTicketManager is deployed on Avalance, and then make your calculations again.

AtanasDimulski

The admin pays the CCIP fees on ehtereum + the gas fees for executing the transaction on Ethereum. The attacker pays for the fees to cancel the transaction on Avalance, and the LINK that was deposited by the admin in the WinnablesTicketManager is spent as well. It is best to have common sense!

ArnieSec

The attacker gains nothing, loses his gas cost which will be higher than usual since he is front running meaning he has to pay a higher fee in order to have his tx included first.

Furthermore the raffle creator can just frontrun the attacker and create the raffle.

The issue does not meet the criterium for a high

> The issue causes locking of funds for users for more than a week. The issue impacts the availability of time-sensitive functions (cutoff functions are not considered time-sensitive). If at least one of these are describing the case, the issue can be a Medium.

It simply is not a high imo, there is no need to get emotional...

AtanasDimulski

Thanks for the anger management session, should I start quoting your comments so you don't delete them all the time?

ArnieSec

The discussion has moved from its intended purpose, which was to find the correct severity of the issue. You haven't defended your issue using Sherlock rules as I have defended my position. I think it's time for me to move on and let the judge make his decision

AtanasDimulski

> The discussion has moved from its intended purpose, which was to find the correct severity of the issue. You haven't defended your issue using Sherlock rules as I have defended my position. I think it's time for me to move on and let the judge make his decision

@ArnieSec this discussion shouldn't have occurred in the first place. In my above comments, I have presented multiple arguments both from Sherlock rules and using common sense. The fact that you refuse to, or can't comprehend the severity of the issue is an entirely different topic. I am not gonna spend time calculating the exact attack costs, it is pretty clear who is going to pay more fees - the admin(LINK on Ethereum, ETH on Ethereum, LINK on Avalance). The attacker only pays for gas on Avalance. Again fees are not the main problem here. You have repeatedly quoted two sentences from the Sherloc rules, and refused to apply any logic. In your previous two comments which you have now deleted, you demonstrated a complete misunderstanding of how the protocol is supposed to work, thus my snappy remark.

WangSecurity

> In the next block, the same attack can re-occur. Attacker can front-run any start raffle to cancel it. As a consequence, no raffle can be started That's what we are describing here ser. Please check https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/51 if you think this submission does not outline that clearly

As exactly said in the rule, it's considered a DOS of one-block (even though it can be repeated perpetually). As also said in the rules, it's invalid, unless the function is time-sensitive. Here, the function is not time-sensitive.

> from the readme: The protocol working as expected relies on having an admin creating raffles. - this clearly is not the case.

The statement is not broken, the protocol has an admin that can create a raffle in the next block, this line from the README is not broken here.

About the LINK loss due to sending the CCIP message, as we can see in this page from Chainlink docs the billing mechanism is the blockchain fee + network fee. Blockchain fee is not considered a loss of funds. The Network fee is a couple of dollars. Therefore, it doesn't exceed the "small and finite amounts" for Medium severity:

> Causes a loss of funds but requires certain external conditions or

> specific states, or a loss is highly constrained. The losses must exceed small, finite amount of funds, and any amount relevant based on the precision or significance of the loss.

Excuse me for the confusion caused above. Planning to accept the escalation and invalidate the issue.

Brivan-26

@WangSecurity I really do not understand why constantly applying the rule *DOS of one-block*. The issue is NOT just about blocking an action in one block. This issue will PREVENT ANY RAFFLE FROM BEING STARTED. What's the point of the protocol then?

WangSecurity

@Brivan-26 could you explain how you understand the following rule:

> Griefing for gas (frontrunning a transaction to fail, even if can be done perpetually) is considered a DoS of a single block, hence only if the function is clearly time-sensitive, it can be a Medium severity issue.

This way I would understand what's missing and needs to be explained, cause as I understand the explanation in this comment wasn't good enough?

Brivan-26

> @Brivan-26 could you explain how you understand the following rule:
>
> > Griefing for gas (frontrunning a transaction to fail, even if can be done perpetually) is considered a DoS of a single block, hence only if the function is clearly time-sensitive, it can be a Medium severity issue.
>
> This way I would understand what's missing and needs to be explained, cause as I understand the explanation in this comment wasn't good enough?

I do understand the above rule, ***but it should not be used blindly everywhere***. If a DoS can happen for a **critical** function that makes the protocol completely unusable, we don't invalidate the issue just because there is a general rule like that.

For the last time, the purpose of this protocol is to **create raffles and reward a random winner**. An attacker can front-run **ANY** raffle start to cancel it and prevent the raffle from being started. This means no raffle can be started, which makes the protocol completely unusable and useless, isn't this considered as DoS?

That rule should not be used blindly here. Consider the impact of this issue for a second. This will be my last comment for this issue

philmnds

This is indeed an attack that renders the protocol useless.

And one important thing to add is that it is a simple attack, not a front running one.

The flow of the raffle creation is:

1. Raffle owner calls WinnablesPrizeManager::lockNFT() (or lockETH() or lockTokens())
2. Raffle owner waits for the CCIP message to go from WinnablesPrizeManager to WinnablesTicketManager
3. Raffle owner calls WinnablesTicketManager::createRaffle()

The attacker does not need to wait for the protocol owner to call createRaffle(). If the protocol owner does not immediately call createRaffle() upon the receipt of the CCIP message, the attacker just needs to call WinnablesTicketManager::cancelRaffle(). No mempool reading, no higher gas fees, just a simple txn.

So this is a very simple and hassle free way of making the protocol useless. Any of the white hackers that audited this protocol would be proficient enough to get a simple script running that does this - consider someone with a motivation to do so, such as a another raffle protocol that see Winnables as a competitor.

The dissent towards the invalidation completely loses sight of the security aspect in question, which is the fact that this issue breaks README assertion that the protocol should be able to start a raffle.

It seems bizarre to me that this discussion is taking so long. Not only because this is so clearly a valid issue, but also because it has absolutely no impact for anyone whether it will be valid or invalid - considering the number of dups, this will pay a penny.

ArnieSec

The rule does not state that the attack needs to be frontrunning attack in order to be a dos of 1 block. It states that frontrunning a tx to fail is considered a dos of one block. A dos of one block, with frontrunning or not, is still not grounds for a valid issue according to Sherlock rules.

Just because there is technically no tx front ran, does not change the fact that the issue is still a dos of one block. Because a raffle can therefore be started the next block.

And even if the raffle is then dosed for that block, it is still a dos of one block.

> It seems bizarre to me that this discussion is taking so long. Not only because this is so clearly a valid issue, but also because it has absolutely no impact for anyone whether it will be valid or invalid - considering the number of dups, this will pay a penny.

The impact is that people who judged the issue according to Sherlock rules will be penalized. Additionally users who did not submit the bug because they knew about the ruling are also penalized. While users who submit an issue that does not qualify to be valid according to Sherlock rules are rewarded.

> The dissent towards the invalidation completely loses sight of the security aspect in question, which is the fact that this issue breaks README assertion that the protocol should be able to start a raffle.

From the read me

> We will report issues where the core protocol functionality is inaccessible for at least 7 days. Would you like to override this value?
>
> This value seems reasonable

0xsimao

@WangSecurity this is not a 1 block DoS due to creating a raffle being a cross chain message. The admin would have to wait at least a few blocks to try again because creating a raffle requires a cross chain message.

Also, although the fee cost is small each time, the attack can be carried out infinite times, exceeding small and finite funds (I believe there is also a rule for this).

kuprumxyz

> Also, although the fee cost is small each time, the attack can be carried out infinite times, exceeding small and finite funds (I believe there is also a rule for this).

It is, indeed, V. How to identify a medium issue:

> Breaks core contract functionality, rendering the contract useless or leading to loss of funds of the affected party larger than 0.01% and 10 USD.
>
> > Note: If a single attack can cause a 0.01% loss but can be replayed indefinitely, it will be considered a 100% loss and can be medium or high, depending on the constraints.

DemoreXTess

@WangSecurity

I want to present a different perspective on this issue. We accept issue #26 as valid. While #26 differs slightly, it is quite similar to this issue. In both cases, a raffle that should not be cancellable can be cancelled by an attacker. The only distinction between the two is the timing: in #26, the attacker needs to wait for an edge case, whereas in this issue, the attacker must wait for the CCIP message.

I believe this issue should not be categorized as a DoS because it directly violates a fundamental invariant: a raffle that is meant to be non-cancellable can be cancelled without any conditions.

Given this, I think high severity is appropriate for this issue as it compromises the entire system. If we cannot classify this issue as high severity, I doubt we could classify any other issue as high severity in this contest.

WangSecurity

Watsons, please refrain from off-topic statements and claims that don't add value to the discussion (i.e. who has which interests, how large is the pay for this finding, etc.) and keep your arguments objective.

> I do understand the above rule, but it should not be used blindly everywhere. If a DoS can happen for a critical function that makes the protocol completely unusable, we don't invalidate the issue just because there is a general rule like that.

The rule is not "blindly applied everywhere", this issue (regarding the DOS impact) perfectly fits the rule which was created for these exact issues where one iteration of the attack has little to no impact, i.e. the call can be repeated in the next block successfully.

> For the last time, the purpose of this protocol is to create raffles and reward a random winner. An attacker can front-run ANY raffle start to cancel it and prevent the raffle from being started. This means no raffle can be started, which makes the protocol completely unusable and useless, isn't this considered as DoS?

This is repeating the same arguments as previously. Based on the 1-block DOS rule, the raffle can be started right after the attack is executed.

> which is the fact that this issue breaks README assertion that the protocol should be able to start a raffle.

The statement in the README is different, and it was already explained above why it's not broken. No need to repeat the same arguments.

> Besides this, the protocol team agrees with the existence of the vuln and confirmed they will fix it.

The sponsor confirming and fixing the issue has no effect on the judgment.

> this is not a 1 block DoS due to creating a raffle being a cross-chain message. The admin would have to wait at least a few blocks to try again because creating a raffle requires a cross chain message.

Fair point, thank you for that, but still, considering the one iteration of the attack, the raffle can be started in a couple of minutes.

> Also, although the fee cost is small each time, the attack can be carried out infinite times, exceeding small and finite funds (I believe there is also a rule for this).

Good point as well, although this rule was merged after the contest started, it's still fair to say that by repeating this attack, the attacker can cause larger material damage.

> *You may have questions why, in the case of 1-block DOS, the repetition of the attack is not considered, but in the case of material damage, it can be considered. It's simple: the thing is in the material damage. If the damage is not material (i.e. DOS), then it's considered griefing for gas and calling the function a second time is not a Medium severity impact (unless it's time-sensitive). If the damage is material, then DOS rules do not apply (including the 1-block DOS rule).*

In that sense, I agree that the material damage can exceed small and finite amounts. Moreover, it can lead to the loss of all the Link tokens in the contract, with the attack cost (gas fees on Avalanche) being much less. Hence, I agree that even high severity is appropriate. The decision is to reject the escalation and leave the issue as it is.

@DemoreXTess, these two findings are very different, and just breaking the invariant is not sufficient for the issue to be Medium or High. I believe the extensive discussions on both issues show how different they are.

WangSecurity

Result: High Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- neko-nyaa: rejected

Issue H-3: Method refundPlayers doesn't update _lockedETH in WinnableTicketManager

Source: https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/138

Found by

0x0bserver, 0x6a70, 0x73696d616f, 0xShahilHussain, 0xbrivan, 0xjarix, 4gontuk, Afriaudit, AllTooWell, AuditorPraise, BlocSoc_Audits, CatchEmAll, Galturok, GenevaRoc, IvanFitro, MSK, Offensive021, Paradox, Penaldo, PeterSR, S3v3ru5, Silvermist, TessKimy, Waydou, almurhasan, anonymousjoe, araj, charles__cheerful, dany.armstrong90, dimi6oni, dimulski, dobrevaleri, gajiknownnothing, gerd, gkrastenov, iamnmt, ihtishamsudo, ironside, irresponsible, matejdb, neko_nyaa, ni8mare, oxelmiguel, oxwhite, p0wd3r, pandasec, pashap9990, phoenixv110, sakshamguruji, shikhar, shui, utsav, vinica_boy, y4y

Summary

The variable `_lockedETH` keeps track of the ETH(AVAX) collected by the raffles that are underway. The owner can't withdraw this amount. If a raffle is cancelled then users get to withdraw their ETH(AVAX) paid to buy tickets. But the `_lockedETH` is not updated. So in the future raffle which do gets completed the owner is supposed to get the ticket amount. But since the `_lockedETH` from previously wasn't set to 0 it having some value leads to that much amount getting stuck in the contract forever.

Root Cause

In `https://github.com/sherlock-audit/2024-08-winnables-raffles/blob/main/public-contracts/contracts/WinnablesTicketManager.sol#L215-L228` the refunded amount should've been subtracted from `_lockedETH` amount. Since it's not updated the owner will not be able to withdraw this much amount ever
`https://github.com/sherlock-audit/2024-08-winnables-raffles/blob/main/public-contracts/contracts/WinnablesTicketManager.sol#L300-L306`

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

1. A new raffle starts. Alice buys tickets worth of 2 ETH(AVAX). The `_lockedETH` is updated from 0 to 2.
2. Bob buys tickets worth of 1 ETH(AVAX). The `_lockedETH` is updated from 2 to 3.
3. Now the Raffle gets cancelled because of some reason. Both Alice and Bob withdraws their money `refundPlayers()`
4. In Future another raffle starts. Both Alice and Bob stakes 2 ETH each. The `_lockedETH` gets updated to 4 + 3 = 7.
5. The Raffle finishes and winner is chosen. The `propagateRaffleWinner()` updated `_lockedETH` to 7-4 = 3.
6. Now when the Owner tries to withdraw the payment which was 4 ETH it reverts since the `_lockedETH` is 3. So the owner is only allowed to withdraw 1ETH. Rest of the 3ETH will be stuck in the contract.

Impact

It leads to locking of ETH(AVAX) in the contract forever that was protocol income.

PoC

Add the following snippet in `/test/TicketManager.js`

```
it('Should be able to refund tickets purchased', async () => {
  const contractBalanceBefore = await
↪   ethers.provider.getBalance(manager.address);
  const userBalanceBefore = await ethers.provider.getBalance(buyer2.address);
  let lockedETH = await manager.getLockedEth()
  console.log("Locked ETH is: ", lockedETH)
  const tx = await manager.refundPlayers(1, [buyer2.address]);
  lockedETH = await manager.getLockedEth()
  console.log("Locked ETH after player unlock is: ", lockedETH)
  const { events } = await tx.wait();
  expect(events).to.have.lengthOf(1);
  const [ event ] = events;
  expect(event.event).to.equal('PlayerRefund');
  const contractBalanceAfter = await ethers.provider.getBalance(manager.address);
  const userBalanceAfter = await ethers.provider.getBalance(buyer2.address);
  expect(contractBalanceAfter).to.eq(contractBalanceBefore.sub(100));
  expect(userBalanceAfter).to.eq(userBalanceBefore.add(100));
  const { withdrawn } = await manager.getParticipation(1, buyer2.address);
  expect(withdrawn).to.eq(true);
```

```
});
```

Output:

Mitigation

Update the _lockedETH variable in `refundPlayers()` as below:

```
function refundPlayers(uint256 raffleId, address[] calldata players) external {
    Raffle storage raffle = _raffles[raffleId];
    if (raffle.status != RaffleStatus.CANCELED) revert InvalidRaffle();
    for (uint256 i = 0; i < players.length; ) {
        address player = players[i];
        uint256 participation = uint256(raffle.participations[player]);
        if (((participation >> 160) & 1) == 1) revert
↪   PlayerAlreadyRefunded(player);
        raffle.participations[player] = bytes32(participation | (1 << 160));
        uint256 amountToSend = (participation & type(uint128).max);
        _lockedETH -= amountToSend;
        _sendETH(amountToSend, player);
        emit PlayerRefund(raffleId, player, bytes32(participation));
        unchecked { ++i; }
    }
}
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/Winnables/public-contracts/pull/5

Issue M-1: Anyone can cancel a raffle with tickets == minTicketsThreshold, griefing all participants

Source: https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/26

Found by

0x0x0xw3, Oblivionis, Offensive021, Silvermist, almurhasan, anonymousjoe, aslanbek, jsmi, kuprum, neko_nyaa, p0wd3r, rbserver

Summary

```
function _checkShouldDraw(uint256 raffleId) internal view {
    Raffle storage raffle = _raffles[raffleId];
    if (raffle.status != RaffleStatus.IDLE) revert InvalidRaffle();
    uint256 currentTicketSold =
↪   IWinnablesTicket(TICKETS_CONTRACT).supplyOf(raffleId);
    if (currentTicketSold == 0) revert NoParticipants();
    if (block.timestamp < raffle.endsAt) {
        if (currentTicketSold < raffle.maxTicketSupply) revert
↪   RaffleIsStillOpen();
    }
    if (currentTicketSold < raffle.minTicketsThreshold) revert
↪   TargetTicketsNotReached();
}
```

```
function _checkShouldCancel(uint256 raffleId) internal view {
    Raffle storage raffle = _raffles[raffleId];
    if (raffle.status == RaffleStatus.PRIZE_LOCKED) return;
    if (raffle.status != RaffleStatus.IDLE) revert InvalidRaffle();
    if (raffle.endsAt > block.timestamp) revert RaffleIsStillOpen();
    uint256 supply = IWinnablesTicket(TICKETS_CONTRACT).supplyOf(raffleId);
    if (supply > raffle.minTicketsThreshold) revert TargetTicketsReached();
}
```

As we can see, once the ticket sale is over, if exactly `minTicketsThreshold` were sold, both `cancelRaffle` and `drawWinner` are available. If anyone (e.g. a participant who does not want to participate anymore, or a griefer) manages to call `cancelRaffle` before anyone calls `drawWinner` for that `raffleId`, it would cancel a raffle that should have been drawn, according to the code comment.

Root Cause

`_checkShouldCancel` does not revert when `supply == raffle.minTicketsThreshold`.

Internal pre-conditions

`currentTicketSold == raffle.minTicketsThreshold` `block.timestamp >= raffle.endsAt` `raffleStatus == RaffleStatus.IDLE`

Attack Path

Anyone calls `cancelRaffle` when `block.timestamp >= raffle.endsAt`, and before `drawWinner` is called.

Impact

Raffle that should be drawn is cancelled.

Mitigation

```
    function _checkShouldCancel(uint256 raffleId) internal view {
        Raffle storage raffle = _raffles[raffleId];
        if (raffle.status == RaffleStatus.PRIZE_LOCKED) return;
        if (raffle.status != RaffleStatus.IDLE) revert InvalidRaffle();
        if (raffle.endsAt > block.timestamp) revert RaffleIsStillOpen();
        uint256 supply = IWinnablesTicket(TICKETS_CONTRACT).supplyOf(raffleId);
-       if (supply > raffle.minTicketsThreshold) revert TargetTicketsReached();
+       if (supply >= raffle.minTicketsThreshold) revert TargetTicketsReached();
    }
```

Discussion

aslanbekaibimov

Escalate

Permanent DoS of `drawWinner` for a subset of drawable raffles (which is arguably a time-sensitive function).

Breaks core contract functionality (drawing a winner) for a subset of drawable raffles, (arguably) rendering the contract useless.

Loss of funds in the form of lost EV for all participants if total value of the tickets was smaller than the prize.

sherlock-admin3

> Escalate
>
> Permanent DoS of `drawWinner` for a subset of drawable raffles (which is arguably a time-sensitive function).
>
> Breaks core contract functionality (drawing a winner) for a subset of drawable raffles, (arguably) rendering the contract useless.
>
> Loss of funds in the form of lost EV for all participants if total value of the tickets was smaller than the prize.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

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

Brivan-26

The root cause is indeed valid, but what's the impact here? Only **if someone** calls `cancelRaffle` before `drawWinner` the raffle will be canceled?

- There is no loss of funds, `refundPlayers` can be called to refund the bidders
- The next raffle can start normally, so there is no *permanent DoS* as stated by the waston

The impact is LOW, and the likelihood is between Low/medium as it requires both:

1. the number of tickets sold should be **exactly** `minTicketsThreshold`
2. Someone needs to call `cancelRaffle` before `drawWinner`

jsmi0703

#340 is a duplicate of this issue.

0xweb333

https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/330 is a duplicate

kuprumxyz

#550 is a partial duplicate of this (the second vulnerability + the second attack path); or this is a partial duplicate of #550, whatever fits.

According to Sherlock rules:

> DoS has two separate scores on which it can become an issue:
>
> - The issue causes locking of funds for users for more than a week.

- The issue impacts the availability of time-sensitive functions (cutoff functions are not considered time-sensitive). If at least one of these are describing the case, the issue can be a Medium. If both apply, the issue can be considered of High severity. Additional constraints related to the issue may decrease its severity accordingly.

Both of the above apply:

- *The user funds are locked for more than a week*: From the link I supplied in my finding, Long Running Raffle (scroll under the "Activity" tab): user funds have been locked **for a month** under this raffle. Anyone can cancel the raffle, thus the funds have been locked for so long in vein.
- *The issue impacts the availability of time-sensitive functions*: cancelling a raffle means that `drawWinner` **can't be called anymore**, i.e. it becomes unavailable. As this function can be called only *before* the raffle is canceled, it is time-sensitive.

Thus, according to the Sherlock rules this may be even a valid High; but it is clearly at least a valid Medium.

mystery0x

A fixed is desired to perfect the conditional check but it's deemed low given the trivial change needed.

kuprumxyz

@mystery0x I would kindly ask to point to the Sherlock docs which allow to qualify the finding as Low if the fix is trivial.

To the best of my knowledge no such docs exist. Moreover, they can't exist. There are literally billions of funds lost due to bugs with trivial fixes. As an example, take a look at the Wormhole Bridge Exploit Incident Analysis. The exploit was for over $320M, but the fix is trivial: replace the deprecated function load_current_index with `load_current_index_checked`.

Brivan-26

It's not about whether the fix is trivial or not, the impact is low and the likelihood is low/medium. This doesn't quality for Medium severity

kuprumxyz

> It's not about whether the fix is trivial or not, the impact is low and the likelihood is low/medium. This doesn't quality for Medium severity

To *that* I have already elaborated above; have nothing to add. The severity is at least Medium; arguably can be a High.

WangSecurity

@kuprumxyz about your points in the previous comment:

> The user funds are locked for more than a week: From the link I supplied in my finding, Long Running Raffle (scroll under the "Activity" tab): user funds have been locked for a month under this raffle. Anyone can cancel the raffle, thus the funds have been locked for so long in vein

I don't think this "lock" can be considered a DOS here, because it's how the protocol should work, i.e. you enter the raffle and your funds are locked until the raffle has finished or gets cancelled. Moreover, I don't think the finding is about this, the finding is about cancelling the raffle before the winner was drawn and I don't think any funds are locked here, since as noted above anyone can call `refundPlayers`.

> The issue impacts the availability of time-sensitive functions: cancelling a raffle means that drawWinner can't be called anymore, i.e. it becomes unavailable. As this function can be called only before the raffle is canceled, it is time-sensitive

But, in that sense, just cancelling the raffle is DOSing the drawing of the winner. Could you elaborate a bit more on why you consider this a time-sensitive function, I don't think the ability to cancel a raffle, makes the function of picking a winner time-sensitive.

kuprumxyz

@WangSecurity the scenario is as follows:

1. A raffle is created.
2. Users are buying tickets. *When users are buying tickets, their funds are locked in the raffle*. As I demonstrated above, a month is a normal duration for a raffle, so the funds are locked for a month.
3. The raffle reaches the end of its duration (a month), and the number of tickets sold reaches `minTicketsThreshold`.
4. At that point, a raffle is both draw-able, but also cancel-able, while it should be *only* draw-able; this is the bug.

We see that "*the user funds are locked for more than a week*". Yes, this is how the protocol works, but the point is that the users lock their funds not just because of some weird desire to lock them for a month; they lock them because each of them wishes to participate in a raffle, and may be win. Depriving them of the right to draw a winner, and of winning a raffle, is the same as DoS of their funds for a month.

> But, in that sense, just cancelling the raffle is DOSing the drawing of the winner. Could you elaborate a bit more on why you consider this a time-sensitive function, I don't think the ability to cancel a raffle, makes the function of picking a winner time-sensitive.

Point 4. above is the bifurcation point, at which only one of the two alternatives is possible: either the winner will be drawn ***xor*** the raffle will be canceled. From the time point when one happens, another can't happen anymore; thus drawing a winner is time-sensitive *for this specific combination of parameters*.

kuprumxyz

@WangSecurity, here is another point to consider:

If the above scenario happens, i.e. users were waiting for a month, and then the raffle got canceled though it should have been drawn, the trust of users into the protocol will be severely undermined, and most likely they won't like to participate anymore. This may easily lead to the protocol collapse.

aslanbekaibimov

I would like to add that in some raffles, at the end of the buying period, someone will buy just enough tickets so the raffle reaches minTicketsThreshold, reasonably expecting that it would guarantee that the raffle will be drawn.

Brivan-26

> @WangSecurity, here is another point to consider:
>
> If the above scenario happens, i.e. users were waiting for a month, and then the raffle got canceled though it should have been drawn, the trust of users into the protocol will be severely undermined, and most likely they won't like to participate anymore. This may easily lead to the protocol collapse.

This is not a valid impact.

- There is no loss of funds, `refundPlayers` can be called to refund the bidders
- The next raffle can start normally, so there is no permanent DoS neither

The likelihood is between Low/medium as it requires both:

- the number of tickets sold should be exactly `minTicketsThreshold`
- Someone needs to call `cancelRaffle` before `drawWinner`

@aslanbekaibimov

> I would like to add that in some raffles, at the end of the buying period, someone will buy just enough tickets so the raffle reaches minTicketsThreshold, reasonably expecting that it would guarantee that the raffle will be drawn.

Even in such a case, if someone calls `drawWinner`, the draw will happen.

aslanbekaibimov

@Brivan-26

> Even in such a case, if someone calls drawWinner, the draw will happen.

It will happen only if `cancelRaffle` was not called. For raffles that reached `minTicketsThreshold`, `drawWinner` should always be available.

WangSecurity

After additionally considering this, indeed the raffle shouldn't be cancellable when the minTickets threshold is reached. Therefore, this is a break of the contract functionality, since you can cancel the raffle when you shouldn't be able to. Hence, it qualifies for the following:

> Breaks core contract functionality, rendering the contract useless or leading to loss of funds.

Planning to accept the escalation and validate with medium severity. This will be the main issue of the new family, the duplicates are:

- #126
- #210
- #272
- #312
- #479
- #507
- #550 (I see there are 2 issues, but I believe this report focuses slightly more on this issue. FYI, in the future, put these as two issues, since the first scenario breaks the README statement, while the second is a different issue. Hence, in this contest, it would have been 2 valid issues, but we can't separate one report and reward it twice).
- #340
- #330
- #114
- #475

  Note: if there are other missing duplicates, let me know. Also, big thanks to Watsons who didn't escalate their issues and just flagged under the escalation.

kuprumxyz

> I see there are 2 issues, but I believe this report focuses slightly more on this issue. FYI, in the future, put these as two issues, since the first

> scenario breaks the README statement, while the second is a different issue. Hence, in this contest, it would have been 2 valid issues, but we can't separate one report and reward it twice

@WangSecurity, thank you; the advice is highly appreciated.

yashar0x

hey @WangSecurity #114 is a dup of this

0xsimao

@WangSecurity the mentioned comment says

> Minimum number of tickets required to be sold for this raffle

Which means the raffle **can** draw a winner if this amount is reached, which is true. The only thing enforced by this statement is that if not enough tickets are bought, the round can not be drawn, which holds. Just because the round may **also** be cancelled at the exact same number of tickets, it does not mean the comment is broken.

There is no vulnerability in allowing the round to be both drawable and cancelled when the tickets are exactly `minTicketsThreshold`, unless there was a comment specifically saying that rounds should be cancelled only when `minTicketsThreshold` is exceeded, which is not the case.

We should assume people act in their best interest and if they want to ensure the round is drawn, they can just buy the number of tickets left to `minTicketsThreshold + 1` to guarantee it is not cancelled. This is because it is only possible to cancel after the raffle ends, so they have plenty of time to do so.

Almur100

#475 is duplicate of this issue

WangSecurity

I didn't say the comment is broken and that's why the issue is validated. The problem here is that the intended design of the protocol is that the raffle shouldn't be cancellable when the min tickets threshold requirement is passed (i.e. tickets == minTicketsThreshold). Therefore, this issue qualifies for medium severity since it breaks the functionality (the raffle can be cancelled when it shouldn't be) and thus the raffle that should've been finalised, wouldn't be finalised:

> Breaks core contract functionality, rendering the contract useless or leading to loss of funds.

The decision remains the same, accept the escalation and validate with medium severity. The duplication list has been updated.

0xsimao

@WangSecurity could you elaborate on this part?

> We should assume people act in their best interest and if they want to ensure the round is drawn, they can just buy the number of tickets left to minTicketsThreshold + 1 to guarantee it is not cancelled. This is because it is only possible to cancel after the raffle ends, so they have plenty of time to do so.

Due to this, this will never happen unless users want to, which is fair.

Also, this boils down to just 1 difference in the number of tickets needed to cancel, I don't see how this is breaking core functionality as it is a very small error.

yashar0x

hey @WangSecurity i don't see my report (#114) in the duplicate list

WangSecurity

> We should assume people act in their best interest and if they want to ensure the round is drawn, they can just buy the number of tickets left to minTicketsThreshold + 1 to guarantee it is not cancelled. This is because it is only possible to cancel after the raffle ends, so they have plenty of time to do so

It's fair argument, but still, the scenario where the bought tickets equal exactly the minimum tickets threshold and no one wants to buy more is completely viable. I understand that it's only one ticket difference, but this can fairly happen. About the broken functionality, the raffle shouldn't be cancellable when the tickets reach that min threshold, but this report clearly shows how it can be reached. Thus, the raffle that should've been drawn, got cancelled. That's why it qualifies for medium severity as a broken core contract functionality, rendering the contract useless.

The decision remains, accept the escalation and validate with medium severity. The duplication list has been updated.

WangSecurity

Result: Medium Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- aslanbekaibimov: accepted

shikhar229169

@WangSecurity https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/270 The above is also a dup of this.

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/Winnables/public-contracts/pull/23

Issue M-2: The setRole() function grants role instead of removing

Source: https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/53

Found by

0x0x0xw3, 0xAadi, 0xarno, 0xbrivan, 0xnolo, Afriaudit, DrasticWatermelon, Drynooo, Galturok, KaligoAudits, KingNFT, MaanVader, MrPotatoMagic, PTolev, Paradox, Penaldo, PeterSR, PratRed, Saurabh_Singh, Trooper, UAARRR, araj, azanux, boringslav, casper, chaduke, charles__cheerful, dany.armstrong90, denzi_, dimulski, dwaipayan01, iamnmt, ironside, joshuajee, ke1caM, korok, mahmud, matejdb, neko_nyaa, nilay27, petro1912, rogueereggiant, sharonphiliplima, shikhar, thisvishalsingh, unnamed, utsav, ydlee

Summary

Access control in the Winnables Raffles protocol is handled with the `Roles` contract. It works similarly to OpenZeppelin's access control but uses bit flags to determine whether a user has a role. Each user has a bytes32 representing the bitfield of roles. Role `0` is an admin role, allowing its members to grant or deny(remove) roles to other users.

The `setRole(address user, uint8 role, bool status)` function, as it stands, always adds a role by performing a bitwise OR operation. However, it does not handle the removal of roles if the `status` parameter is `false`. This oversight results in incorrect role management within the contracts, potentially leading to accidental privilege grants or the inability to revoke privileges from compromised or revoked accounts.

Root Cause

In `Roles.sol:L29` the `_setRole()` function always adds a role by performing a bitwise OR operation:

https://github.com/sherlock-audit/2024-08-winnables-raffles/blob/main/public-contracts/contracts/Roles.sol#L29-L33

This internal function is used in the `setRole()` function:

https://github.com/sherlock-audit/2024-08-winnables-raffles/blob/main/public-contracts/contracts/Roles.sol#L35-L37

Internal pre-conditions

The `setRole()` function can only be called by the `Admin`.

External pre-conditions

No response

Attack Path

1. `Admin` deploys the `WinnablesTicketTest` contract.
2. `Admin` grants role `1` to `Alice` by calling the `setRole()` function of the `WinnablesTicketTest` contract. The role is granted to `Alice`.
3. `Alice` mints 10 tickets to `Bob` using the role.
4. `Admin` revokes role `1` from `Alice` by calling the `setRole()` function.
5. The role is not removed from `Alice`. She can still mint tickets to `Bob`.

Impact

The improper implementation results in incorrect role management within the contracts, potentially leading to accidental privilege grants or the inability to revoke privileges from compromised or revoked accounts.

PoC

```
describe('Ticket behaviour', () => {
...
    it('Should not be able to mint tickets afer role deny', async () => {
      await (await ticket.setRole(signers[2].address, 1, true)).wait();

      const { events } = await (await
↪   ticket.connect(signers[2]).mint(signers[3].address, 1, 1)).wait();
      expect(events).to.have.lengthOf(2);
      expect(events[0].event).to.eq('NewTicket');
      expect(events[1].event).to.eq('TransferSingle');

      await (await ticket.setRole(signers[2].address, 1, false)).wait();
      await expect(ticket.connect(signers[2]).mint(signers[2].address, 1,
↪   1)).to.be.revertedWithCustomError(
        ticket,
        'MissingRole'
      );
```

```
    });
...
```

Mitigation

Modify the `_setRole()` function to handle both adding and removing roles based on the status parameter:

```
function _setRole(address user, uint8 role, bool status) internal virtual {
    uint256 roles = uint256(_addressRoles[user]);
    if (status) {
        _addressRoles[user] = bytes32(roles | (1 << role));
    } else {
        _addressRoles[user] = bytes32(roles & ~(1 << role));
    }
    emit RoleUpdated(user, role, status);
}
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/Winnables/public-contracts/pull/4

Issue M-3: Admin can unrestrictedly affect the odds of a raffle by setting themselves up with role(1) in `WinnablesTicket`

Source: https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/129

Found by

0x0bserver, 0x73696d616f, 0xShahilHussain, 0xbrivan, Oblivionis, S3v3ru5, dimulski, jennifer37, neko_nyaa

Summary

A core invariant defined in the contest README is that:

> Admins cannot affect the odds of a raffle

While the centralization risk of admin self-minting tickets is noted, the following assumption is noted:

> The existence of max ticket supply and max holdings however guarantees a minimum level of fairness

By setting themselves up with role(1) directly in the `WinnablesTicket` token contract, the admin can bypass all these assumptions (max ticket supply, max holdings), and affect the winning odds with no limit.

Root Cause

First of all, let's look at `mint()` in `WinnablesTickets` contract:

```
function mint(address to, uint256 id, uint256 amount) external onlyRole(1) {
```

https://github.com/sherlock-audit/2024-08-winnables-raffles/blob/main/public-contracts/contracts/WinnablesTicket.sol#L182-L199

It is clear that role(1) can mint unlimited tickets. Furthermore, the admin can also grant themselves the role, bypassing any restrictions in `buyTicket()`. We now investigate the impact (how the results are affected by the admin minting tickets to themselves)

When the raffle results are created, the winner is calculated using the `supply` from the ticket storage.

```
function _getWinnerByRequestId(uint256 requestId) internal view returns(address)
↪ {
    RequestStatus storage request = _chainlinkRequests[requestId];
```

```
    uint256 supply =
↪   IWinnablesTicket(TICKETS_CONTRACT).supplyOf(request.raffleId); // @audit
↪   supplyOf is taken from the ticket
    uint256 winningTicketNumber = request.randomWord % supply;
    return IWinnablesTicket(TICKETS_CONTRACT).ownerOf(request.raffleId,
↪   winningTicketNumber);
}
```

https://github.com/sherlock-audit/2024-08-winnables-raffles/blob/main/public-contracts/contracts/WinnablesTicketManager.sol#L472

By minting (virtually) unlimited tickets to themselves, the admin bypasses all restrictions imposed in ticket purchase, granting themselves victory odds far exceeding the restrictions imposed.

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

1. The admin locks a prize, and starts a raffle as normal. People starts buying tickets to enter the raffle.
2. Admin grants themselves role(1) on the `WinnablesTickets` (not the ticket manager), and mints themselves (or any related party) almost unlimited tickets.
3. Admin bypasses all max ticket restrictions, and said party is virtually guaranteed to be the winner.

Impact

Admin can unrestrictedly affect the odds of a raffle, breaking protocol invariant

PoC

No response

Mitigation

No response

Discussion

neko-nyaa

Escalate. As per the README, the admin should not have a method to tamper with the odds of drawing a raffle. This submission shows one such method.

sherlock-admin3

> Escalate. As per the README, the admin should not have a method to tamper with the odds of drawing a raffle. This submission shows one such method.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

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

Brivan-26

Please note that #79 and #86 so far are dups of this

rickkk137

- First:

  > If the protocol team provides specific information in the README or CODE COMMENTS, that information stands above all judging rules. In case of contradictions between the README and CODE COMMENTS, the README is the chosen source of truth

Sponsor clearly mention every admin's action which prevent winners to claim their prize is broken assumption

> The principles that must always remain true are: Winnables admins cannot do anything to prevent a winner from withdrawing their prize

Second: #86 root cause:admin can send arbitrary message to `WinnablesPrizeManager::_ccipReceive` and can change winner #129 #79: root cause:admin can change supply tokens after randomWord will be provided by chainlink node,

hence we cannot classified them with eachother because root causes and fixes are different

0x-Shahil-Hussain

#482 is also a dup of this

tejas-warambhe

#163 is dup similar to this.

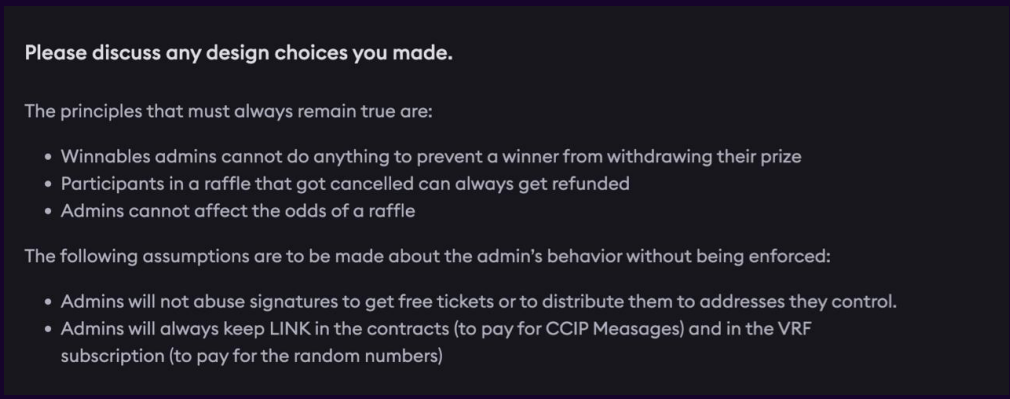
S3v3ru5

The following two issues have the same impact as this one but different root cause. #289 - admin can send arbitrary message to WinnablesPrizeManager::_ccipReceive and can change winner by using _setCCIPCounterpart #271 - Admin can use integer overflow after the winner is declared to overwrite the winning ticket owner.

Mentioning them here as I cannot escalate them

Joshuajee

I doubt the validity of this though, there is an underlying assumption that the admin would never do this. Admin will never abuse the signature to mint free tickets for themselves or addresses that they control



Brivan-26

@Joshuajee You misunderstand, this issue is not about abusing the signature. the issue is about the admin minting directly the ticket by interacting with `WinnableTickets` directly after the random number is fulfilled. He can easily determine the winner after the `randomword` is known

Joshuajee

That was about a separate issue which I validated, this one is talking about buying tickets during the raffle which the admin won't do as mentioned in the readme. The one I validated has to do with the admin minting it after the winner has been selected i.e they are choosing winners not the VRF.

johnson37

#88 is dup similar to this.

S3v3ru5

> That was about a separate issue which I validated, this one is talking about buying tickets during the raffle which the admin won't do as mentioned in the readme. The one I validated has to do with the admin

> minting it after the winner has been selected i.e they are choosing winners not the VRF.

@Joshuajee are you referring to this one #271 ?

Joshuajee

Yes exactly @S3v3ru5

mystery0x

All admin related issues have been discussed with the HOJ and the sponsor who wished the following governing statements could have been more carefully written and/or omitted:

"The protocol working as expected relies on having an admin creating raffles. It should be expected that the admin will do their job. However it is not expected that the admin can steal funds that should have ended in a raffle participant's wallet in any conceivable way."

Much as many of the findings suggest good fixes, the conclusion is that they will all be deemed low unless it's beyond the trusted admin control.

Joshuajee

This is not what the Readme said and this is making the work of security researchers and people who judged the contest as wasted efforts.

Brivan-26

The issues submitted in this report clearly breaks the following invariant:

> Admins cannot affect the odds of a raffle

after the random word is fulfilled—meaning after the fulfillRandomWords function is called and the random word is known. At this point, the admin can determine the random word and exploit it by calling the WinnablesTicket::mint function to mint additional tickets. By increasing the ticket supply, the admin can manipulate the outcome of the raffle, thereby controlling who wins

Check #79 for better details

WangSecurity

I agree this issue breaks the invariant from the README:

> Admins cannot affect the odds of a raffle

And I don't think it breaks the following statement, i.e. there's no abuse of the signatures involved. The tickets are bought, but not due to the abused signature:

> Admins will not abuse signatures to get free tickets or to distribute them to addresses they control

Hence, this issue is medium severity, planning to accept the escalation. This issue will be the main one. The issue will be based on the conceptual mistake of the admin being able to affect the odds of a raffle, therefore, the duplicates are also the issues about this invariant broken:

- #79
- #482
- #271
- #88
- #191
- #303
- #224
- #580
- #229

  *Note: there are other similar reports (even mentioned above), most likely because they break the other invariant (about the admin preventing the winner from claiming their prize), or it may have been missed. In the second case, if you're confident your report is about affecting the odds of a raffle, please let me know.

Brivan-26

@WangSecurity #37 is NOT a valid duplicate (it talks about Using ERC721.transferFrom() instead of safeTransferFrom())

rickkk137

@WangSecurity I hope,this can be helpful

root cause:admin can change total supply and mint more ticket and finally change winner

- #79
- #482
- #271
- #88
- #303
- #224
- #337

- #580

root cause:admins can send arbitrary message to `WinnablesPrizeManager` contract and change winner in favor of themself

- #86
- #98
- #289

#273 is not a valid duplicate let's assume maxTicketSupply = 100 and maxHoldings = 100 and admin give signture to specific user to buy 100 ticket and that user buy all 100 ticket,hence other users cannot buy ticket

```
unchecked {
    if (supply + ticketCount > raffle.maxTicketSupply) revert TooManyTickets();
}
```

https://github.com/sherlock-audit/2024-08-winnables-raffles/blob/81b28633d0f450e33a8b32976e17122418f5d47e/public-contracts/contracts/WinnablesTicketManager.sol#L414 becuase of this line its mean if a user can buy all tickets,other users cannot participate is that raffle

0xshivay

@WangSecurity Issue #580 is a valid duplicate of issue #129.

Brivan-26

@WangSecurity @rickkk137 #86 #98 #289 are dups of #277 not #129

All of the three mentioned issues above **share the same root cause (updating the ccip counterpart to a contract controlled by the admin to control the winner)** and they highlight different impacts of #277

The issue here is totally different: after the random word is fulfilled and the random word is known, the admin can call the WinnablesTicket::mint function to mint additional tickets. By increasing the ticket supply, the admin can manipulate the outcome of the raffle, thereby controlling who wins

#273 is an invalid issue and describes an intended behavior (check my comment there)

WangSecurity

@rickkk137 @Brivan-26 Yes, you're correct, these issues should be duplicated with #277, excuse me for that. Also, correct about #273. Updated the duplication lists.

The decision remains, accept the escalation and apply the changes from my previous comments (it was edited and it's up-to-date).

Brivan-26

Thanks for your reconsideration @WangSecurity Can you provide more explanation as to why Medium severity is appropriate and not High? As far as I understood from your comment, your decision is based:

1. there's no abuse of the signatures involved.
2. The issue will be based on the conceptual mistake of the admin being able to affect the odds of a raffle

For the first point, indeed, there is no signature abuse, but it is not involved in the attack context. For the 2nd point, this issue is not about a *mistake*, but about intentional manipulation of raffle outcomes

The impact of this issue is affecting the odds of the raffles which prevents the real winner from receiving his rewards and thus losing the amount he sold tickets with (especially if he bought a large amount of tickets to increase his win chances). So, as per sherlock rules, this issue should be HIGH

WangSecurity

> Can you provide more explanation as to why Medium severity is appropriate and not High?

This issue is valid only because it's breaking the invariant from the README that the admin cannot affect the odds of the raffle. If there were no such invariant, this would be invalid. Hence, it should be medium severity.

The decision remains as in this comment.

0xsimao

#229 is a dup

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- neko-nyaa: accepted

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/Winnables/public-contracts/pull/20

Issue M-4: Admin can prevent raffle winner from claiming their reward

Source: https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/277

Found by

0rpse, 0x0bserver, 0x73696d616f, IvanFitro, Offensive021, Oxsadeeq, Paradox, S3v3ru5, aslanbek, charles__cheerful, dimah7, dimulski, durov, iamnmt, jennifer37, neko_nyaa, p0wd3r, pashap9990, philmnds, shaflow01, tjonair

Summary

By whitelisting their address and sending a WINNER_DRAWN CCIP message to PrizeManager, Admin can steal the raffle prize, which breaks the invariant from README:

> Winnables admins cannot do anything to prevent a winner from withdrawing their prize

Root Cause

In WinnablesPrizeManager, Admin can add/remove any address as CCIP counterpart at any time.

Internal pre-conditions

A raffle is able to reach REQUESTED stage (enough tickets were sold).

Attack Path

1. Admin sends NFT (or token/ETH) to PrizeManager.
2. Admin calls PrizeManager#lockNFT.
3. Admin calls TicketManager#createRaffle.
4. Raffle (ticket sale) ends successfully.
5. drawWinner is called.
6. Admin adds himself as PrizeManager's CCIP counterpart and sends WINNER_DRAWN CCIP message with his own address as `winner` to PrizeManager; Admin removes TicketManager from PrizeManager's CCIP counterparts.

7. Chainlink VRF request from step 5 is fulfilled.
8. TicketManager#propagate is called, which propagates Alice as the winner of the raffle, but CCIP message reverts on the destination chain with `UnauthorizedCCIPSender()`.
9. Admin claims reward from PrizeManager.

Impact

Admin steals the raffle reward.

Mitigation

Admin should not be able to add or remove CCIP counterparts during Raffles.

Discussion

aslanbekaibimov

Escalate

Sherlock rules:

> The protocol team can use the README (and only the README) to define language that indicates the codebase's restrictions and/or expected functionality. Issues that break these statements, irrespective of whether the impact is low/unknown, will be assigned Medium severity. High severity will be applied only if the issue falls into the High severity category in the judging guidelines.
>
> > Example: The README states "Admin can only call XYZ function once" but the code allows the Admin to call XYZ function twice; this is a valid Medium

README:

> The principles that must always remain true are:
>
> - Winnables admins cannot do anything to prevent a winner from withdrawing their prize

sherlock-admin3

> Escalate
>
> Sherlock rules:
>
> > The protocol team can use the README (and only the README) to define language that indicates the codebase's restrictions

> > and/or expected functionality. Issues that break these statements, irrespective of whether the impact is low/unknown, will be assigned Medium severity. High severity will be applied only if the issue falls into the High severity category in the judging guidelines.
> >
> > > Example: The README states "Admin can only call XYZ function once" but the code allows the Admin to call XYZ function twice; this is a valid Medium
>
> README:
>
> > The principles that must always remain true are:
> >
> > - Winnables admins cannot do anything to prevent a winner from withdrawing their prize

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

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

mystery0x

All admin related issues have been discussed with the HOJ and the sponsor who wished the following governing statements could have been more carefully written and/or omitted:

"The protocol working as expected relies on having an admin creating raffles. It should be expected that the admin will do their job. However it is not expected that the admin can steal funds that should have ended in a raffle participant’s wallet in any conceivable way."

Much as many of the findings suggest good fixes, the conclusion is that they will all be deemed low unless it's beyond the trusted admin control.

WangSecurity

After additionally considering this issue, I agree that it breaks an invariant from the README, so should be valid based on the following rule:

> The protocol team can use the README (and only the README) to define language that indicates the codebase's restrictions and/or expected functionality. Issues that break these statements, irrespective of whether the impact is low/unknown, will be assigned Medium severity. High severity will be applied only if the issue falls into the High severity category in the judging guidelines.

Planning to accept the escalation and validate this issue with medium severity. This report will be the main one and the duplication of the family will be based on the conceptual mistake of admins being able to steal funds. The duplicates are:

- #25
- #62
- #63
- #161
- #217
- #254
- #286
- #524
- #462
- #350
- #117
- #163

Additional duplicates:

- #289
- #613
- #228
- #595
- #100
- #98
- #86
- #348
- #414

  Note: if there are any missing duplicates, please let me know.

shaflow01

@WangSecurity #348 is duplicates

Brivan-26

@WangSecurity Counterpart is a core contract that is needed for cross-chain communication, you can not prevent the admin from updating the counterpart contract (as suggested by the report and the duplicates). This report even suggests preventing the admin from updating the counterpart when the raffle is already ongoing, what if the admin sets a malicious counterpart before the raffle starts?

I believe the usage of counterpart by the admin falls under the trust assumption.

rickkk137

@WangSecurity #86 is dup of #277 also

S3v3ru5

#289 is also a duplicate of this one

0xshivay

@WangSecurity Issue #613 is a duplicate of this issue.

0xsimao

#228 is a dup.

lolka8

#595 is a dup of this

iamnmt

#100 is a dup

WangSecurity

> Counterpart is a core contract that is needed for cross-chain communication, you can not prevent the admin from updating the counterpart contract (as suggested by the report and the duplicates). This report even suggests preventing the admin from updating the counterpart when the raffle is already ongoing, what if the admin sets a malicious counterpart before the raffle starts? I believe the usage of counterpart by the admin falls under the trust assumption.

It's a very fair point and it would be invalid under normal circumstances, but here there's a statement in the README that an admin cannot prevent the winner from getting their prize, while this report shows that an admin can do that. Hence, it should be valid, based on the following:

> The protocol team can use the README (and only the README) to define language that indicates the codebase's restrictions and/or expected functionality. Issues that break these statements, irrespective of whether the impact is low/unknown, will be assigned Medium severity.

> High severity will be applied only if the issue falls into the High severity category in the judging guidelines.

As for the other issues, I've added them into the previous comment into the "additional duplicates" question. Planning to accept the escalation and apply the changes in this comment.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- aslanbekaibimov: accepted

0xvj

@WangSecurity #414 is a valid duplicate of this issue.

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/Winnables/public-contracts/pull/22

Issue M-5: Users buying too many tickets will DoS them and the protocol if they are the winner due to OOG

Source: https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/398

Found by

0x73696d616f, Oblivionis, S3v3ru5, TessKimy, kuprum, neko_nyaa

Summary

`WinnablesTicket` stores `nft` ownership by setting the first minted nft id ownership to the user minting and all the next minted nfts remain as `0`. This means it always costs the same to mint, but the `ownerOf()` function becomes much more expensive, to the point where it may cause OOG errors. In this case, the user is able to buy tickets via WinnablesTicketManager::buyTickets(), the draw is made in WinnablesTicketManager::drawWinner() and the chainlink request is fulfilled with the winner in WinnablesTicketManager::fulfillRandomWords(). However, in WinnablesTicketManager::propagateWinner(), it reverts due to OOG when calling WinnablesTicket::ownerOf().

Root Cause

In `WinnablesTicket:97-99`, it may run out of gas if enough tickets were bought.

Internal pre-conditions

1. Admin sets max holdings and max tickets to a number of at least 4000 or simillar.

External pre-conditions

None.

Attack Path

1. Users by tickets via WinnablesTicketManager::buyTickets()‘
2. The draw is made in `WinnablesTicketManager::drawWinner()`
3. The chainlink request is fulfilled with the winner in ‘WinnablesTicketManager::fulfillRandomWords()]
4. In `WinnablesTicketManager::propagateWinner()`, it reverts due to OOG when calling `WinnablesTicket::ownerOf()`.

Impact

DoSed winner and protocol `ETH` from the raffle.

PoC

The following test costs 9 million gas, more than the block limit on Avalanche.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import {WinnablesPrizeManager, LinkTokenInterface} from
↪  "contracts/WinnablesPrizeManager.sol";
import {WinnablesTicket} from "contracts/WinnablesTicket.sol";
import {IRouterClient} from
↪  "@chainlink/contracts-ccip/src/v0.8/ccip/interfaces/IRouterClient.sol";
import {Client} from
↪  "@chainlink/contracts-ccip/src/v0.8/ccip/libraries/Client.sol";

contract WinnablesPrizeManagerTest is Test {
    WinnablesPrizeManager public winnablesPrizeManager;
    WinnablesTicket public winnablesTicket;
    LinkTokenInterface public link;
    IRouterClient public router;
    address public owner;


    function setUp() public {
        vm.createSelectFork(vm.envString("ETHEREUM_RPC_URL"));
        link = LinkTokenInterface(0x514910771AF9Ca656af840dff83E8264EcF986CA);
        router = IRouterClient(0x80226fc0Ee2b096224EeAc085Bb9a8cba1146f7D);
        owner = makeAddr("owner");

        vm.prank(owner);
        winnablesPrizeManager = new WinnablesPrizeManager(address(link),
↪  address(router));

        vm.prank(owner);
        winnablesTicket = new WinnablesTicket();
    }

    function test_POC_OwnerOfDos() public {
        vm.prank(owner);
        winnablesTicket.setRole(owner, 1, true);

        vm.prank(owner);
```

```
        winnablesTicket.mint(owner, 1, 4000);

        winnablesTicket.ownerOf(1, 3999);
    }
}
```

Mitigation

Set a reasonable cap for max holdings.

Discussion

0xsimao

Escalate

This finding should be valid as it was not known and is outside what is considered admin responsability to set reasonable values (it is not simply a matter of array length, for example).

sherlock-admin3

> Escalate
>
> This finding should be valid as it was not known and is outside what is considered admin responsability to set reasonable values (it is not simply a matter of array length, for example).

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

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

kuprumxyz

#413 is a duplicate of this, or vice versa; whatever fits. A couple more arguments of why this is a **valid High**:

- In my finding I arrived at the estimate of `12608` tickets in a batch for this issue to be triggered; the present finding provides an even more precise value of `~4000`, which is completely realistic even in the current system. Citing from my finding:

  > Long Running Raffle has max tickets 60000, out of which at the time of writing 7043 are sold, and the largest sold batch contains 4347 tickets.

- Sherlock rules say:

    How to identify a high issue:

    - Definite loss of funds without (extensive) limitations of external conditions.

According to the rules, this is a **valid High** because:

- **The loss of funds is definite**: the funds are locked permanently, as neither the winner can be drawn, nor the raffle can be canceled.
- **There are no limitations of external conditions**: the bug will trigger by itself as long as the internal conditions are met (the winning ticket is at the end of a large enough batch of bought tickets).

DemoreXTess

#331 is a duplicate of this

S3v3ru5

#349

mystery0x

This is being linked to https://github.com/sherlock-audit/2024-08-winnables-raffles-judging/issues/142 where the sponsor has disputed this finding alleging it would require admin mistake via the API misconfiguration allowing players to buy excessive amount of tickets, as already clarified in the discord channel.

As such, this finding will be low at best.

kuprumxyz

@mystery0x, I respectfully disagree.

- According to Sherlock's rules, this is a **valid High**: see my previous comment on that.
- What sponsor says about an external API is irrelevant for the contest:
    - What's audited are the contracts, not the admin's API. API has not been made accessible or even known to Watsons. The contracts do allow this behavior.
    - Before this finding, the restrictions on the number of tickets **was not imposed** by the API, or even known to the admins, as follows from sponsor's comments on Discord.
    - In any case, what the sponsor says now, *after the fact*, about these restrictions, is irrelevant. What is relevant is that according to Sherlock's

hierarchy of truth the number of bought tickets in one transaction could be `type(uint16).max`.

- Here is the relevant information from the README:

  Q: Are there any limitations on values set by admins (or other roles) in the codebase, including restrictions on array lengths? None other than the restrictions inherent to the types used (for example: `WinnablesTicketManager::buyTickets()` take a `uint16` argument for `ticketCount`. That means the maximum number of tickets that can be purchased in a single transaction is `type(uint16).max`)

Brivan-26

Buying tickets is an off-chain mechanism, if a user can buy too many tickets to DoS the `ownerOf` function, it would be because of the API allowing so, which is Out of scope

DemoreXTess

@Brivan-26 @kuprumxyz

This message clarify everything. I believe this README proves it to validate this issue.

> @mystery0x, I respectfully disagree.
>
> - According to Sherlock's rules, this is a **valid High**: see my previous comment on that.
> - What sponsor says about an external API is irrelevant for the contest:
>   - What's audited are the contracts, not the admin's API. API has not been made accessible or even known to Watsons. The contracts do allow this behavior.
>   - Before this finding, the restrictions on the number of tickets **was not imposed** by the API, or even known to the admins, as follows from sponsor's comments on Discord.
>   - In any case, what the sponsor says now, *after the fact*, about these restrictions, is irrelevant. What is relevant is that according to Sherlock's hierarchy of truth the number of bought tickets in one transaction could be `type(uint16).max`.
> - Here is the relevant information from the README:
>
>   **Q: Are there any limitations on values set by admins (or other roles) in the codebase, including restrictions on array lengths?** None other than the restrictions inherent to the types used (for example:

> `WinnablesTicketManager::buyTickets()` take a `uint16` argument for `ticketCount`. That means the maximum number of tickets that can be purchased in a single transaction is `type(uint16).max`)

Brivan-26

The only statement that *could* validate this issue is:

> Before this finding, the restrictions on the number of tickets was not imposed by the API, or even known to the admins, as follows from sponsor's comments on Discord.

However, as the sponsor said, we are not auditing the API. The ticket purchase flow is very clear, users buy tickets from the API, the API generates the signatures and then the users call the smart contract.

So, the purchase is happening off-chain and the root reason for this finding is the fact that users can buy too many tickets. The root reason is off-chain

kuprumxyz

@Brivan-26 you are completely overturning the reasoning. The Watsons, when they are participating in a contest, are not aware of anything besides what's stated in Sherlock's hierarchy of truth. If there were any restrictions to the inputs, they should have been made known in the README. If this is not done (and in fact as we see the opposite is done, specifying `type(uint16).max` as the range) ==> this is the valid finding.

If we followed your reasoning, *absolutely any finding can be invalidated by a sponsor after the contest*: they would come and say "*There is an API that will restrict the inputs to this and that function, and the finding is no more*". Let me repeat here the allegory I have already written elsewhere:

> *A careless person drives a car, holding the steering wheel with one hand, and with the other hand chatting on a smartphone. The policeman notices that driver's eyes are not on the road, but on a phone, and that there is a tree trunk right ahead; so the policeman whistles sharply to prevent the car crash. The driver breaks sharply, leaves the car, and instead of thanking the officer says "You sharp whistle scared me to death! Why were you whistling?!? I was holding the steering wheel, my foot was on the breaking pedal, I can always break in time!"*
>
> The officer was seeing what he was seeing: a car driving right into the tree trunk, and the driver looking not ahead of him, but on the phone. The officer had no evidence that the person was even aware of the tree ahead; all evidence was to the opposite; so he was right to whistle and prevent an accident.

Brivan-26

Hey @kuprumxyz I didn't say at all that the sponsor can invalidate the finding. I'm just referring to how ticket purchase works. If users can buy tickets in the smart contracts, I'd totally agree with you that this is a valid issue. But, the root cause here does not happen in the smart contracts (buying too many tickets) but in an off-chain API.

I said my arguments here, I can't provide more.

kuprumxyz

Hey @Brivan-26, the root cause happens exactly in the smart contracts:

- The root cause is in WinnablesTicket::ownerOf();
- which is triggered by buying tickets via WinnablesTicketManager::buyTickets();
- and the valid input to this function is `type(uint16).max`, as specified directly in the README:

> `WinnablesTicketManager::buyTickets()` take a `uint16` argument for `ticketCount`. That means the maximum number of tickets that can be purchased in a single transaction is `type(uint16).max`

That's it. I also have nothing to add.

Brivan-26

- The root cause is **not** in `ownerOf`, the impact is located on `ownerOf` (OOG)
- Buying tickets is **not** via `buyTickets` function, buying tickets happens off-chain, users just provide the signature generated by the API on `buyTickets` to claim the tickets. I don't know why you keep avoiding this fact

kuprumxyz

@Brivan-26 I am tired of this discussion tbh... we are running in circles. If you want, take a look at my finding #413 for the explanation why `ownerOf` is the root cause (and how to refactor it to mitigate the issue), and why `buyTickets` is the internal precondition.

I would kindly ask @WangSecurity to offer their view; and will refrain from any further comments until then.

WangSecurity

I'm not sure I understand that part about the API. The README says:

> Ticket purchases need to be approved by the API which grants a signature

It doesn't say it will not allow users to buy big batches of tickets. Moreover, we have the following line:

> That means the maximum number of tickets that can be purchased in a single transaction is type(uint16).max)

Hence, each user can buy as many tickets as they want, even though they have to be approved by the API. @Brivan-26 I may need your clarification as to why you think this issue is invalid.

Brivan-26

@WangSecurity The root reason for this issue is users buying as many tickets as they want, correct? Now:

- the root reason is the purchase of the tickets, happens on the API => off-chain
- The mitigation is restricting how many tickets a user can purchase, where does this mitigation take place? Again, Off-chain

I can't provide more than the above two points

debugging3

Internal pre-conditions of this report is:

> Admin sets max holdings and max tickets to a number of at least 4000 or simillar.

Sherlock Rule is:

> Exception: In case the array length is controlled by the trusted admin/owner or the issue describes an impractical usage of parameters to reach OOG state then these submissions would be considered as low.

From the above, Setting max holdings and max tickets smaller than 4000 is the admin's responsibility. Therefore this report should be considered as low.

Oblivionis214

> Exception: In case the array length is controlled by the trusted admin/owner or the issue describes an impractical usage of parameters to reach OOG state then these submissions would be considered as low.

This statement is overriden by

> III. Sherlock's standards: Hierarchy of truth: If the protocol team provides no specific information, the default rules apply (judging guidelines).
>
> If the protocol team provides specific information in the README or CODE COMMENTS, that information stands above all judging rules. In case of contradictions between the README and CODE COMMENTS, the README is the chosen source of truth.

Brivan-26

@Oblivionis214 The README did not override the following rule:

> In case the **array length is controlled** by the trusted admin/owner or the issue describes an impractical usage of parameters to reach OOG state then these submissions would be considered as low

The API is still controlling the number of tickets a user can purchase. Check this comment

DemoreXTess

@WangSecurity

While @Brivan-26 makes a valid point regarding this submission, I still believe the issue is valid due to the contents of the README file. As @kuprumxyz noted, the README explicitly mentions a possible limit on the number of tickets in the buyTicket function.

kuprumxyz

I didn't want to interfere anymore, but being called upon by @DemoreXTess I need to... Having

> III. Sherlock's standards: Hierarchy of truth: If the protocol team provides no specific information, the default rules apply (judging guidelines). If the protocol team provides specific information in the README or CODE COMMENTS, that information stands above all judging rules. In case of contradictions between the README and CODE COMMENTS, the README is the chosen source of truth.

and then

> @Oblivionis214 The README did not override the following rule:
>
> > In case the **array length is controlled** by the trusted admin/owner or the issue describes an impractical usage of parameters to reach OOG state then these submissions would be considered as low
>
> The API is still controlling the number of tickets a user can purchase. Check this comment

@Brivan-26 you arrived at the logical contradiction which is:

- how is it possible that "*If the protocol team provides specific information in the README or CODE COMMENTS, that information stands above all judging rules.*"
- and then you say "*The README did not override the following rule ...*"?

- *All* means *every single one*.

Brivan-26

@kuprumxyz There is no contradiction. I said that the README or CODE COMMENT did not override the following rule:

> In case the **array length is controlled by the trusted admin/owner**...

The number of tickets a user can purchase is still controlled by the API that is controlled by the admin.

Oblivionis214

Hierarchy of truth: If the protocol team provides no specific information, the default rules apply (judging guidelines).

If the protocol team provides specific information in the README or CODE COMMENTS, that information stands above all judging rules.

Readme did override the rule about array.

> WinnablesTicketManager::buyTickets() take a uint16 argument for ticketCount. That means the maximum number of tickets that can be purchased in a single transaction is type(uint16).max

This means protocol team want to inform watsons the maximum "array length" in protocol design. So anything below this value should be considered possible.

WangSecurity

To confirm, if I understand correctly, you talk about the `_ticketOwnership` mapping when referring to the array. And it can be controlled by an admin to set max holdings and max tickets to a value that wouldn't cause an OOG issue. That's a fair assumption. But, we have the following line:

> WinnablesTicketManager::buyTickets() take a uint16 argument for ticketCount. That means the maximum number of tickets that can be purchased in a single transaction is type(uint16).max

Thus, it's a possible scenario that the user can buy up to 65535 tickets, and we should consider this: even though it's done off-chain through API, the user is still able to buy this many tickets. Hence, I believe this should be a valid issue with medium severity due to the conditions that the user should buy lots of tickets (more than 4000, since the report is imprecise and the block gas limit on Avalanche is 15 limit IIUC) and the winning ticket should be near to the last ticket. Hence, I believe medium is more appropriate.

Planning to accept the escalation. The duplicates are:

- #413

- #331
- #349
- #339
- #142

  Special thanks to @Brivan-26 for telling on other issues that the main discussion is here.

@mystery0x are there missing duplicates?

neko-nyaa

#431 is not a dupe. That issue was always judged correctly.

kuprumxyz

@WangSecurity,

I agree, #431 is not a dupe. I should also note that per Sherlock's rules, #339 is not a dupe either, as it doesn't include a coded PoC.

We have Requirements:

> PoC is required for all issues falling into any of the following groups:
>
> - non-obvious ones with complex vulnerabilities/attack paths
> - issues for which there are non-trivial limitations/constraints on inputs, to show that the attack is possible despite those
> - issues related to precision loss
> - reentrancy attacks
> - ***attacks related to the gas consumption and/or reverting message calls***

Also we have Duplication rules:

> The duplication rules assume we have a "target issue", and the "potential duplicate" of that issue needs to meet the following requirements to be considered a duplicate.
>
> - Identify the root cause
> - Identify at least a Medium impact
> - Identify a valid attack path or vulnerability path
> - Fulfills other submission quality requirements (e.g. ***provides a PoC for categories that require one***)

> Only when the "potential duplicate" meets all four requirements will the "potential duplicate" be duplicated with the "target issue", and all duplicates will be awarded the highest severity identified among the duplicates.
>
> Otherwise, if the "potential duplicate" doesn't meet all requirements, the "potential duplicate" will not be duplicated but could still be judged any other way (solo, a duplicate of another issue, invalid, or any other severity)

neko-nyaa

Disagree. The rule never states that the PoC has to be coded. Submission #339 has shown all the calculations needed to prove that the number in range can cause a DoS, from Avalanche's block gas limit, to the cost of each storage read, down to the number of tickets that can be purchased, showing that a value in the range can indeed cause the issue, which is sufficient as a proof.

Brivan-26

@WangSecurity

> To confirm, if I understand correctly, you talk about the _ticketOwnership mapping when referring to the array. And it can be controlled by an admin to set max holdings and max tickets to a value that wouldn't cause an OOG issue. That's a fair assumption. But, we have the following line: WinnablesTicketManager::buyTickets() take a uint16 argument for ticketCount. That means the maximum number of tickets that can be purchased in a single transaction is type(uint16).max

Can you explain how this scenario is an exception of the following Sherlock rule:

> An admin action can break certain assumptions about the functioning of the code. Example: Pausing a collateral causes some users to be unfairly liquidated or any other action causing loss of funds. This is not considered a valid issue.

The number of tickets that can be purchased is still controlled by the API which is controlled by the admin. So, if the admin sets that large number that causes the OOG, how this is different from other contests in which such actions are judged invalid? It doesn't matter if the sponsor acknowledged that the tickets that can be purchased is very large, **the input is still controlled by the admin**

kuprumxyz

> Disagree. The rule never states that the PoC has to be coded. Submission #339 has shown all the calculations needed to prove that the number in range can cause a DoS, from Avalanche's block gas limit, to the cost of each storage read, down to the number of tickets that can

> be purchased, showing that a value in the range can indeed cause the issue, which is sufficient as a proof.

yeah, may be you are right. I agree that your calculations are enough of a proof.

It's not clarified in the rules how the PoC should look like, and I am used to think about a PoC as a coded one. @WangSecurity, can we get a clarification what is actually understood by PoC in the rules?

Oblivionis214

If a coded poc to show gas comsumption is needed, #331 also has no POC

kuprumxyz

> If a coded poc to show gas comsumption is needed, #331 also has no POC

I was reading #331 for the second time, and realized that I was mistaken previously: it does talk about the same issue as this one.

It's true though that it doesn't have a coded PoC, and also doesn't have proper calculations as in #339 to explain why an OOG would happen; it has only a code path/walkthrough that somehow explains the problem. So it all really depends on the definition of a PoC.

Let's wait for judge's input.

DemoreXTess

The coded PoC is not required for obvious issues. uint16.max limit is really high for a for loop especially while working with storage variables. I believe both #331 and #339 should be accepted because both of them states the problem accurately.

kuprumxyz

On the one hand, I tend to agree. On the other, *specifically because of the rules*, I've spent time to write a coded PoC (otherwise I would not). I would like at least for the future to obtain a clarification whether I need to do it, or not.

WangSecurity

> The number of tickets that can be purchased is still controlled by the API which is controlled by the admin. So, if the admin sets that large number that causes the OOG, how this is different from other contests in which such actions are judged invalid? It doesn't matter if the sponsor acknowledged that the tickets that can be purchased is very large, the input is still controlled by the admin

Thank you, it's a great question. I'll try to explain, but let me know if you still have questions. Even though the input is still controlled by the admin, still there can be type(uint16).max tickets bought in one transaction. But here, the issue doesn't

arise due to admin setting the `maxTickets` and `maxHoldings` to a high, the code can work with them perfectly, unless one user buys too many tokens in such a scenario. Hence, it's not the admin action that breaks the code, that's why i don't apply the rule.

Regarding the POC, indeed the POC can be a very detailed texted one and the coded POC would be ideal, but not mandatory. All the issues have either texted or a coded one, while not of the are perfect, still they're sufficient.

The decision remains, accept the escalation and validate with medium severity.

WangSecurity

Result: Medium Has duplicates

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- 0xsimao: accepted

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/Winnables/public-contracts/pull/12

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.