SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Arrakis
Prepared by: Sherlock
Lead Security Expert: 0x52
Dates Audited: June 14 - June 29, 2023
Prepared on: August 21, 2023

Introduction

Building trustless market making infrastructure & strategies on Uniswap V3. Unlock your liquidity's greatest potential.

Scope

Repository: ArrakisFinance/v2-periphery

Branch: main

Commit: ee6d7c5f3ffb212887db4ec0e595618ea418070f

Repository: ArrakisFinance/v2-manager-templates

Branch: main

Commit: 9b598356f9fb31e4fbaf07acf060e1f60409a7b0

Repository: ArrakisFinance/v2-core

Branch: tests/new-test

Commit: 9133fc412b65c7a902f62f1ad135f062e927b092

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	2

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

kutugu
0xRobocop
BenRai
cergyk
auditsea
ast3ros
rvierdiiev
branch_indigo
p12473
DadeKuma
0x007
elephant_coral
n33k
immeas
0x52
vnavascues
0xpinky
0xDjango
tallo
rugpull_detector
auditor0517
Jeiwan
dipp
0xg0

Issue H-1: Pool deviation check in SimpleManager on re-balance can be bypassed

Source: https://github.com/sherlock-audit/2023-06-arrakis-judging/issues/26

Found by

cergyk, n33k

Summary

In `SimpleManager` a price deviation check is enforced to prevent an operator to add liquidity to a UniV3 pool at a really unfavorable price during rebalance and backrun to extract vault funds. We will show here that this check can be entirely bypassed by a malicious operator.

Vulnerability Detail

Rebalance context

During a call to `SimpleManger.rebalance`, the following operations are run:

- 1/ Enforce price deviation not too large for mint pools: https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-manager-templates/contracts/SimpleManager.sol#L366-L385

and correct slippage parameter for swap: https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-manager-templates/contracts/SimpleManager.sol#L318-L354

- 2/ Remove liquidity on specified UniV3 ranges (we are not going to use it here)
- 3/ Use a low level call to a whitelisted Router to execute a swap https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-core/contracts/ArrakisV2.sol#L334-L336
- 4/ Enforce received amounts from the swap
- 5/ Provide liquidity on specified UniV3 ranges
- 6/ Enforce provided amounts during addLiquidity (these parameters are provided by operator and unbounded, so they can be (0, 0), and check is a noop).

Exploit description

We are going to use the swap step (3/) to imbalance the pools after the check of price deviation (1/) is passed, so the liquidity provided in 5/ is done at a really

unfavorable price, and can be backrun by the operator to extract funds from the vault.

To not trigger the slippage protection after the swap, we are going to use the router to swap on a totally unrelated pool of tokens controlled by the malicious operator: `PSN/PSN2` (PSN stands for Poison).

`PSN` token has a callback in the `_transfer` function to make a large swap on UNIv3 pool where the operator is going to provide liquidity in 5/, to deviate it a lot.

after the call to the router is done, no changes to the balances of the vault have been made, the slippage checks out.

Liquidity provided at 5/ is done at a terrible price for some of the ranges, and the operator backruns for a nice profit.

NB: All these steps can be run inside a flashloan callback, to not use attacker own funds

Impact

An arbitrary amount can be drained from the vault by an operator

Code Snippet

Tool used

Manual Review

Recommendation

Ideally the check on price deviation should be enforced right before the liquidity providing.

Discussion

syjcnss

Escalate

This one should be a valid high and my #187 should be the only duplicate of this.

This one and #187 both showed that the `swap.router.call` calls into a malicious ERC20 contract that manipulates liquidity pool prices to bypass slippage checks and sandwich attack liquidity adding. I've added additional proof in the comment.

#43, #135, #213 are clearly unrelated and should be reevaluated.

#213 should be a dup of #181. They are all about missing checks on burn pools.

sherlock-admin

> Escalate
>
> This one should be a valid high and my #187 should be the only duplicate of this.
>
> This one and #187 both showed that the `swap.router.call` calls into a malicious ERC20 contract that manipulates liquidity pool prices to bypass slippage checks and sandwich attack liquidity adding. I've added additional proof in the comment.
>
> #43, #135, #213 are clearly unrelated and should be reevaluated.
>
> #213 should be a dup of #181. They are all about missing checks on burn pools.

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.

SergeKireev

Agree with @syjcnss

P12473

> PSN token has a callback in the _transfer function to make a large swap on UNIv3 pool where the operator is going to provide liquidity in 5/, to deviate it a lot.

The core of this exploit requires a PSN/PSN2 pool where the PSN tokens have a malicious _transfer function but according to the README `These should be assumed to use SimpleManager.sol and Chainlink oracle feed(s) and thus only support "Major" tokens with suitable chainlink oracle support..`

how easy is it For one to create a malicious token and to provide a feed for it on chainlink?

syjcnss

@P12473 The section `1. on the swap slippage check` in my comment in #187 should be able to anwser your question.

SergeKireev

> > PSN token has a callback in the _transfer function to make a large swap on UNIv3 pool where the operator is going to provide liquidity in 5/, to deviate it a lot.
>
> The core of this exploit requires a PSN/PSN2 pool where the PSN tokens have a malicious _transfer function but according to the README `These should be assumed to use SimpleManager.sol and Chainlink oracle`

> > `feed(s) and thus only support "Major" tokens with suitable chainlink oracle support..`
>
> How easy is it for one to create a malicious token and to provide a feed for it on chainlink?

@syjcnss explained it very well in his linked comment.

I would like to add there is no need for chainlink feeds on poison tokens, since they would only be passed inside the payload used for the router call, and Arrakis never decodes this payload

P12473

> > PSN token has a callback in the _transfer function to make a large swap on UNIv3 pool where the operator is going to provide liquidity in 5/, to deviate it a lot.
>
> The core of this exploit requires a PSN/PSN2 pool where the PSN tokens have a malicious _transfer function but according to the README `These should be assumed to use SimpleManager.sol and Chainlink oracle feed(s) and thus only support "Major" tokens with suitable chainlink oracle support..` How easy is it for one to create a malicious token and to provide a feed for it on chainlink?
>
> @syjcnss explained it very well in his linked comment.
>
> I would like to add there is no need for chainlink feeds on poison tokens, since they would only be passed inside the payload used for the router call, and Arrakis never decodes this payload

Ah yes, apologies for that. Great explanation shared by @syjcnss

SergeKireev

Also obviously agree with @syjcnss that this should be of `high` severity since impact is unbounded theft of funds by an operator

Gevarist

operators are semi trusted, we should consider this issue as medium.

hrishibhat

@Gevarist Readme says:

> However the rebalances that are executed MUST NOT be exploitable by frontrun or sandwich. I think this should be a high based on another duplicate and POC here: https://github.com/sherlock-audit/2023-06-arrakis-judging/issues/187#issuecomment-1636038030

kassandraoftroy

(EDITED after some thought and discussion) ... Hmm I'm still not convinced this is a high. For me a high is when an arbitrary address can extract value, not a semi-trusted party.

Here: `However the rebalances that are executed MUST NOT be exploitable by frontrun or sandwich.` we refer to a THIRD PARTY being able to frontrun or sandwich the rebalance. If ONLY whitelisted manager operator themself can frontrun/sandwich/extract it's still a medium to me, to be fair to "Real" high's that allow an arbitrary address to exploit the contract.

syjcnss

The readme also says about the operator self sandwich,

> HOWEVER in this "Public Vault" setting the "Manager" role is taken by the SimpleManager.sol smart contract which should add additional checks that make it impossible for SimpleManager.operators to frontrun/sandwich their own rebalance transactions and extract value beyond the accepted slippage tolerance defined in the SimpleManager smart contract for any Arrakis V2 vault managed by SimpleManager.

Having this, `However the rebalances that are executed MUST NOT be exploitable by frontrun or sandwich.` doesn't look like only referring to THIRD PARTY sandwich to me.

The attack breaks the above trust assumption in readme and will result in a material loss of funds.

hrishibhat

Result: High Has duplicates Based on the context of the readme, the assumption set is that there is a clear possibility of having a malicious operator. Given that there is a clear loss of funds as show in this issue and #187 Considering this a valid high based on the above comments

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- syjcnss: accepted

Gevarist

We add another deviation check after minting part of rebalance on SimpleManager.sol. PR : https://github.com/ArrakisFinance/v2-manager-templates/pull/26

IAm0x52

Fix looks good. Deviation is checked before and after each rebalance to prevent pool becoming imbalanced between calls

Issue H-2: ArrakisV2Router#addLiquidityPermit2 will strand ETH

Source: https://github.com/sherlock-audit/2023-06-arrakis-judging/issues/183

Found by

0x007, 0x52, 0xpinky, BenRai, DadeKuma, Jeiwan, auditor0517, auditsea, branch_indigo, elephant_coral, kutugu, rvierdiiev, tallo

Summary

Inside ArrakisV2Router#addLiquidityPermit2, `isToken0Weth` is set incorrectly leading to the wrong amount of ETH being refunded to the user

Vulnerability Detail

ArrakisV2Router.sol#L278-L298

```
bool isToken0Weth;
_permit2Add(params_, amount0, amount1, token0, token1);

_addLiquidity(
    params_.addData.vault,
    amount0,
    amount1,
    sharesReceived,
    params_.addData.gauge,
    params_.addData.receiver,
    token0,
    token1
);

if (msg.value > 0) {
    if (isToken0Weth && msg.value > amount0) {
        payable(msg.sender).sendValue(msg.value - amount0);
    } else if (!isToken0Weth && msg.value > amount1) {
        payable(msg.sender).sendValue(msg.value - amount1);
    }
}
```

Above we see that excess msg.value is returned to the user at the end of the function. This uses the value of isToken0Weth to determine the amount to send back to the user. The issue is that `isToken0Weth` is set incorrectly and will lead to ETH being stranded in the contract. `isToken0Weth` is never set, it will always be

`false`. This means that when WETH actually is token0 the incorrect amount of ETH will be sent back to the user.

This same issue can also be used to steal the ETH left in the contract by a malicious user. To make matters worse, the attacker can manipulate the underlying pools to increase the amount of ETH left in the contract so they can steal even more.

Impact

ETH will be stranded in contract and stolen

Code Snippet

ArrakisV2Router.sol#L238-L299

Tool used

Manual Review

Recommendation

Move `isToken0Weth` and set it correctly:

```
-   bool isToken0Weth;
    _permit2Add(params_, amount0, amount1, token0, token1);

    _addLiquidity(
        params_.addData.vault,
        amount0,
        amount1,
        sharesReceived,
        params_.addData.gauge,
        params_.addData.receiver,
        token0,
        token1
    );

    if (msg.value > 0) {
+       bool isToken0Weth = _isToken0Weth(address(token0), address(token1));
        if (isToken0Weth && msg.value > amount0) {
            payable(msg.sender).sendValue(msg.value - amount0);
        } else if (!isToken0Weth && msg.value > amount1) {
            payable(msg.sender).sendValue(msg.value - amount1);
        }
    }
```

Discussion

Gevarist

We consider the issue as a medium issue, some user fund can be lost. Only the stranded eth can be potentially stolen.

ctf-sec

The finding result in lose of fund, recommend maintaining high severity.

Gevarist

@ctf-sec stealing fund is only possible if loss of fund happen before. I would argue that seems like a medium severity issue.

0xpinky

@Gevarist stealing also one of the way for losing funds.. but thats not the only way.. any form of loss is loss only.

Gevarist

You can find a fix on v2-periphery repository, isToken0Weth is now correctly set using _isToken0Weth.

IAm0x52

Fix looks good. `isToken0Weth` is now set correctly in all cases

Issue M-1: Then getAmountsForDelta function at Underlying.sol is implemented incorrectly

Source: https://github.com/sherlock-audit/2023-06-arrakis-judging/issues/8

Found by

0x007, 0xRobocop

Summary

The function `getAmountsForDelta()` at the `Underlying.sol` contract is used to compute the quantity of `token0` and `token1` to add to the position given a delta of liquidity. These quantities depend on the delta of liquidity, the current tick and the ticks of the range boundaries. Actually, `getAmountsForDelta()` uses the sqrt prices instead of the ticks, but they are equivalent since each tick represents a sqrt price.

There exists 3 cases:

- The current tick is outside the range from the left, this means only `token0` should be added.
- The current tick is within the range, this means both `token0` and `token1` should be added.
- The current tick is outside the range from the right, this means only `token1` should be added.

Vulnerability Detail

The issue on the implementation is on the first case, which is coded as follows:

```
if (sqrtRatioX96 <= sqrtRatioAX96) {
      amount0 = SafeCast.toUint256(
          SqrtPriceMath.getAmount0Delta(
               sqrtRatioAX96,
               sqrtRatioBX96,
               liquidity
          )
      );
}
```

The implementation says that if the current price is equal to the price of the lower tick, it means that it is outside of the range and hence only `token0` should be added to the position.

But for the UniswapV3 implementation, the current price must be lower in order to consider it outside:

```
if (_slot0.tick < params.tickLower) {
   // current tick is below the passed range; liquidity can only become in range
 ↪  by crossing from left to
   // right, when we'll need _more_ token0 (it's becoming more valuable) so user
 ↪  must provide it
   amount0 = SqrtPriceMath.getAmount0Delta(
          TickMath.getSqrtRatioAtTick(params.tickLower),
          TickMath.getSqrtRatioAtTick(params.tickUpper),
          params.liquidityDelta
    );
}
```

Reference

Impact

When the current price is equal to the left boundary of the range, the uniswap pool will request both `token0` and `token1`, but arrakis will only request from the user `token0` so the pool will lose some `token1` if it has enough to cover it.

Code Snippet

https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-core/contracts/libraries/Underlying.sol#LL311-L318

Tool used

Manual Review

Recommendation

Change from:

```
// @audit-issue Change <= to <.
if (sqrtRatioX96 <= sqrtRatioAX96) {
     amount0 = SafeCast.toUint256(
        SqrtPriceMath.getAmount0Delta(
           sqrtRatioAX96,
           sqrtRatioBX96,
           liquidity
         )
     );
}
```

to:

```
if (sqrtRatioX96 < sqrtRatioAX96) {
     amount0 = SafeCast.toUint256(
        SqrtPriceMath.getAmount0Delta(
           sqrtRatioAX96,
           sqrtRatioBX96,
           liquidity
         )
     );
}
```

Discussion

Gevarist

We consider this issue as a medium severity issue, because the cost of an attacker to benefit from this vulnerability and steal some token1 as expensive. The attacker needs to provide the equivalent amount of token0.

ctf-sec

The calculated amount to supply the token mismatched the actually supplied amount depends on the ticker range and the over-charged part from user fund is lost, recommend maintaining the high severity.

0xRobocop

Escalate

Apart from #142

The issues #65 #118 #149 and #269 are not duplicates of this issue.

Regardless on their own validity, the mitigation is different, pointing to different root causes.

ctf-sec

Emm You may need to edit from "Escalate for 10 USDC" to "Escalate" in the comments.

sherlock-admin

> Escalate
>
> Apart from #142
>
> The issues #65 #118 #149 and #269 are not duplicates of this issue.
>
> Regardless on their own validity, the mitigation is different, pointing to different root causes.

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.

P12473

> The calculated amount to supply the token mismatched the actually supplied amount depends on the ticker range and the over-charged part from user fund is lost, recommend maintaining the high severity.

Should be medium considering high is defined as "This vulnerability would result in a material loss of funds, and the cost of the attack is low (relative to the amount of funds lost)."

Jeiwan

Escalate

This is a medium severity issue. The miscalculation won't cause a loss of funds since the wrong amount will be rejected by the underlying Uniswap pool. Also, the protocol has robust slippage protections, which won't allow users to deposit more tokens than required.

All in all, the issue doesn't demonstrate an attack scenario, or a scenario when users lose significant funds, thus it cannot have a high severity.

sherlock-admin

> Escalate
>
> This is a medium severity issue. The miscalculation won't cause a loss of funds since the wrong amount will be rejected by the underlying Uniswap pool. Also, the protocol has robust slippage protections, which won't allow users to deposit more tokens than required.
>
> All in all, the issue doesn't demonstrate an attack scenario, or a scenario when users lose significant funds, thus it cannot have a high severity.

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.

JeffCX

After reading the escalation, seems like the severity is medium

> When the current price is equal to the left boundary of the range, the uniswap pool will request both token0 and token1, but arrakis will only

> request from the user token0 so the pool will lose some token1 if it has enough to cover it.

this does cause protocol to lose fund because the protocol's fund instead of user's fund is used to add liquidity.

but mainly because the attacker not able to leverage this bug to steal fund and the pre-condition:

> current price is equal to the left boundary of the range

Recommend changing severity from high to medium

I will adress 0xRobocoop's escalation seperately

Gevarist

we agree that this issue is medium.

hrishibhat

Result: Medium Has duplicates Accepting both escalations. This is a valid medium issue. and only #142 is the duplicate

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xRobocop: accepted
- Jeiwan: accepted

Gevarist

We replace it with strict inequality here. PR: https://github.com/ArrakisFinance/v2-core/pull/159

IAm0x52

Fix looks good. Now uses < instead of <=

Issue M-2: Lack of rebalance rate limiting allow operators to drain vaults

Source: https://github.com/sherlock-audit/2023-06-arrakis-judging/issues/25

Found by

cergyk, immeas, n33k, p12473

Summary

Operators of Arrakis vaults are constrained by the checks defined in `SimpleManager.sol`, these checks prevent them from causing too much of a fund loss on a single rebalance call (check univ3 pool price deviation, enforce minimum swap slippage parameters).

However since there is no rate limiting for an operator to call rebalance on SimpleManager, an operator can simply drain the vault by applying the accepted slippage a hundred times in one transaction.

Vulnerability Detail

There are mostly two safety measures for preventing an operator to extract vault funds when calling rebalance:

- Check pool price deviation for mints: By checking that a pool price is close to a price given by a chainlink feed, operator is prevented from adding liquidity to a pool at a really unfavorable price, and backrun later to extract vault funds
- Check slippage parameters for swap (`_checkMinReturn`): By checking that a minimum amount of tokens is returned to the vault after the swap, it is preventing the operator to swap tokens at a too unfavorable price. Min amount out enforced in ArrakisV2:

https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-core/contracts/ArrakisV2.sol#L341-L363

As stated by the sponsor, in a public trustless setup, these slippage parameters should be restricted to ~1%.

However since an operator is not rate limited for the number of calls she can do on `SimpleManager.rebalance`, she can simply call it multiple times in a very short timespan, extract an arbitrarily large share of vault funds.

Impact

Vault funds can be drained by a malicious operator.

Code Snippet

Tool used

Manual Review

Recommendation

Enforce a rate limiting policy to the number of calls which can be made to SimpleManager's rebalance, or even better enforce a rate limit on loss of funds which can occur due to rebalances (by evaluating `totalUnderlyingWithFees` before and after the execution for example).

Discussion

ctf-sec

> Operators are "semi trusted" only to be awake and adhere to the expected vault rebalancing strategy. Thus a malicious operator on the SimpleManager.sol should not be able to do anything worse than "grief" - they MAY not execute rebalances or MAY not execute the expected strategy. However the rebalances that are executed MUST NOT be exploitable by frontrun or sandwich.

Based the info from the contest readme doc, upgrading the severity to high

IAm0x52

Escalate

Should be medium not high. This attacks has multiple conditions to work. First of all the operator must become compromised. Second the profitability of a sandwich attack is highly dependent on the composition of the underlying pools and the slippage setting on the manager. If the vault is primarily trading in a 0.3% (0.6% combined buy and sell) fee pool then a slippage cap of 0.5% makes it impossible for this type of attack to work. Additionally depending on the depth of liquidity for nearby relevant buckets the cost to move the pool may outweigh any gains making it unprofitable.

sherlock-admin

> Escalate
>
> Should be medium not high. This attacks has multiple conditions to work. First of all the operator must become compromised. Second the profitability of a sandwich attack is highly dependent on the composition of the underlying pools and the slippage setting on the manager. If the vault is primarily trading in a 0.3% (0.6% combined buy and sell) fee pool then a slippage cap of 0.5% makes it impossible for this type of attack to

> work. Additionally depending on the depth of liquidity for nearby relevant buckets the cost to move the pool may outweigh any gains making it unprofitable.

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.

Jeiwan

Escalate

This is not a valid issue, the behaviour of operators is expected. As per the contest details (https://audits.sherlock.xyz/contests/86):

> Q: Are there any off-chain mechanisms or off-chain procedures for the protocol (keeper bots, input validation expectations, etc)? Yes the SimpleManager.operators are keeper bots running the "off chain logic" that defines the "market making strategy." While it is generally expected that these keeper bots are not malicious, are mostly available/awake, and are running off chain logic to create the rebalance payloads on the expected "market making strategy," even in the absence of this where a malicious keeper bot passes arbitrary payloads to rebalance, there should be no way to extract value from these rebalances directly beyond the acceptable slippage tolerance defined in SimpleManager.

As the finding describes, operators cannot extract value from rebalance beyond the slippage tolerance. Even though if they can rebalance multiple times, they cannot get more value than is set by the slippage tolerance. This is expected behaviour and it was explicitly mentioned by the sponsor.

I also disagree with this comment: https://github.com/sherlock-audit/2023-06-arrakis-judging/issues/25#issuecomment-1634990625 The finding doesn't mention a front run or a sandwiching attack, thus the statement is not applicable. The upgrade to high severity is not justified. The demonstrated scenario in the finding is pretty much in line with the definition of operators: they are semi trusted actors that execute a strategy and cannot manipulate the price more than the specified by the slippage tolerance. The finding doesn't prove the slippage tolerance check can be avoided.

sherlock-admin

> Escalate
>
> This is not a valid issue, the behaviour of operators is expected. As per the contest details (https://audits.sherlock.xyz/contests/86):
>
> > Q: Are there any off-chain mechanisms or off-chain procedures

> for the protocol (keeper bots, input validation expectations, etc)? Yes the SimpleManager.operators are keeper bots running the "off chain logic" that defines the "market making strategy." While it is generally expected that these keeper bots are not malicious, are mostly available/awake, and are running off chain logic to create the rebalance payloads on the expected "market making strategy," even in the absence of this where a malicious keeper bot passes arbitrary payloads to rebalance, there should be no way to extract value from these rebalances directly beyond the acceptable slippage tolerance defined in SimpleManager.

> As the finding describes, operators cannot extract value from rebalance beyond the slippage tolerance. Even though if they can rebalance multiple times, they cannot get more value than is set by the slippage tolerance. This is expected behaviour and it was explicitly mentioned by the sponsor.

> I also disagree with this comment: https://github.com/sherlock-audit/2023-06-arrakis-judging/issues/25#issuecomment-1634990625 The finding doesn't mention a front run or a sandwiching attack, thus the statement is not applicable. The upgrade to high severity is not justified. The demonstrated scenario in the finding is pretty much in line with the definition of operators: they are semi trusted actors that execute a strategy and cannot manipulate the price more than the specified by the slippage tolerance. The finding doesn't prove the slippage tolerance check can be avoided.

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.

IAm0x52

Agreed with @Jeiwan that this should be invalid since the operator never violates the slippage tolerance

syjcnss

Not agree on the two escalations.

First. In the README of the audit it says 'A wide variety' of tokens will be supported. And there's no limit on the fee tier of supported pairs. So there will be vaults that are vulnerable to this attack.

Second. This bug is not about violating the the slippage tolerance. But is about leveraging the slippage tolerance to profit. This attack will definitely cause a

massive loss to this protocol and I don't think this is the expected behaviour.

Jeiwan

> Second. This bug is not about violating the the slippage tolerance.

Exactly. The contest description was very clear about what attack vectors involving operators are deemed valid. The slippage tolerance is not violated, thus this is not a valid attack.

> But is about leveraging the slippage tolerance to profit.

This point is also addressed in the contest details:

> Operators are "semi trusted" only to be awake and adhere to the expected vault rebalancing strategy. While it is generally expected that these keeper bots are not malicious

Operators are trusted to not be malicious and to execute the rebalancing strategy as its programmed. The finding doesn't demonstrate how an operator could go beyond what it's trusted to do. There's no abuse of the strategy. The slippage tolerance works as expected. Operators are expected to do multiple rebalances. Price manipulations are protected by the slippage tolerance–the amount of loss that's available by the tolerance setting is expected and acceptable by the protocol.

Jeiwan

I checked the duplicate reports of this issue, some of them are more detailed about how this can be exploited. However, all those exploit scenarios were addressed in the contest description:

> where a malicious keeper bot passes arbitrary payloads to rebalance, there should be no way to extract value from these rebalances directly beyond the acceptable slippage tolerance defined in SimpleManager.

So, we have two exploiting techniques for the attack to be valid:

1. Price manipulation. This is already addressed by the code: the slippage tolerance check protects against that, while allowing some expected loses.
2. Payload manipulation. The finding doesn't prove that payload manipulation can break the slippage tolerance check.

Even after checking the other reports, I still believe this finding is out of scope. It describes expected behavior of operators.

JeffCX

Ok. After reading the escalation, for now I agree with the senior watson's escalation, can change the severity from high to medium

So

> they are semi trusted actors that execute a strategy and cannot manipulate the price more than the specified by the slippage tolerance.

I would interpret this as "operator" are not trusted.

Second, I agree within single transaction, the slippage protection is not bypassed

but suppose we set the slippage to 1%

If operator keep doing rebalance:

- first rebalance, 99% of the token we received
- second rebalance: 98.01% of the token we received
- third rebalance: 97.02% of the token we received

...

but the attack may not be profitable for attacker :)

I would recommend downgrade the severity from high to medium (report still valid)

unless there are more proof for this statement:

> This attack will definitely cause a massive loss to this protocol and I don't think this is the expected behaviour.

syjcnss

Agree with the lead judge and senior watson's escalation.

Jeiwan

My view still holds true. Since the slippage tolerance check is not avoided this is not a valid attack scenario. It was expected by the sponsor and the contest description was clear that operators are "semi trusted" and only breaking the slippage tolerance check can be considered an attack.

@hrishibhat

SergeKireev

I would like to address the escalations:

@Jeiwan: The contest page also says:

```
Thus a malicious operator on the SimpleManager.sol should not be able to do
↪  anything worse than "grief" - they MAY not execute rebalances or MAY not
↪  execute the expected strategy. However the rebalances that are executed MUST
↪  NOT be exploitable by frontrun or sandwich.
```

Here the impact is that a malicious operator can drain a big chunk of TVL in the protocol in one single transaction.

@IAm0x52 These are very interesting points:

> Should be medium not high. This attacks has multiple conditions to work. First of all the operator must become compromised. Second the profitability of a sandwich attack is highly dependent on the composition of the underlying pools and the slippage setting on the manager. If the vault is primarily trading in a 0.3% (0.6% combined buy and sell) fee pool then a slippage cap of 0.5% makes it impossible for this type of attack to work. Additionally depending on the depth of liquidity for nearby relevant buckets the cost to move the pool may outweigh any gains making it unprofitable.

I agree that the operator has to be compromised as a condition.

Since the operator can use the router payload to swap on any pool, only limited by vault.maxSlippage (for which a value of 1% should be considered according to sponsor), the malicious operator can create a bogus pool with a poison token which could just be used to extract the tokens sent by Arrakis with a fee of 0.01%.

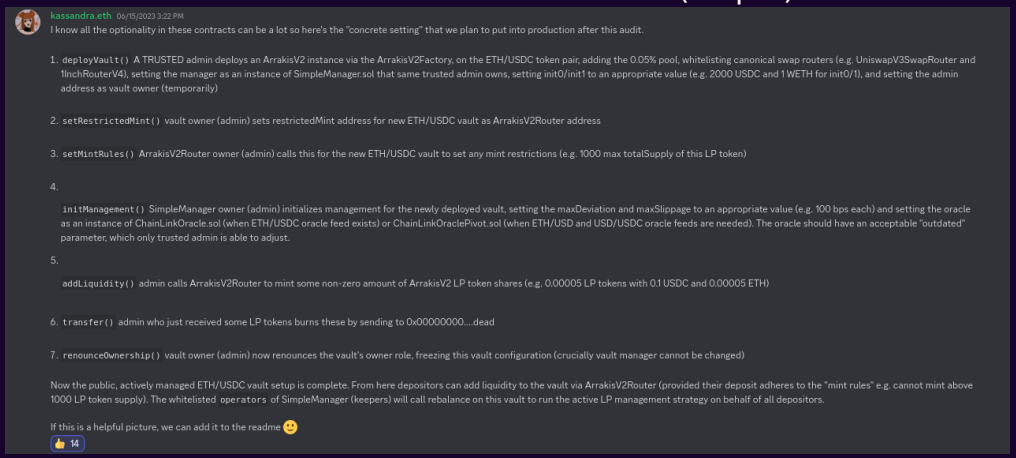
Which means 0.99% extracted on each call. Since multiple calls can be done to `rebalance` in a single transaction, the malicious operator can drain an arbitrary amount of funds before the owner can even try to react and change it

ctf-sec

emm it is not possible to inject own's opinion into another haha, yeah just recommend leave factual based comments and wait for sherlock to resolve the escalation, recommend changing severity to medium :)

SergeKireev

Found again the message posted by sponsor in the public discord where slippage and deviation would be considered to be 1% (step 4):



syjcnss

> but suppose we set the slippage to 1%

> If operator keep doing rebalance:
>
> first rebalance, 99% of the token we received second rebalance: 98.01% of the token we received third rebalance: 97.02% of the token we received ...

So, for 100 times of rebalance we will only get 0.99**100 = 36.60% of the token left in the vault? For 300, it's very close to empty. 0.99**300 = 4.90%

If I'm right on the math above, then a compromised operator may not be needed. 100 times of rebalance is very much likely achievable in a short period like a few months in a fluctuant market. A compromised operator will make a massive loss quickly.

0ximmeas

As i stated in my dup of this, from the docs:

> Operators are "semi trusted" only to be awake and adhere to the expected vault rebalancing strategy. Thus a malicious operator on the SimpleManager.sol should not be able to do anything worse than "grief"

This is doing something worse than grief, the sponsor just didn't think of this specific attack scenario. An operator can steal a significant amount of the vaults funds in a single transaction. Thus high.

Gevarist

If the operator is malicious, owner has the option to replace it.

SergeKireev

> If the operator is malicious, owner has the option to replace it.

Without some kind of rate limiting, the draining can happen in one transaction, no way for the owner to mitigate it

Gevarist

How much iteration can the malicious operator do in 1 block? Less than thirty rebalance in one block for sure. The gas cost of such attack is considerable. And operators are semi-trusted, we think this issue is a medium.

SergeKireev

If we assume that gas consumed is approx 1 uniswapv3 swap (the operator can choose to do only the swap step during rebalance with 1% slippage), 30 is a good estimate of the max number of rebalances possible during 1 block.

Which caps the theft of funds during one block at (1-0.99**30) ~= 26% of the funds.

I agree @Gevarist that this attack is a bit less practical than what described in previous discussion

hrishibhat

Result: Medium Has duplicates Considering this a valid medium based on the above final comments.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- IAm0x52: accepted
- Jeiwan: rejected

Gevarist

We add a cooldown period during which rebalance is not possible by operator here.
PR : https://github.com/ArrakisFinance/v2-manager-templates/pull/26

IAm0x52

Fix looks good. Added a cooldown period to rebalances

Issue M-3: Min deposit protection during rebalancing can be bypassed if multiple fee tiers

Source: https://github.com/sherlock-audit/2023-06-arrakis-judging/issues/28

Found by

cergyk

Summary

Arrakis vault implements a slippage protection during rebalancing, but it does not protect when multiple feeTiers are used.

Vulnerability Detail

The slippage protection used here: https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-core/contracts/ArrakisV2.sol#L408-L409

can be used to check that liquidity is added around a given price (acts like a slippage protection), when used on one pool.

However when aggregated on multiple pools as it is done here (over same tokens but multiple fee tiers), it does not protect the user, as pools can be imbalanced in different directions, and funds can be provided in the same proportion but provide liquidity at a worse price,

Example:

Arrakis vault handles the price range 1200-2000 on WETH-USDC in fee tiers [0.3%, 0.5%], price of WETH sits at 1600.

Alice sees that Bob the operator tries to rebalance the pool and thus provides liquidity on both ranges

Alice front runs Bob transaction, driving the price of WETH-USDC on fee tier 0.3% to 1200, and on fee tier 0.5% to 2000.

Bob executes his operation, and provides bigger amounts of WETH and USDC to provide same liquidity (worse prices), and so the slippage checks out.

Alice back runs the operation and makes a nice profit.

Please note that in the setup using `SimpleManager`, price deviation is checked on every minting pool: https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-manager-templates/contracts/SimpleManager.sol#L189-L194

So the sandwiching is limited by the deviation parameter (1%). This issue still holds, since maybe Bob wanted to enforce a stricter deviation for his rebalance (0.5%)

using this check, and fails to do so.

Impact

Bob makes an unfortunate rebalance to the profit of a sandwich.

Code Snippet

Tool used

Manual Review

Recommendation

Check deposited amounts in aggregate but grouped by fee tiers.

Discussion

kassandraoftroy

For me this issue is a valid medium and not a duplicate of #164 since it is about rebalance() not addLiquidity() (so does not have the proper checks to sniff out manipulation when adding liquidity on multiple fee tiers simultaneously).

Gevarist

For this issue we have a fix here, we are letting the operator to fix a custom max slippage value (lower than the vault max slippage defines by owner during initialization of the management).

IAm0x52

Fix looks good. Operator can now specify a tighter slippage value

Issue M-4: outdated variable is not effective to check price feed timeliness

Source: https://github.com/sherlock-audit/2023-06-arrakis-judging/issues/83

Found by

0x007, 0x52, 0xg0, BenRai, Jeiwan, ast3ros, cergyk, elephant_coral, rvierdiiev, vnavascues

Summary

In ChainlinkOraclePivot, it uses one `outdated` variable to check if the two price feeds are outdated. However, this is not effective because the price feeds have different update frequencies.

Vulnerability Detail

Let's have an example:

In Polygon mainnet, ChainlinkOraclePivot uses two Chainlink price feeds: MATIC/ETH and ETH/USD.

The setup can be the same in this test case: https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-manager-templates/test/foundry/ChainLinkOraclePivotWrapper.t.sol#L49-L63

We can see that

- priceFeedA: MATIC/ETH price feed has a heartbeat of 86400s (https://data.chain.link/polygon/mainnet/crypto-eth/matic-eth).
- priceFeedB: ETH/USD price feed has a heartbeat of 27s (https://data.chain.link/polygon/mainnet/crypto-usd/eth-usd).

In function `_getLatestRoundData`, both price feeds use the same `outdated` variable.

- If we set the `outdated` variable to 27s, the priceFeedA will revert most of the time since it is too short for the 86400s heartbeat.
- If we set the `outdated` variable to 86400s, the priceFeedB can have a very outdated value without revert.

```
try priceFeedA.latestRoundData() returns (
    uint80,
    int256 price,
    uint256,
    uint256 updatedAt,
    uint80
```

```
) {
    require(
        block.timestamp - updatedAt <= outdated, // solhint-disable-line
↪   not-rely-on-time
        "ChainLinkOracle: priceFeedA outdated."
    );

    priceA = SafeCast.toUint256(price);
} catch {
    revert("ChainLinkOracle: price feed A call failed.");
}

try priceFeedB.latestRoundData() returns (
    uint80,
    int256 price,
    uint256,
    uint256 updatedAt,
    uint80
) {
    require(
        block.timestamp - updatedAt <= outdated, // solhint-disable-line
↪   not-rely-on-time
        "ChainLinkOracle: priceFeedB outdated."
    );

    priceB = SafeCast.toUint256(price);
} catch {
    revert("ChainLinkOracle: price feed B call failed.");
}
```

https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-manager-templates/contracts/oracles/ChainLinkOraclePivot.sol#L239-L271

Impact

The `outdated` variable is not effective to check the timeliness of prices. It can allow stale prices in one price feed or always revert in another price feed.

Code Snippet

https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-manager-templates/contracts/oracles/ChainLinkOraclePivot.sol#L31
https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-manager-templates/contracts/oracles/ChainLinkOraclePivot.sol#L239-L271

Tool used

Manual Review

Recommendation

Having two `outdated` values for each price feed A and B.

Discussion

Gevarist

oracle price check will always revert for few specific feeds. Should not result in fund loss, we are not considering the issue as medium level.

ctf-sec

I think this regular revert impact the rebalance...

#249 describes the issue well as well

recommend maintaining severity level

Gevarist

Yes that can impact rebalance but should not result in fund loss.

Gevarist

You can find the fix here on v2-manager-template repository, we now have two "outdated" variables to check individually if price feeds are outdated.

IAm0x52

Fix looks good. Oracle now has two separate outdated variables to accommodate feeds with different heartbeats

Issue M-5: Update to `managerFeeBPS` applied to pending tokens yet to be claimed

Source: https://github.com/sherlock-audit/2023-06-arrakis-judging/issues/198

Found by

0xDjango, Jeiwan, ast3ros, dipp, immeas, rugpull_detector, rvierdiiev

Summary

A manager (malicious or not) can update the `managerFeeBPS` by calling `ArrakisV2.setManagerFeeBPS()`. The newly-updated `managerFeeBPS` will be retroactively applied to the pending fees yet to be claimed by the `ArrakisV2` contract.

Vulnerability Detail

Whenever UniV3 fees are collected (via `burn()` or `rebalance()`), the manager fees are applied to the received pending tokens.

```
function _applyFees(uint256 fee0_, uint256 fee1_) internal {
    uint16 mManagerFeeBPS = managerFeeBPS;
    managerBalance0 += (fee0_ * mManagerFeeBPS) / hundredPercent;
    managerBalance1 += (fee1_ * mManagerFeeBPS) / hundredPercent;
}
```

Since the manager can update the `managerFeeBPS` whenever, this calculation can be altered to take up to 100% of the pending fees in favor of the manager.

```
function setManagerFeeBPS(uint16 managerFeeBPS_) external onlyManager {
    require(managerFeeBPS_ <= 10000, "MFO");
    managerFeeBPS = managerFeeBPS_;
    emit LogSetManagerFeeBPS(managerFeeBPS_);
}
```

Impact

- Manager's ability to intentionally or accidently steal pending fees owed to stakers

Code Snippet

https://github.com/sherlock-audit/2023-06-arrakis/blob/main/v2-core/contracts/a

bstract/ArrakisV2Storage.sol#L218-L222

Tool used

Manual Review

Recommendation

Fees should be collected at the start of execution within the `setManagerFeeBPS()` function. This effectively checkpoints the fees properly, prior to updating the `managerFeeBPS` variable.

Discussion

Gevarist

We provide a fix on the v2-core repository, we add an internal function _collectFeesOnPools that will be called when the manager will set managerFeeBPS

IAm0x52

Fix looks good. Fees are now updated and collected for manager anytime there is an update to manager or fee BPS