SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	KyberSwap
Prepared by:	Sherlock
Lead Security Expert:	0x52
Dates Audited:	August 21 - September 6, 2023
Prepared on:	October 26, 2023

Introduction

Our flagship product KyberSwap is the crosschain DEX & aggregator on 15 chains enabling users to trade smart and maximize yields.

Scope

Repository: KyberNetwork/ks-elastic-sc

Branch: audit_sherlock

Commit: 4ab08c0a60f74809f731bdd333076e32d05f1d17

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
2	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0x52	MohammedRizwan

Issue M-1: UUPSUpgradeable vulnerability in OpenZeppelin Contracts

Source: https://github.com/sherlock-audit/2023-07-kyber-swap-judging/issues/25

Found by

0x52, MohammedRizwan UUPSUpgradeable vulnerability in OpenZeppelin Contracts

Vulnerability Detail

Openzeppelin has found the critical severity bug in UUPSUpgradeable. The kyber-swap contracts has used both openzeppelin contracts as well as openzeppelin upgrabable contracts with version v4.3.1. This is confirmed from package.json.

```
File: ks-elastic-sc/package.json

    "@openzeppelin/contracts": "4.3.1",
    "@openzeppelin/test-helpers": "0.5.6",
    "@openzeppelin/contracts-upgradeable": "4.3.1",
```

The `UUPSUpgradeable` vulnerability has been found in openzeppelin version as follows,

@openzeppelin/contracts : Affected versions >= 4.1.0 < 4.3.2
@openzeppelin/contracts-upgradeable : >= 4.1.0 < 4.3.2

However, openzeppelin has fixed this issue in versions 4.3.2

Openzeppelin bug acceptance and fix: check here

The following contracts has been affected due to this vulnerability

1) PoolOracle.sol
2) TokenPositionDescriptor.sol

Both of these contracts are UUPSUpgradeable and the issue must be fixed.

Impact

Upgradeable contracts using UUPSUpgradeable may be vulnerable to an attack affecting uninitialized implementation contracts.

Code Snippet

https://github.com/sherlock-audit/2023-07-kyber-swap/blob/main/ks-elastic-sc/package.json#L35-L37

https://github.com/sherlock-audit/2023-07-kyber-swap/blob/main/ks-elastic-sc/contracts/oracle/PoolOracle.sol#L19

https://github.com/sherlock-audit/2023-07-kyber-swap/blob/main/ks-elastic-sc/contracts/periphery/TokenPositionDescriptor.sol#L13

Tool used

Manual Review

Recommendation

1) Update the openzeppelin library to latest version.
2) Check this openzeppelin security advisory to initialize the UUPS implementation contracts.
3) Check this openzeppelin UUPS documentation.

Discussion

sherlock-admin

1 comment(s) were left on this issue during the judging contest.

Trumpero commented:

> valid medium about OZ upgradable 4.3.1

manhlx3006

Sponsor Confirmed

- The finding is valid and a known issue (we have initialized all Pool Oracle for deployed contracts and transferred ownership to a multisig). The UUPSUpgradable has a vulnerability with the version we used. If the implementation is not initialized, someone can initialize and try to destroy the implementation.
- For deployed contracts: We have initialized all implementation of deployed contracts and currently our multisig is the owner of all, thus, for the current deployed contracts, the issue won't happen.
- For the future deployment: we will move to the new OZ version 4.3.2.

nevillehuang

Escalate

This should be low severity and invalid according to sherlock rules. This comes down to admin deployment errors where admin deploying PoolOracle contract fails to initialize contracts, which kyberswap admins have already done and in fact are aware of this issue.

It really only affects future deployments, but given kyberswap are moving towards 4.3.2, and no other contracts are going to be deployed with this version of OZ 4.3.1, this is a non-issue as there is no existing risk to the protocol, unless the admin makes an error and mistakenly deploy new contracts with this versions.

sherlock-admin2

> Escalate
>
> This should be low severity and invalid according to sherlock rules. This comes down to admin deployment errors where admin deploying PoolOracle contract fails to initialize contracts, which kyberswap admins have already done and in fact are aware of this issue.
>
> It really only affects future deployments, but given kyberswap are moving towards 4.3.2, and no other contracts are going to be deployed with this version of OZ 4.3.1, this is a non-issue as there is no existing risk to the protocol, unless the admin makes an error and mistakenly deploy new contracts with this versions.

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.

jkoppel

Escalate

This is out of scope. This is at best a flaw in package.json, which is not in scope. The rules about vulns in libraries were changed *during the judging period* and should not apply to this contest.

Further, Sherlock has in the past ruled it invalid when there is an issue in the code that the protocol team has already fixed by a deployment strategy. https://github.com/sherlock-audit/2023-02-blueberry-judging/issues/146/#issuecomment-1475484578

sherlock-admin2

> Escalate
>
> This is out of scope. This is at best a flaw in package.json, which is not in scope. The rules about vulns in libraries were changed *during the*

> *judging period* and should not apply to this contest.
>
> Further, Sherlock has in the past ruled it invalid when there is an issue in the code that the protocol team has already fixed by a deployment strategy. https://github.com/sherlock-audit/2023-02-blueberry-judging/issues/146/#issuecomment-1475484578

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.

jingyi2811

Escalate

i don't agree with both escalations.

UUPSUpgradeable vulnerability in OpenZeppelin Contracts is a critical vulnerability per the openzeppelin which was found in 2021 and to be noted this was the only second critical issue found till date in openzeppelin library. `If the implementation is not initialized, someone can initialize and try to destroy the implementation` which i agree with the sponsor comment here.

It should be noted here that TokenPositionDescriptor.sol as well as PoolOracle.sol is affected from this issue. `TokenPositionDescriptor.sol` is also in scope contract. As sponsor confirmed the issue with reasoning and judge reasoning for this issue confirmation too, This is absolutely a valid issue. Below is the additioanal context for raised escalations.

Issues found in used library dependencies has been judged as Medium at sherlock in past irrespective of new judging guidlines. Below are few issues for references,

1) Harpie audit, check Issue M-6: Signature malleability not protected agains t---Similar openzeppelin library bug issue judged as Medium, However the library issue was High severity and here it is critical severity.
2) UXD audit, check FullMath library overflow issue----library issue taken from uniswap
3) Bond audit, check Solmate code size issue-----solmate library token code size issue judged as Medium
4) I wouldn't had added this reference but i think it should be as the escalations focuses on rules and past issues. Check this issue. The issue is different but the escalation decision revolves around the used library where the Sherlock team reasoning for escalation rejection was

   > The library is used in the in-scope contract and the error impacts an in-scope contract. This is a valid issue.

To be noted, all these issues were happened before July, 2023 and the new rules made significant judging changes after July, 2023.(This was not intended to add but to respond a escalation, It was necessary to add for reference purpose only)

Per the sherlock rules too, This finding is valid and Sherlock rules states,

> In case the vulnerability exists in a library and an in-scope contract uses it and is affected by this bug this is a valid issue.

Sherlock docs reference here

Contest readme.md also dont have this as known issue even the previous audits done by kyberSwap team does not highlight this critical issue too. I believe this issue is correctly judged by the lead judge and it's good that this issue is being found at sherlock which was missed in previous audits.

I hope, the above explanation is good enough for the validation of this issue.

Thank you!

sherlock-admin2

> Escalate
>
> i don't agree with both escalations.
>
> UUPSUpgradeable vulnerability in OpenZeppelin Contracts is a critical vulnerability per the openzeppelin which was found in 2021 and to be noted this was the only second critical issue found till date in openzeppelin library. `If the implementation is not initialized, someone can initialize and try to destroy the implementation` which i agree with the sponsor comment here.
>
> It should be noted here that TokenPositionDescriptor.sol as well as PoolOracle.sol is affected from this issue. `TokenPositionDescriptor.sol` is also in scope contract. As sponsor confirmed the issue with reasoning and judge reasoning for this issue confirmation too, This is absolutely a valid issue. Below is the additioanal context for raised escalations.
>
> Issues found in used library dependencies has been judged as Medium at sherlock in past irrespective of new judging guidlines. Below are few issues for references,
>
> 1) Harpie audit, check Issue M-6: Signature malleability not protected against---Similar openzeppelin library bug issue judged as Medium, However the library issue was High severity and here it is critical severity.
> 2) UXD audit, check FullMath library overflow issue----library issue taken from uniswap

> 3) Bond audit, check Solmate code size issue-----solmate library token code size issue judged as Medium
>
> 4) I wouldn't had added this reference but i think it should be as the escalations focuses on rules and past issues. Check this issue. The issue is different but the escalation decision revolves around the used library where the Sherlock team reasoning for escalation rejection was
>
> > The library is used in the in-scope contract and the error impacts an in-scope contract. This is a valid issue.
>
> To be noted, all these issues were happened before July, 2023 and the new rules made significant judging changes after July, 2023.(This was not intended to add but to respond a escalation, It was necessary to add for reference purpose only)
>
> Per the sherlock rules too, This finding is valid and Sherlock rules states,
>
> > In case the vulnerability exists in a library and an in-scope contract uses it and is affected by this bug this is a valid issue.
>
> Sherlock docs reference here
>
> Contest readme.md also dont have this as known issue even the previous audits done by kyberSwap team does not highlight this critical issue too. I believe this issue is correctly judged by the lead judge and it's good that this issue is being found at sherlock which was missed in previous audits.
>
> I hope, the above explanation is good enough for the validation of this issue.
>
> Thank you!

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.

jkoppel

@jingyi2811 The part of the Sherlock rules you cite was added just 9 days ago. https://discord.com/channels/812037309376495636/1087792569196486706/1151162792074813592

nevillehuang

> Escalate
>
> i don't agree with both escalations.

UUPSUpgradeable vulnerability in OpenZeppelin Contracts is a critical vulnerability per the openzeppelin which was found in 2021 and to be noted this was the only second critical issue found till date in openzeppelin library. `If the implementation is not initialized, someone can initialize and try to destroy the implementation` which i agree with the sponsor comment here.

It should be noted here that TokenPositionDescriptor.sol as well as PoolOracle.sol is affected from this issue. `TokenPositionDescriptor.sol` is also in scope contract. As sponsor confirmed the issue with reasoning and judge reasoning for this issue confirmation too, This is absolutely a valid issue. Below is the additioanal context for raised escalations.

Issues found in used library dependencies has been judged as Medium at sherlock in past irrespective of new judging guidlines. Below are few issues for references,

1. Harpie audit, check Issue M-6: Signature malleability not protected against---Similar openzeppelin library bug issue judged as Medium, However the library issue was High severity and here it is critical severity.
2. UXD audit, check FullMath library overflow issue----library issue taken from uniswap
3. Bond audit, check Solmate code size issue-----solmate library token code size issue judged as Medium
4. I wouldn't had added this reference but i think it should be as the escalations focuses on rules and past issues. Check this issue. The issue is different but the escalation decision revolves around the used library where the Sherlock team reasoning for escalation rejection was

   > The library is used in the in-scope contract and the error impacts an in-scope contract. This is a valid issue.

To be noted, all these issues were happened before July, 2023 and the new rules made significant judging changes after July, 2023.(This was not intended to add but to respond a escalation, It was necessary to add for reference purpose only)

Per the sherlock rules too, This finding is valid and Sherlock rules states,

> In case the vulnerability exists in a library and an in-scope contract uses it and is affected by this bug this is a valid issue.

Sherlock docs reference here

Contest readme.md also dont have this as known issue even the previous audits done by kyberSwap team does not highlight this critical issue too.

> I believe this issue is correctly judged by the lead judge and it's good that this issue is being found at sherlock which was missed in previous audits.
>
> I hope, the above explanation is good enough for the validation of this issue.
>
> Thank you!

Sure it is valid in sponsors eyes and yes it is not made a known issue. The point here is it falls under admin error, and thus is invalid according to sherlocks rule here. There is no existing risks, and if there is, it revolves around future deployment errors by the admin, where historically admin errors has not been accepted as valid issues.

On top of that, all the issues u quoted involves user facing functions, whereas this issue revolves admin facing functions. This is in addition to the fact that there will never be another `initialize()` function with OZ 4.3.1 for a user to call for either contracts. Will let @hrishibhat decide on this one.

jingyi2811

If a suggestion made in the in-scope file fix the problem, I would say this is a valid issue. Kindly read 0x52 duplicate issue of this.

0xRizwan

> > Escalate i don't agree with both escalations. UUPSUpgradeable vulnerability in OpenZeppelin Contracts is a critical vulnerability per the openzeppelin which was found in 2021 and to be noted this was the only second critical issue found till date in openzeppelin library. `If the implementation is not initialized, someone can initialize and try to destroy the implementation` which i agree with the sponsor comment here. It should be noted here that TokenPositionDescriptor.sol as well as PoolOracle.sol is affected from this issue. `TokenPositionDescriptor.sol` is also in scope contract. As sponsor confirmed the issue with reasoning and judge reasoning for this issue confirmation too, This is absolutely a valid issue. Below is the additioanal context for raised escalations. Issues found in used library dependencies has been judged as Medium at sherlock in past irrespective of new judging guidlines. Below are few issues for references,
> >
> > 1. Harpie audit, check Issue M-6: Signature malleability not protected against---Similar openzeppelin library bug issue judged as Medium, However the library issue was High severity and here it is critical severity.
> > 2. UXD audit, check FullMath library overflow issue----library issue taken from uniswap

3. Bond audit, check Solmate code size issue-----solmate library token code size issue judged as Medium
4. I wouldn't had added this reference but i think it should be as the escalations focuses on rules and past issues. Check this issue. The issue is different but the escalation decision revolves around the used library where the Sherlock team reasoning for escalation rejection was

   > The library is used in the in-scope contract and the error impacts an in-scope contract. This is a valid issue.

To be noted, all these issues were happened before July, 2023 and the new rules made significant judging changes after July, 2023.(This was not intended to add but to respond a escalation, It was necessary to add for reference purpose only) Per the sherlock rules too, This finding is valid and Sherlock rules states,

> In case the vulnerability exists in a library and an in-scope contract uses it and is affected by this bug this is a valid issue.

Sherlock docs reference here Contest readme.md also dont have this as known issue even the previous audits done by kyberSwap team does not highlight this critical issue too. I believe this issue is correctly judged by the lead judge and it's good that this issue is being found at sherlock which was missed in previous audits. I hope, the above explanation is good enough for the validation of this issue. Thank you!

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.

I agree with @jingyi2811 escalation and the issue is correcty judged and it is valid.

@jkoppel Per the escalation, it is clearly explained the issues judement of such libraries issue have been judged as Medium in past irrespective of new juding guidlines. quoting the link shared by you here. sherlock team informs about the documentation updates. This announcement by sherlock team does not mention that the new rules wont be applicable to the contest happened before the rules announcement. Its a documentation update which ofcourse help in judging the contest. Please do check the point 4 in @jingyi2811 escalation.

@nevillehuang Thanks for agreeing on some points. Please note the issue arises from the openzeppelin library. Having a critical issue in dependency library is not an admin fault. This can not be considered as an admin error since the admin who is nothing but protocol team has been made aware of this critical issue via sherlock audit reports. If i am correct, the contracts were being audited two times in past and this issue was not raised in any audit report. The sponsor is confirming the issue with the reasoning and the lead judge is also confirming it. I believe the issue is valid. However, Let it be decided by sherlock team and lead judge.

jkoppel

@mohammedrizwann123 Can you give an explanation for why this issue should be judged differently from the Blueberry issue I cited?

Trumpero

I believe this issue should be considered a valid medium. It isn't obligatory for the admin to initialize the PoolOracle contract after deploying it if the correct version of OZ was utilized. It's a fact that KyberSwap didn't initialize their PoolOracle contracts until this bug was spotted recently before the contest, and they will move to OZ 4.3.2 for the future deployments. However, the protocol team didn't identify this issue as a known issue in the contest docs, so this issue should be valid according to Sherlock rules.

Additionally, this issue is truly in-scope since the PoolOracle contract inherits the UUPSUpgradeable contract version 4.3.1; it's not similar to using a library. The entire vulnerability exists in the PoolOracle contract, which can be exploited by an attacker.

jkoppel

> It's a fact that KyberSwap didn't initialize their PoolOracle contracts until this bug was spotted recently before the contest

Link to transaction or discussion please?

I am still seeing nothing to differentiate this from the Blueberry issue I cited.

nevillehuang

This finding was reported by the OZ team in september of 2021 here:

https://forum.openzeppelin.com/t/security-advisory-initialize-uups-implementation-contracts/15301

Pretty sure its not “recent”, unless kyberswap intended to mistakenly leave the bug there for 2 years

hrishibhat

To address the @jkoppel's comments on blueberry the context of that issue does not apply here: That issue involved a Mockup contract and Sponsor had an unusual implementation mechanism that was already in place. That issue was discussed extensively with the Lead Watson and resolved. On the blueberry decision, there are strong arguments for both sides. But the factors of that decision do not apply here given the Mockup library.

As pointed out by the above escalation from @jingyi2811 historically using vulnerable external libraries is a valid issue. Especially for the external libraries that have a known bug in them. And the update of the library-related rule in the rulebook will not impact the decision here as they have always been rewarded. Also the update was intended to address cases like these. Will add more clarity on the same.

This is not an admin error. To simplify an admin error is using an incorrect parameter that causes loss because admin is expected to input the right parameter that is supposed to behave as expected. In this case, admin expects the the external library to work correctly.

Hence agree with the Lead Judge and escalations that consider this a valid issue.

hrishibhat

Result: Medium Has duplicates Considering this issue a valid medium based on the above comment.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- jingyi2811: accepted
- nevillehuang: rejected
- jkoppel: rejected

manhlx3006

For current deployments, we have initialized all contracts with a multisig as the owner. PR to update dependencies for future deployments: https://github.com/KyberNetwork/ks-elastic-sc/pull/21

IAm0x52

Fix looks good. Update to OZ library only changes UUPS and nothing else

Issue M-2: Router.sol is vulnerable to address collission

Source: https://github.com/sherlock-audit/2023-07-kyber-swap-judging/issues/90

Found by

0x52

Router.sol never verifies that the callback msg.sender is actually a deployed pool. This allows for a provable address collision that can be used to drain all allowances to the router.

Vulnerability Detail

The pool address check in the the callback function isn't strict enough and can suffer issues with collision. Due to the truncated nature of the create2 opcode the collision resistance is already impaired to 2ˆ160 as that is total number of possible hashes after truncation. Obviously if you are searching for a single hash, this is (basically) impossible. The issue here is that one does not need to search for a single address as the router never verifies that the pool actually exists. This is the crux of the problem, but first let's do a little math as to why this is a problem.

A primer on hash collision probability can be found here (https://kevingal.com/blog/collisions.html). The odds of a collision are dependant on both the amount of addresses to search against as well as the number of brute force attempts to match them. From the article linked we can see that the estimated odds of collision are:

$$1 - e^{\frac{-k(k-1)}{2N}}$$

Where k is the number of hash values and N is the number of possible hashes

For very large numbers we can further approximate the exponent to:

$$\frac{-k^2}{2N}$$

This exponent is now trivial to solve for an approximate attack value which is:

$$k = \sqrt{2N}$$

In our scenario N is 2ˆ160 (our truncated keccak256) which means that our approximate attack value is 2ˆ80 since we need to generate two sets of hashes. The first set is to generate 2ˆ80 public addresses and the second is to generate pool address from variations in the pool specifics(token0, token1, fee). Here we reach a final attack value of 2ˆ81 hashes. Using the provided calculator we see that

2ˆ81 hashes has an approximate 86.4% chance of a collision. Increase that number to 2ˆ82 and the odds of collision becomes 99.96%. In this case, a collision between addresses means breaking this check and draining the allowances of all users to a specific token. This is because the EOA address will collide with the supposed pool address allowing it to bypass the msg.sender check. Now we considered the specific of the contract.

Router.sol#L47-L51

```
require(
  msg.sender == address(_getPool(tokenIn, tokenOut, fee)),
  'Router: invalid callback sender'
);
```

The above snippet from the swapCallback function is used to verify that msg.sender is the address of the pool.

Router.sol#L224-L231

```
function _getPool(
    address tokenA,
    address tokenB,
    uint24 fee
) private view returns (IPool) {
    return IPool(PoolAddress.computeAddress(factory, tokenA, tokenB, fee,
     ↪  poolInitHash));
}
```

We see that these lines never check with the factory that the pool exists or any of the inputs are valid in any way. token0 can be constant and we can achieve the variation in the hash by changing token1. The attacker could use token0 = WETH and vary token1. This would allow them to steal all allowances of WETH. Since allowances are forever until revoked, this could put hundreds of millions of dollars at risk.

Although this would require a large amount of compute it is already possible to break with current computing. Given the enormity of the value potentially at stake it would be a lucrative attack to anyone who could fund it. In less than a decade this would likely be a fairly easily attained amount of compute, nearly guaranteeing this attack.

Impact

Address collision can cause all allowances to be drained

Code Snippet

Router.sol#L39-L70

Tool used

Manual Review

Recommendation

Verify with the factory that msg.sender is a valid pool

Discussion

sherlock-admin

1 comment(s) were left on this issue during the judging contest.

Trumpero commented:

> invalid, it's impossible to override an existing contract in EVM. From EIP-684

IAm0x52

Escalate

I believe this has been incorrectly excluded.

> invalid, it's impossible to override an existing contract in EVM. From EIP-684

Judges comment here is incorrect. A contract won't exist at this address since the collision would likely occur with at least one ERC20 token that doesn't exist. To reiterate what is in my submission, I think that in the current implementation the collision resistance is not strong enough. The currently available hashing power makes this risk very real (profitability will improve with time) and the losses could be enormous. The router contract is an immutable contract which means that it is almost guaranteed that with time this will be exploited, since it can't be fixed via upgrade. By implementing my suggested mitigation, the chance of exploit becomes zero due to EIP-684.

sherlock-admin2

> Escalate
>
> I believe this has been incorrectly excluded.
>
> > invalid, it's impossible to override an existing contract in EVM. From EIP-684

> Judges comment here is incorrect. A contract won't exist at this address since the collision would likely occur with at least one ERC20 token that doesn't exist. To reiterate what is in my submission, I think that in the current implementation the collision resistance is not strong enough. The currently available hashing power makes this risk very real (profitability will improve with time) and the losses could be enormous. The router contract is an immutable contract which means that it is almost guaranteed that with time this will be exploited, since it can't be fixed via upgrade. By implementing my suggested mitigation, the chance of exploit becomes zero due to EIP-684.

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.

Trumpero

@IAm0x52 I agree that the statements in this issue are correct. However, the likelihood of this issue occurring is nearly impossible, and the cost to attack it will be unreal. These issues can be found in other protocols such as UniswapV3, and I believe those protocols don't need to fix it because the issue is mostly impossible to happen in reality. Therefore, this issue should be considered as a low or non-issue. References: https://github.com/Uniswap/v3-periphery/blob/main/contracts/SwapRouter.sol#L65 https://github.com/Uniswap/v3-periphery/blob/main/contracts/libraries/CallbackValidation.sol#L21

IAm0x52

2ˆ80 is 1.208e24

The current hashrate of Bitcoin is 4.71e20

1.208e24/4.71e20 = 2565.61

This means 2ˆ80 hashes can be generated in 2565 seconds or less than 1 hour. 2ˆ82 (our near guaranteed collision) would be generated in 10260 seconds or ~3 hours. I am aware that Bitcoin uses SHA256 while the hash in this case is Keccak256 but SHA256 is only

manhlx3006

Hi, Even though it is possible for the collision, as you also mentioned, the attacker needs to have a huge computation power, which also means it takes a long time and spends a big amount of expense in order to check IF the attack is possible. In another note, the Router is developed and mainly used by developers which makes the potential profit from the attack is even less (if they wait for a benefit is high enough to start performing the attack, they can not be sure the actual attackable funds after 3-4 weeks, while spending huge amount of funds from their side).

On the other hand, while the collision can be achieved by 2ˆ82 hash, it also requires a database of 2ˆ81 * 20 bytes to store all the EOA addresses (which is quite impossible for current hardware market).

Thus, the likelihood of this attack to happen is really low and unpractical.

jkoppel

0x52 is correct. I checked the calculations. The calculator linked to is for a slightly different scenario than this one and is too low by a factor of 2 for number of hashes needed. Even with that, this attack is certain to be feasible, and quite economical within the next decade. All it takes is a single large transaction going through the router for this attack to become profitable.

@manhlx3006 is correct that a naive collision-finding algorithm would take an astronomical amount of space. But you don't build a secure system by requiring an attacker to use a naive algorithm. You want to use 20 bytes to store every hash computed? Try using a Bloom filter and using 1 bit instead. I'm sure it can be brought down more; I expect that some clever hashing schemes can produce a result of the form "This input has a hash collision with one of these 2ˆ40 other inputs." That reduces the 2ˆ81 compute 2ˆ80 storage problem into a 2ˆ81 compute 2ˆ41 storage problem followed by a second 2ˆ40 compute problem.

Regardless, how much do you want to bank on a solution to the storage problem not existing? The hashing problem is believed to be computationally irreducible; the storage problem is not.

IAm0x52

@jkoppel hit the nail on the head with his explanation with respect to the storage requirement.

Additionally you are deploying an immutable router contract. Approval are permanent unless actively revoked. Very easy to accrue tens of millions of dollars worth of vulnerable approvals even with a low volume protocol. As an example, the vulnerable sushiswap router accumulated multiple millions worth in a few days. Simply being an approval to an immutable contract massively increases the scope of the vulnerable funds.

Storage and compute are getting exponentially cheaper with each passing year, bringing the cost down significantly with time.

IAm0x52

My final set of thoughts on the matter:

I don't think there is anything much hypothetical left in this scenario.

The profit motive is huge. Relatively low use applications garner millions in exploitable approvals (i.e. Sushiswap Router)

You can see a coded POC here showing exactly what I'm talking about in the link below. It generates 2ˆ8 random addresses then looks for matches in a domain of 2ˆ24. The average across 10 runs is 65586, which is almost perfectly 2ˆ16 (65536). This exactly lines up with the figures presented in my original submission.

https://gist.github.com/zobront/263f7cd6df079a48d9f96c83ef369f69

There is no hypothetical to the collision. The collision WILL happen as shown from the POC above. The collision resistance is 2ˆ81 which is not suitable collision protection.

The Bitcoin network performs ~2ˆ67 hashes per second, which means the network's hash power is enough to perform this attack every ~2 hours. Meanwhile, the block rewards for Bitcoin are 6.25 BTC / block, implying that this much hash power is purchased for ~$2mm every 2 hours. The compute is already possible.

Optimization to storage makes storage requirements possible as pointed out in @jkoppel comment above.

Compute and storage will continue to become cheaper and the contract is immutable, so no upgrade can fix this. The cost of exploit becomes lower with each passing day.

Another consideration is that subsequent attacks become half as expensive. Once you have the EOA's generated you can reuse them to start attacking the next target coin.

Last but not least this entire exploit is easily preventable with a single change to the contract:

> Verify with the factory that msg.sender is a valid pool

I think at this point I believe I have adequately proved this is an medium severity issue according to Sherlock criteria. Since it is not only possible (even more so in the future) but also profitable, causing massive losses to users. IMO the discussion here should be whether this is a medium or high risk issue, not between medium and low.

nevillehuang

I was initially skeptical of this finding too, due to the large computing power required for the attack. But given how easy the fix is, and the potential positive profit margin ever increasing for say a high TVL pool, this should be a valid M given sherlock's rule here:

> 3. A material loss of funds, no/minimal profit for the attacker at a considerable cost

Trumpero

Respectfully, here is my final statement for this issue:

I believe this issue is only have impact in the far future. Using the Bitcoin network's computing power to estimate this issue amplifies the hypothesis for exploitation and doesn't make sense, since the power of the Bitcoin network is much higher than that of the most powerful supercomputers in the world. The current hashrate of Bitcoin is 449.64 EH/s (4.49e20), while the total hashrate of the top 10 supercomputers in the world is less than 2000 TH/s (2e15) (I used the latest statistics from the top500 website for supercomputers to compute the hashrate https://www.top500.org/lists/top500/2023/06/ and used the approximate conversion from FPlops to hashrate from there https://www.quora.com/How-do-you-convert-m-flop-s-to-hash-s https://bitcointalk.org/index.php?topic=52303.0)

Therefore, if the Bitcoin network needs 2 hours to generate address collisions, a combination of the top 10 supercomputers would need more than 400,000 hours to exploit it, even though these supercomputers are massively expensive and complex. Any supercomputer always requires a very high and unreal cost for attacker, but its power is still insufficient to exploit this issue. A supercomputer is the most powerful option that attacker can obtain, and it even nearly impossible for attacker to access the most powerful supercomputers in the world. Therefore, I believe it is impossible and unprofitable for attackers to exploit this in the next few years (maybe few decades), as using Bitcoin's computing power amplifies the hypothesis for exploitation.

This issue should be an informational issue in my opinion. However, I will leave it to Sherlock team to make the final decision and will respect the final result.

jkoppel

Comparing the hashrate of the top supercomputers to that of the Bitcoin network is like comparing the carrying capacity of a humanoid robot to that of a container ship. Yes, all the humanoid robots in the world might not be able to carry as much as a single container on a container ship, but that doesn't mean shipping a single container overseas is prohibitively expensive.

It may be true that the supercomputers combined only have 2000 TH/s. But anyone can beat that by spending $30k on Antminers. https://koinly.io/blog/best-crypto-mining-hardware/

T1MOH593

As I understand, all the calculations above state that any random collision is found. However not every address gives approval to router in this specific protocol. Probability that collisionned address has given approval to KyberSwap Router must be also applied. What set of addresses can give approve to KyberSwap among all possible addresses - computation above assumes that all the existing, which is obviously incorrect. If take 1e9 among 2ˆ256 possible (or 2ˆ160, not sure) - above calculation is far from reality. Am I wrong?

jkoppel

@T1MOH593 Yes, you are wrong. The collision needs to be between (a) the address of a pool using some currency, and (b) any EOA controlled by the attacker. Once this collision is found, the attacker has the ability to steal all approvals to the router using that currency.

T1MOH593

@jkoppel Computation above states that 2ˆ80 number of tries needed to find collision with probability 50%. However this computation states that collision between **any** a) evm address and b) any EOA will be found. However a) must be restricted to "(a) the address of a pool using some currency". This condition complicates calculations, because set of satisfied addresses reduces from 2ˆ160 to subset of KyberSwap pools

I really appreciate if someone corrects me

jkoppel

@T1MOH593 I don't know where you're getting that from, so it's hard for me to give a response more constructive than "read the Router code and understand how a collision between an arbitrary attacker-controlled EOA and an arbitrary pool with a fixed currency allows major theft."

Then if you can iterate through 2ˆ80 addresses for KyberSwap pools all of which use currency A, and 2ˆ80 random EOA addresses, you have a large collision probability.

T1MOH593

My bad, you're right. Take a look on Router code helped Thanks for explanation, I expected I was wrong

hrishibhat

Result: Medium Unique Based on all the information from the comments above and discussing further with various parties, this issue seems to be a serious issue and while the timeline is unclear but depends on the computational power available and other factors mentioned above. Its possibility is likely in the near future from hours to days to months to years and may not be a hypothetical scenario anymore.

While Sherlock understands the details of this particular issue may evolve in the future, from the perspective of this contest given that the protocol can still fix this, considering this issue a valid medium.

Note: Given the nature of this issue, and the possibility of more information surfacing on the same, this issue must be referred to for future contests only under matching scenarios, and would advice against drawing conclusions/interpretations inspired by this issue on any of the future issues. Also currently the rulebook does not have a fixed timeline within which the issue must occur. This will be discussed further and added to the rule.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- IAm0x52: accepted

c14sh

Great discussion! @IAm0x52 Is the PoC written by zobront? or is it that 0x52 == zobront?

c14sh

> Great discussion! @IAm0x52 Is the PoC written by zobront? or is it that 0x52 == zobront?

Seems the PoC is finding the first hash of 256 hashes in 24bit space, not collision. Surely, the answer is 2ˆ24/256 = 65536 in average from probabilistic perspective.

manhlx3006

[Will fix in the upcoming version of the protocol]

Thanks for all provided information and explanation. We understand the risk of the issue.

Given the fact that the timeline is unclear and depends on many factors, as well as the Factory has been deployed on all supported networks (which can not have logics of storing the list deployed pools), we will put this fix into our in-progress and up-coming protocol.

For the Router contract, since it is used mostly by developers (as we are an Aggregator and users interact with another Router instead of this one), we will put a notice into our docs and advice developers to be careful when approving token allowances to this Router contract, and the new fix will also be added into the upcoming version, and adjust the Router logic accordingly.

In the mean time, we are analyzing the potential risks as well as the likelihood of the issue (computation power, expenses, etc).

Thanks all.