SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Stealth
Prepared by: Sherlock
Lead Security Expert: hash
Dates Audited: February 13 - February 16, 2024
Prepared on: February 23, 2024

Introduction

To be provided shortly.

Scope

Repository: nero-tang/uniswap_router_contract

Branch: main

Commit: 0292a9a3228933894556a5471ae55c7e8cf967e6

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

Issue M-1: `UniswapV3RouterUpgradeable.sol` is vulnerable to address collission

Source: https://github.com/sherlock-audit/2024-02-stealth-judging/issues/1

Found by

MohammedRizwan, hash

Summary

`UniswapV3RouterUpgradeable.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

Before going in details, This issue is referenced from this issue which was also found in Kyberswap audit at sherlock. Further this sherlock comment states to mention this particular reference for similar issues

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 2^160 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,

For more details, refer this article on `The probability of a hash collision`. Also refer this issue.

uniswapV3SwapCallback() calls `verifyCallback()` to verify the callback.

```
  function uniswapV3SwapCallback(
    int256 amount0Delta,
    int256 amount1Delta,
    bytes calldata _data
  ) external override {
    require(amount0Delta > 0 || amount1Delta > 0);
    SwapCallbackData memory data = abi.decode(_data, (SwapCallbackData));
@>    (address tokenIn, address tokenOut, address payer) = verifyCallback(data);

   . . . some code
  }
```

verifyCallback().

```
function verifyCallback(
  SwapCallbackData memory data
) internal view returns (address tokenIn, address tokenOut, address payer) {
  (tokenIn, tokenOut, payer) = (data.tokenIn, data.tokenOut, data.payer);
  address poolAddress = computePoolAddress(tokenIn, tokenOut, data.fee);
  require(msg.sender == poolAddress);    @audit// only checks msg.sender from
 ↪  computed pool address
}
```

The `verifyCallback()` used in `uniswapV3SwapCallback()` function is used to verify that msg.sender is the address of the pool and only a require check is performed to verify if msg.sender has been computed via computePoolAddress().

According to the UniswapV3 Doc,

> when using uniswapV3SwapCallback, the caller of this method must be verified to be a UniswapV3Pool deployed by the canonical UniswapV3Factory.

However, the function never checks with the `factory` that the pool exists or any of the inputs are valid in any way. tokenIn can be constant and we can achieve the variation in the hash by changing tokenOut. The attacker could use tokenIn = WETH and vary tokenOut. 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.

This comment further proves the attack is possible.

Impact

The pool address check in the the callback function isn't strict enough and can suffer issues with collision. **Address collision can cause all allowances to be drained.** 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.

Code Snippet

https://github.com/sherlock-audit/2024-02-stealth/blob/main/uniswap_router_contract/contracts/base/UniswapV3RouterUpgradeable.sol#L114

Additional References

1) Reference 1
2) Reference 2

Additional information to consider:

1) Vitalik Buterin in November 2021 claims that: “even without address space extension, collisions today take 2^80 time to compute, and that length of time is already within range of nation-state-level actors willing to spend huge amounts of resources. For reference, the Bitcoin network performs 2^80 SHA256 hashes once every two hours.” https://ethresear.ch/t/what-would-break-if-we-lose-address-collision-resistance/11356
2) Mysten Labs have recently (October 2023) estimated the cost of a collision attack. They claim that “it's not a surprise that today, a 2^80 attack would cost a few million dollars”. https://mystenlabs.com/blog/ambush-attacks-on-160bit-objectids-addresses

Tool used

Manual Review

Recommendation

Verify with the factory that `msg.sender` is a valid pool i.e Change the verifyCallback to call factory.getPool to check that msg.sender is a Uniswap pool deployed by the factory

Discussion

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/nero-tang/uniswap_router_contract/pull/1.

10xhash

> The protocol team fixed this issue in PR/commit nero-tang/uniswap_router_contract#1.

@nero-tang @maxrails

Fixes the uniswapV3Callback

Since computePoolAddress is still used inside the `swap` calls, it opens a possibility to make the call to an attacker controlled contract. In `exactInputTokensForETH` such

a call will allow the attacker contract to specify any random number for amountOut which will be withdrawn from WETH. However this should not be a problem unless WETH is deposited externally for the contract. Just wanted to mention this

nero-tang

> The protocol team fixed this issue in PR/commit nero-tang/uniswap_router_contract#1.

> @nero-tang @maxrails
>
> Fixes the uniswapV3Callback
>
> Since computePoolAddress is still used inside the `swap` calls, it opens a possibility to make the call to an attacker controlled contract. In `exactInputTokensForETH` such a call will allow the attacker contract to specify any random number for amountOut which will be withdrawn from WETH. However this should not be a problem unless WETH is deposited externally for the contract. Just wanted to mention this

Hi @10xhash, thanks for point it out. Yes we don't deposit WETH externally so it should not be a problem in general.

sherlock-admin

The Lead Senior Watson signed off on the fix.

Issue M-2: `UniswapV3RouterUpgradeable::exactInputETHForTokens` does not check whether the received amount is greater than or equal to `amountOutMin`

Source: https://github.com/sherlock-audit/2024-02-stealth-judging/issues/2

Found by

ck, hash

Summary

`UniswapV3RouterUpgradeable::exactInputETHForTokens` does not check whether the received amount is greater than or equal to `amountOutMin`

Vulnerability Detail

When a swap is done through `UniswapV3RouterUpgradeable::exactInputETHForTokens`, it is expected that the account should receive at least the `amountOutMin` specified in the parameters:

```
function exactInputETHForTokens(
  uint256 amountOutMin,
  address tokenOut,
  uint24 poolFee,
  uint256 feeBips,
  uint256 deadline,
  uint256 ethAmountToCoinbase
)
```

The check implemented is flawed as it doesn't take into account fee on transfer tokens.

```
(int256 amount0, int256 amount1) = IUniswapV3Pool(poolAddress).swap(
  msg.sender,
  zeroForOne,
  SafeCast.toInt256(amountIn),
  zeroForOne ? MIN_SQRT_RATIO + 1 : MAX_SQRT_RATIO - 1,
  abi.encode(SwapCallbackData(WETH, tokenOut, address(this), poolFee))
);


actualAmountIn = uint256(zeroForOne ? amount0 : amount1);
```

```
amountOut = uint256(-(zeroForOne ? amount1 : amount0));
require(amountOut >= amountOutMin, "INSUFFICIENT_OUTPUT_AMOUNT");
```

This just checks the result of the swap function instead of checking the recipient's balance before and after the swap.

Impact

The recipient can receive less than the minimum amount they specified translating to a loss of funds.

Code Snippet

https://github.com/sherlock-audit/2024-02-stealth/blob/main/uniswap_router_contract/contracts/base/UniswapV3RouterUpgradeable.sol#L27-L34

https://github.com/sherlock-audit/2024-02-stealth/blob/main/uniswap_router_contract/contracts/base/UniswapV3RouterUpgradeable.sol#L47-L57

Tool used

Manual Review

Recommendation

Add a check to ensure that the change in balance of the recipient before and after the swap is greater than the minimum output amount they specified.

Discussion

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/nero-tang/uniswap_router_contract/pull/1.

T1MOH593

Escalate

Fee-on-transfer tokens are not supported by Uniswap https://docs.uniswap.org/concepts/protocol/integration-issues

sherlock-admin2

> Escalate
>
> Fee-on-transfer tokens are not supported by Uniswap https://docs.uniswap.org/concepts/protocol/integration-issues

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.

iamckn

Escalate

From the link the first escalator has shared - https://docs.uniswap.org/concepts/protocol/integration-issues one of the suggestions is for the creation of customised routers which in essence is what the Stealth protocol is. That link is actually guidance to protocols such as Stealth to implement their own custom logic to help in dealing with these issues.

Stealth also listed USDT explicitly as one of the tokens with non standard behaviors that will be used.

These were therefore issues that we were expected to investigate.

Also note that this issue goes beyond just fee on transfer tokens and addresses any additional issues related to slippage that may be brought about by any other situation. The protcocol has actually implemented my recommended solution to ensure the user always receives an amount greater than or equal to `amountOutMin`.

sherlock-admin2

> Escalate
>
> From the link the first escalator has shared - https://docs.uniswap.org/concepts/protocol/integration-issues one of the suggestions is for the creation of customised routers which in essence is what the Stealth protocol is. That link is actually guidance to protocols such as Stealth to implement their own custom logic to help in dealing with these issues.
>
> Stealth also listed USDT explicitly as one of the tokens with non standard behaviors that will be used.
>
> These were therefore issues that we were expected to investigate.
>
> Also note that this issue goes beyond just fee on transfer tokens and addresses any additional issues related to slippage that may be brought about by any other situation. The protcocol has actually implemented my recommended solution to ensure the user always receives an amount greater than or equal to `amountOutMin`.

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.

cvetanovv

This issue should remain valid Medium.

From the Uniswap documentation, we see: "Fee-on-transfer tokens will not function with our router contracts" - Reference. But it also says that if someone wants to support them they have to customise their router. And that is exactly the case here. The protocol has written that he wants to support them and Watson shows how the user can get fewer tokens than he has set as `amountOutMin`.

0xRizwan

Sponsor comment on FOT #4

> We will align with the official Uniswap router on this one because there's no standard approach to determine transfer tax at runtime.

It seems protocol team wants to go with official uniswap v3 router implementation without any further changes like support for FOT, etc

In addition, the readme mentions FEE-ON-TRANSFER tokens support mainly indicating for uniswap V2.

> Are there any FEE-ON-TRANSFER tokens interacting with the smart contracts? The contract will interact with any FEE-ON-TRANSFER token that has Uniswap V2 or V3 pairs.

The wording used here is `or` and not `and`, it means the protocol only want to go with FOT tokens supported by default which is uniswap V2 in this case, also the protocol wont go with FOT tokens for uniswap v3 as it does not support by default.

Imo, @T1MOH593 escalation is correct.

iamckn

That does not make the issue invalid. The protocol can decide to follow a different path after an issue is identified.

Note that uniswap V3 doesn't prevent the use of fee on transfer pairs, only implementation of their router and that's why they advise protocols to implement their own custom router.

The stealth protocol is a custom router and part of the contest was to identify these issues. From the submissions of the LSW, and also the comments of the lead judge, this is the consensus. Also note that the sponsor has confirmed this issue.

cvetanovv

The most important information to look at is in the Readme. The comment from the sponsor was written a few days ago and at the time of writing the report, Watson did not know this information.

Nowhere do I see in the readme that: "protocol **only** want to go with FOT tokens supported by **default** which is uniswap V2 in this case."

There it is clearly written: "The contract will interact with **any** FEE-ON-TRANSFER token that **has Uniswap V2 or V3 pairs**."

T1MOH593

Need to clarify why raised escalation. I read in Readme that router will interact with any current pairs on Uniswap. Then I checked If Uniswap supports fee-on-transfer tokens and saw that they don't support it. In my understanding `customized` in Uni docs means "customized for exact token", because fee can be implemented in different way. What I saw from code is that is NOT custom implementation of Router, it's standard router with cut functionality and additional fee in ETH. Never came to mind that it must handle fee-on-transfer tokens contrary to standard router

iamckn

This is a custom router, otherwise they would just have cloned the v3 swap router. They mentioned support for any token even going ahead to mention USDT as a coin with non standard behaviour. The LSW and the judge also approached the contest with this in mind based on the comments and the issues submitted.

I therefore have to say, this cannot be logically invalidated while Uniswap does say protocols should create custom routers to support these tokens which have been mentioned in this contest's readme.

Czar102

I think this issue has been correctly judged. Planning to reject escalations and leave the issue as is.

Czar102

I am worried that the protocol team may have applied the recommendation from this issue.

This approach may yield more severe issues (exploits!), see an analogous vuln: https://kebabsec.xyz/posts/critical_vulnerability_in_uniswapx/

@nero-tang @maxrails

edit. There is no callback used, so if the attacker can't gain execution during the token transfer, it should be fine.

nero-tang

We will go with the official Uniswap v3 router implementation at the moment, so FOT tokens will not be supported for v3 pools.

0xRizwan

@Czar102 The sponsor has already disputed this issue on #4 and also the above comment. This is design choice or i would say an intended design by protocol team. FOT balance check is taken care in UniswapV2RouterUpgradeable.sol and not in UniswapV3RouterUpgradeable.sol due to FOT not supported in uniswap v3 as it seems the sponsor was known about it.

I believe the readme answering FOT question is specifically for uniswap v2 to check the correct implementation while auditing. I am not sure, how the issue is valid here. i had similar thought as T1MOH outlined here

iamckn

The readme is the source of truth for the contest. Comments after judging and what direction the sponsor chooses to take can not be used to invalidate an issue. As I keep saying, the lead judge and LSW have this interpretation otherwise issues like this would not have been submitted and the judge would have invalidated it at the onset. This is therefore just an issue of one group wanting to invalidate the issue because they interpreted the readme a different way.

Czar102

@0xRizwan This file was in scope and the README determined a use case scope wider than the original router implementation. This is why it is an issue, even though the original router is a valid implementation in its own scope.

Czar102

Result: Medium Has duplicates

Rejecting both escalations since no changes are made to this issue.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- T1MOH593: rejected
- iamckn: rejected

10xhash

> The protocol team fixed this issue in PR/commit nero-tang/uniswap_router_contract#1.

@nero-tang @maxrails Fixes the `exactInputETHForTokens` function by comparing the user's balance.

In `exactInputTokensForETH`, amountOut param is compared against the amount received from Uniswap and not the final amount received by the user ie. there is `feeBips` being charged further on the amountOut. Is this intended design?

nero-tang

> The protocol team fixed this issue in PR/commit nero-tang/uniswap_router_contract#1.

> @nero-tang @maxrails Fixes the `exactInputETHForTokens` function by comparing the user's balance.
>
> In `exactInputTokensForETH`, amountOut param is compared against the amount received from Uniswap and not the final amount received by the user ie. there is `feeBips` being charged further on the amountOut. Is this intended design?

Hi @10xhash, in `exacInputTokensForETH` the contract will first withdraw the `amountOut` WETH and deduct fee based on `feeBips` and send the rest to the user. We are not comparing before and after balance on WETH because if the contract doesn't receive the `amountOut` in WETH the withdraw will just simply fail.

10xhash

I think we are still misunderstanding here but I think it is because it is intended design. I meant this:

```
    amountOut = uint256(-(zeroForOne ? amount1 : amount0));

//  changed to afterwards to include fees
//  require(amountOut >= amountOutMin, "INSUFFICIENT_OUTPUT_AMOUNT");

    TransferHelper.safeWithdraw(WETH, amountOut);
    uint256 feeAmount = (amountOut * feeBips) / INVERSE_FEE_BIPS;

=>  require(amountOut - feeAmount >= amountOutMin, "INSUFFICIENT_OUTPUT_AMOUNT");

    TransferHelper.safeTransferETH(msg.sender, amountOut - feeAmount);
```

nero-tang

Ahh yes this design is intentional. The `amountOutMin` is estimated based on the slippage and the `amountOut` from a simulation call.

sherlock-admin

The Lead Senior Watson signed off on the fix.

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.