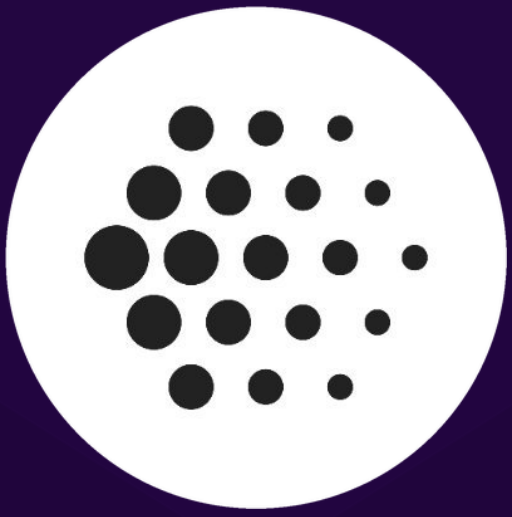
SHERLOCK

SHERLOCK SECURITY REVIEW FOR



Prepared for:	Splits
Prepared by:	Sherlock
Lead Security Expert:	obront
Dates Audited:	April 19 - April 24, 2023
Prepared on:	May 22, 2023

Introduction

The decentralized protocol for secure & efficient onchain payments. Composable, efficient, onchain splits for ETH & ERC20s.

Scope

Repository: 0xSplits/splits-utils

Branch: main

Commit: 0dd263bf6feb0bd26b054da3cf1bb742d0bfb13e

Repository: 0xSplits/splits-oracle

Branch: main

Commit: f6628a116d8721289dad2c70d3e3aa14e4815d4e

Repository: 0xSplits/splits-swapper

Branch: main

Commit: 83ce1124767a097aac37d1cd162a9b27bbc48701

Repository: 0xSplits/splits-pass-through-wallet

Branch: main

Commit: 203badc970b9bb2216cf2ae0e93dcb0a0de19151

Repository: 0xSplits/splits-diversifier

Branch: main

Commit: bdb11a10d9f3aaf731adca09d6a0e05ab359e188

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	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

obront
ctf_sec
theOwl
mstpr-brainbot
0x00ffDa
J4de
alexzoid
nobody2018

Issue M-1: SwapperCallbackValidation doesn't do anything, opens up users to having contracts drained

Source: https://github.com/sherlock-audit/2023-04-splits-judging/issues/9

Found by

0x00ffDa, J4de, alexzoid, nobody2018, obront

Summary

The `SwapperCallbackValidation` library that is intended to be used by contracts performing swaps does not provide any protection. As a result, all functions intended to be used only in a callback setting can be called any time by any user. In the provided example of how they expect this library to be used, this would result in the opportunity for all funds to be stolen.

Vulnerability Detail

The `SwapperCallbackValidation` library is intended to be used by developers to verify that their contracts are only called in a valid, swapper callback scenario. It contains the following function to be implemented:

```
function verifyCallback(SwapperFactory factory_, SwapperImpl swapper_) internal
↪   view returns (bool valid) {
    return factory_.isSwapper(swapper_);
}
```

This function simply pings the `SwapperFactory` and confirms that the function call is coming from a verified swapper. If it is, we assume that it is from a legitimate callback.

For an example of how this is used, see the (out of scope) UniV3Swap contract, which serves as a model for developers to build contracts to support Swappers.

```
SwapperImpl swapper = SwapperImpl(msg.sender);
if (!swapperFactory.verifyCallback(swapper)) {
    revert Unauthorized();
}
```

The contract goes on to perform swaps (which can be skipped by passing empty `exactInputParams`), and then sends all its ETH (or ERC20s) to `msg.sender`. Clearly, this validation is very important to protect such a contract from losing funds.

However, if we look deeper, we can see that this validation is not nearly sufficient.

In fact, `SwapperImpl` inherits from `WalletImpl`, which contains the following function:

```
function execCalls(Call[] calldata calls_)
    external
    payable
    onlyOwner
    returns (uint256 blockNumber, bytes[] memory returnData)
{
    blockNumber = block.number;
    uint256 length = calls_.length;
    returnData = new bytes[](length);

    bool success;
    for (uint256 i; i < length;) {
        Call calldata calli = calls_[i];
        (success, returnData[i]) = calli.to.call{value: calli.value}(calli.data);
        require(success, string(returnData[i]));

        unchecked {
            ++i;
        }
    }

    emit ExecCalls(calls_);
}
```

This function allows the owner of the Swapper to perform arbitrary calls on its behalf.

Since the verification only checks that the caller is, in fact, a Swapper, it is possible for any user to create a Swapper and pass arbitrary calldata into this `execCalls()` function, performing any transaction they would like and passing the `verifyCallback()` check.

In the generic case, this makes the `verifyCallback()` function useless, as any calldata that could be called without that function could similarly be called by deploying a Swapper and sending identical calldata through that Swapper.

In the specific case based on the example provided, this would allow a user to deploy a Swapper, call the `swapperFlashCallback()` function directly (not as a callback), and steal all the funds held by the contract.

Impact

All funds can be stolen from any contracts using the `SwapperCallbackValidation` library, because the `verifyCallback()` function provides no protection.

Code Snippet

https://github.com/sherlock-audit/2023-04-splits/blob/main/splits-swapper/src/peripherals/SwapperCallbackValidation.sol#L11-L19

https://github.com/sherlock-audit/2023-04-splits/blob/main/splits-utils/src/WalletImpl.sol#L43-L65

Tool used

Manual Review

Recommendation

I do not believe that Swappers require the ability to execute arbitrary calls, so should not inherit from WalletImpl.

Alternatively, the verification checks performed by contracts accepting callbacks should be more substantial — specifically, they should store the Swapper they are interacting with's address for the duration of the transaction, and only allow callbacks from that specific address.

Discussion

zobront

Fixed in https://github.com/0xSplits/splits-swapper/pull/3/ by removing the validation and requiring bots to either (a) not hold funds in the contract or (b) create their own validations.

MLON33

Confirming that Splits meant for this fix (3) to be linked to this issue (9):

https://github.com/0xSplits/splits-swapper/pull/3#issue-1693205720

Issue M-2: Oracle tick rounding the wrong direction can lead to Swapper overpaying for swap

Source: https://github.com/sherlock-audit/2023-04-splits-judging/issues/12

Found by

obront

Summary

The UniV3Oracle is intended to be used in situations where asset values can be understated, but must not be overstated. As a result, all results are rounded down to the nearest tick.

However, in the case of the Swapper, we need the opposite. The Swapper's owner has specified the specific discount which they are willing to provide, and the caller is the user calling `flash()` to perform the swap. In this case, rounding must happen against the caller, not against the owner, to ensure the owner's criteria are being met.

Because the Oracle rounds down to the nearest tick (not the nearest decimal price), the difference can be quite dramatic, and can lead to substantial loss of funds for the Swapper owner.

Vulnerability Detail

When the UniV3Oracle calls `OracleLibrary.consult()`, it returns the `arithmeticMeanTick`.

This value is rounded DOWN to the nearest tick. This is because, in most use cases, the price being returned by an oracle is used to determine the value of an asset to be used for something like valuing collateral, where the caller is the one whose collateral is on the line, and it is crucial to ensure that user assets are not overvalued so as to give them an edge.

However, in this case, the oracle is being used to determine the amount owed from the bot performing the swap (from here on "the bot") to the Swap owner (from here on "the owner"). The owner has already included a firm parameter (the scaled offer value) for the discount they are willing to provide. The bot is the caller, who is taking the action within the confines of this predetermined parameter. In this case, the value should be rounded UP to ensure this scaled offer value is protected.

(As an extra data point in reasoning about this, Uniswap rounds values UP when users are buying assets through the platform. In other words, the oracle returns a slightly different price than the actual marginal price that would be received if the bot were buying their assets through the same pool.)

Unfortunately, Uniswap oracle rounding can be quite severe. According to Uniswap's pricing formula, ticks represent 1 basis point (0.01%) of the price. This is true regardless of the price of the asset, the quantity being swapped, or (most importantly) the decimals in the asset.

While in most cases, this may not seem like much, it is likely that Swappers receiving large amounts of funds will only need to offer very slim discounts to incentivize bots. As explained to me by the 0xSplits team in a DM, the slimness of this margin could get so extreme that users may even set a PREMIUM for swaps, because of the risk that the trailing 30 minutes could provide a slight discount.

Proof of Concept

Let's look at an example of an LLC using a Swapper to move all their income into USDC:

- They company earns $10mm per year on chain, which is all sent to the Swapper
- They aren't concerned about frequency of swaps, so they do the math: setting their `$defaultScaledOfferFactor` to 99_99_90 should work out to paying approximately $100 for the year, which seems fair.
- When the assets are swapped, rounding comes into play. Instead of `FAIR_VALUE`, their assets are valued at `FAIR_VALUE - 0.5 basis points`, which equals `FAIR_VALUE * 0.995`. For their $10mm, that's `9,950,000`.
- Finally, their scaled offer factor discounts the price further, down to `9_950_000 * 99_99_90 / 1e6 = 9,949,900.50`.

The result is that they end up paying $50,099.50 for their swaps, instead of $100.

(Note that this same math holds whether the swaps were done all at once or in multiple separate transactions.)

Impact

Because rounding is performed in the wrong direction, Swap owners who set their `$defaultScaledOfferFactor` with a small margin may end up paying substantially more than expected for their swaps.

Code Snippet

https://github.com/sherlock-audit/2023-04-splits/blob/main/splits-oracle/src/UniV3OracleImpl.sol#L274-L282

Specifically, see L35-36 of this Uniswap contract:

https://github.com/Uniswap/v3-periphery/blob/6cce88e63e176af1ddb6cc56e029110289622317/contracts/libraries/OracleLibrary.sol#L16-L41

Tool used

Manual Review

Recommendation

Implement your own version of the OracleLibrary's `consult()` function, which doesn't round the `arithmeticMeanTick` down.

Discussion

zobront

Fixed in https://github.com/0xSplits/splits-oracle/pull/1/ by adding 1 to the tick returned from `OracleLibrary.consult()`

jacksanford1

Confirming that Splits meant for this fix (1) to be linked to this issue (12): https://github.com/0xSplits/splits-oracle/pull/1#issue-1693202318

Issue M-3: Tokens without UniV3 pairs with `tokenToBeneficiary` can be stolen by an attacker

Source: https://github.com/sherlock-audit/2023-04-splits-judging/issues/26

Found by

ctf_sec, mstpr-brainbot, obront, theOwl

Summary

Tokens sent to a Swapper that don't share a UniV3 pool with `tokenToBeneficiary` can be stolen, because an attacker can create such a pool with ultra-low liquidity and maintain it for the length of the TWAP, pricing the tokens at ~0 and swapping to steal them.

Vulnerability Detail

The oracle uses the TWAP price from Uniswap V3 to determine the price of each asset.

If a pair is not listed on Uniswap V3, the oracle will not work, so swapping the asset will not be permitted. In this case, the user should be able to withdraw the asset themselves using the `execCalls()` function.

This will be a relatively common occurrence that doesn't require obscure tokens. Many combinations of tokens on Uniswap are able to be traded because of multi-step hops (ie they don't have a pool directly, but they share a poolmate). However, these multi-step hops are not provided by the oracle. A quick review of Uniswap Pairs List shows that this would impact token pairs as common as MATIC/WBTC or MAKER/FRAX.

In these situations, an attacker could steal all of the tokens in question by performing the following:

- Create a Uniswap V3 pool with the Swapper's `tokenToBeneficiary` and the token in question
- Seed it with an incredibly low amount of liquidity at a ratio that values that token in question at ~0
- Maintain this price for the length of the TWAP, which shouldn't be hard with a new, unused pool and low liquidity (ie if the liquidity amount of lower than the gas needed to perform a swap, arbitrage bots will leave it alone)
- Perform a "swap" from this asset to the `tokenToBeneficiary`, stealing the asset and paying ~0 for it

Impact

Any tokens sent to a Swapper that do not have a pair with `tokenToBeneficiary` in Uniswap V3 can be stolen.

Code Snippet

https://github.com/sherlock-audit/2023-04-splits/blob/main/splits-oracle/src/UniV3OracleImpl.sol#L274-L282

Tool used

Manual Review

Recommendation

`UniV3OracleImpl.sol` should require the pool being used as an oracle to meet certain liquidity thresholds or have existed for a predefined period of time before returning the price to the Swapper.

Discussion

zobront

Fixed by https://github.com/0xSplits/splits-oracle/pull/3/ by only allowing whitelisted pools.

jacksanford1

Confirming that Splits meant for this fix (3) to be linked to this issue (26): https://github.com/0xSplits/splits-oracle/pull/3#issue-1701222164

hrishibhat

Considering this issue as a valid medium given that this attack is possible only in case of non-existent token pair and is common with the issue of low liquidity underlying pool. Combining this issue and #28 with all their duplicates.

hrishibhat

Sherlock's previous rule that held valid for this contest: External Oracle Price Manipulation: Issues related to price manipulation in an external oracle used by the contracts are not considered valid high/medium. Based on the above rule Issue #28 and direct duplicates of the price manipulation of the issue are considered to be low as they are about direct pool manipulation,

However, #26 and its dupes mention non-existent pools which can be created by anyone the obtain the desired price which clearly does not fall under the above rule. Allowing tokens to be used which does not yet have a pool to obtain price from

cannot be considered directly under the price manipulation rule and is a separate issue. Hence considering the above issue and its duplicates as valid medium.

Issue M-4: WalletImpl cannot receive NFTs as intended

Source: https://github.com/sherlock-audit/2023-04-splits-judging/issues/57

Found by

obront

Summary

WalletImpl is intended to turn pass through wallets into generalizable wallets that can be used for any purpose, including receiving NFTs. However, because it does not implement the `onERC721Received()` and `onERC1155Received()` functions, it will not pass the checks for `safeTransferFrom()` and will not be able to be used for NFTs as intended.

Vulnerability Detail

`WalletImpl.sol` is inherited by `PassThroughWalletImpl.sol` and `SwapperImpl.sol` to turn them into generalizable smart contract wallets. It implements an `execCalls()` function that allows the owner to perform any arbitrary action on behalf of the contract.

The purpose of this was explained by Will in the contest Discord channel:

> we wanted it to have the full flexibility of a smart contract wallet for various situations which might arise when using it as a pay to address (eg if you use an address for primary proceeds on artblocks it also must be able to receive & handle NFTS)

However, as it is currently implemented thee contracts will not be able to receive NFTs sent with `safeTransferFrom()`, because they do not implement the necessary functions to safely receive these tokens..

While in many cases such a situation would be Medium severity, looking at Will's example above makes clear that the circumstances in which these wallets will be used could lead to more serious consequences. For example, having a wallet that is entitled to high value NFTs but is not able to receive them is clearly a loss of funds risk, and a High severity issue.

Impact

Any time an ERC721 or ERC1155 is attempted to be transferred with `safeTransferFrom()` or minted with `safeMint()`, the call will fail.

Code Snippet

https://github.com/sherlock-audit/2023-04-splits/blob/main/splits-utils/src/WalletImpl.sol#L9-L66

Tool used

Manual Review

Recommendation

Include `onERC721Received()` and `onERC1155Received()` functions in `WalletImpl.sol`.

Discussion

ctf-sec

@wminshew Would love to hear your thought on this one.

The submission is true but because the issue is confirmed via DM and not in the context readme doc, I think the issue can be a valid low.

wminshew

@ctf-sec definitely a valid issue. I am not sure how loss-of-funds would be defined in my own artblocks example given -- I don't think the NFTs are lost, just that the txn would fail and the artist would have to use a different wallet/address? if the NFTs were indeed lost (vs txn reverting) that would feel like a High to me. if txn reverts, i can see arguments for a low or possibly medium (i suppose even if txn reverts for AB there is a chance this would result in total loss for other protocols?)

ctf-sec

Thanks for your comment. I will add the "low severity" label for now.

In the project doc of the auditing contest.

> Q: Which ERC721 tokens do you expect will interact with the smart contracts? none

wminshew

yes that's a good point and my bad -- the diversifier (pass-through wallet) is expected to interact w arbitrary 721s & 1155s but I flubbed that FAQ sorry

wminshew

> Thanks for your comment. I will add the "low severity" label for now.

thinking about this more i feel like this should be at least medium and possible high -- any nft flow that relied on a pull/claim would've resulted in lost funds

ctf-sec

Comment from Sherlock:

The readme excluding usage of NFT's makes this one really difficult. Have pushed hard for Validating issues based on readme information in the past and not discord messages valid. Will need to stick to it, We need to definitely consider updating contest information mid-contest if needed. Not sure yet how to do that.

Keep it medium from the onERC1155Received() point of view.

In case there is an escalation we can look into it further.

zobront

Fixed in https://github.com/0xSplits/splits-utils/pull/3/

jacksanford1

Confirming that Splits meant for this fix (3) to link to this issue (57): https://github.com/0xSplits/splits-utils/pull/3#issue-1693165292

wminshew

confirmed

securitygrid

Escalate for 10 USDC. It is clearly stated in README.md: ***Q: Which ERC721 tokens do you expect will interact with the smart contracts? none*** so, this issue is not valid H/M.

sherlock-admin

> Escalate for 10 USDC. It is clearly stated in README.md: ***Q: Which ERC721 tokens do you expect will interact with the smart contracts? none*** so, this issue is not valid H/M.

You've created a valid escalation for 10 USDC!

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.

z0ld

Escalate for 10 USDC I would like to bring attention to an issue that I believe deserves further consideration and escalation. As mentioned in the discussion above, the pass-through wallet is expected to interact with arbitrary ERC721s & ERC1155s. However, the rules for the auditing contest mention no expected interaction with ERC721 tokens in the Q&A, and it appears ERC1155 tokens follow the same situation.

sherlock-admin

> Escalate for 10 USDC I would like to bring attention to an issue that I believe deserves further consideration and escalation. As mentioned in the discussion above, the pass-through wallet is expected to interact with arbitrary ERC721s & ERC1155s. However, the rules for the auditing contest mention no expected interaction with ERC721 tokens in the Q&A, and it appears ERC1155 tokens follow the same situation.

You've created a valid escalation for 10 USDC!

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.

hrishibhat

Escalation rejected

While the readme denies any interaction with ERC721 tokens, this issue still holds true for ERC1155 tokens, there is no explicit exclusion of these tokens and the issue raised clearly breaks an intended functionality of the Wallet implementations. This is a valid medium.

sherlock-admin

> Escalation rejected
>
> While the readme denies any interaction with ERC721 tokens, this issue still holds true for ERC1155 tokens, there is no explicit exclusion of these tokens and the issue raised clearly breaks an intended functionality of the Wallet implementations. This is a valid medium.

This issue's escalations have been rejected!

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

Issue M-5: Swapper mechanism cannot incentivize ETH-WETH swaps without risking owner funds

Source: https://github.com/sherlock-audit/2023-04-splits-judging/issues/60

Found by

obront

Summary

Vulnerability Detail

When `flash()` is called on the Swapper contract, pairs of tokens are passed in consisting of (a) a base token, which is currently held by the contract and (b) a quote token, which is the `$tokenToBeneficiary` that the owner would like to receive.

These pairs are passed to the oracle to get the quoted value of each of them:

```
amountsToBeneficiary = $oracle.getQuoteAmounts(quoteParams_);
```

The `UniV3OracleImpl.sol` contract returns a quote per pair of tokens. However, since Uniswap pools only consist of WETH (not ETH) and are ordered by token address, it performs two conversions first: `_convert()` converts ETH to WETH for both base and quote tokens, and `_sort()` orders the pairs by token address.

```
ConvertedQuotePair memory cqp = quoteParams_.quotePair._convert(_convertToken);
SortedConvertedQuotePair memory scqp = cqp._sort();
```

The oracle goes on to check for pair overrides, and gets the `scaledOfferFactor` for the pair being quoted:

```
PairOverride memory po = _getPairOverride(scqp);
if (po.scaledOfferFactor == 0) {
    po.scaledOfferFactor = $defaultScaledOfferFactor;
}
```

The `scaledOfferFactor` is the discount being offered through the Swapper to perform the swap. The assumption is that this will be set to a moderate amount (approximately 5%) to incentivize bots to perform the swaps, but will be overridden with a value of ~0% for the same tokens, to ensure that bots aren't paid for swaps they don't need to perform.

The problem is that these overrides are set on the `scqp` (sorted, converted tokens), not the actual token addresses. For this reason, ETH and WETH are considered identical in terms of overrides.

Therefore, Swapper owners who want to be paid out in ETH (ie where `$tokenToBeneficiary = ETH`) have two options:

1) They can set the WETH-WETH override to 0%, which successfully stops bots from earning a fee on ETH-ETH trades, but will not provide any incentive for bots to swap WETH in the swapper into ETH. This makes the Swapper useless for WETH.
2) They can keep the WETH-WETH pair at the original ~5%, which will incentivize WETH-ETH swaps, but will also pay 5% to bots for doing nothing when they take ETH out of the contract and return ETH. This makes the Swapper waste user funds.

The same issues exist going in the other direction, when `$tokenToBeneficiary = WETH`.

Impact

Users who want to be paid out in ETH or WETH will be forced to either (a) have the Swapper not function properly for a key pair or (b) pay bots to perform useless actions.

Code Snippet

https://github.com/sherlock-audit/2023-04-splits/blob/main/splits-oracle/src/UniV3OracleImpl.sol#L248-L260

Tool used

Manual Review

Recommendation

The `scaledOfferFactor` (along with its overrides) should be stored on the Swapper, not on the Oracle.

In order to keep the system modular and logically organized, the Oracle should always return the accurate price for the `scqp`. Then, it is the job of the Swapper to determine what discount is offered for which asset.

This will allow values to be stored in the actual `base` and `quote` assets being used, and not in their converted, sorted counterparts.

Discussion

zobront

Fixed in https://github.com/0xSplits/splits-swapper/pull/4/files by replacing the `_convertToken` function used by the Swapper to be the identity function, leaving ETH as ETH instead of converting to WETH.

jacksanford1

Confirming that Splits meant for this fix (4) to link to this issue (60): https://github.com/0xSplits/splits-swapper/pull/4#issue-1696988978

securitygrid

Escalate for 10 USDC I also noticed this issue. I didn't submit it because I thought the swapper owner would go bankrupt if he allowed such trading pairs with a discount. ***This is misconfiguration***. In README.MD: Q: Please list any known issues/acceptable risks that should not result in a valid finding. ***user misconfiguration***; swapper#flash callers are expected to be sophisticated (aka will check if a given txn reverts, will use flashbots rpc to avoid FR & owner-griefing, etc) So, this is not a valid H/M.

sherlock-admin

> Escalate for 10 USDC I also noticed this issue. I didn't submit it because I thought the swapper owner would go bankrupt if he allowed such trading pairs with a discount. ***This is misconfiguration***. In README.MD: Q: Please list any known issues/acceptable risks that should not result in a valid finding. ***user misconfiguration***; swapper#flash callers are expected to be sophisticated (aka will check if a given txn reverts, will use flashbots rpc to avoid FR & owner-griefing, etc) So, this is not a valid H/M.

You've created a valid escalation for 10 USDC!

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.

hrishibhat

Escalation rejected

Valid medium This is not a user misconfiguration, the issue here is that the pair overrides do not work as the system intended for ETH-WETH swaps regardless of the user input. So the complexity of the user does not matter here but is about the absent configuration for a pair.

sherlock-admin

> Escalation rejected
>
> Valid medium This is not a user misconfiguration, the issue here is that the pair overrides do not work as the system intended for ETH-WETH

swaps regardless of the user input. So the complexity of the user does not matter here but is about the absent configuration for a pair.

This issue's escalations have been rejected!

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