

BARTER DAO SECURITY AUDIT REPORT

March 15, 2023

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION	3
1.1 Disclaimer	3
1.2 Security Assessment Methodology	3
1.3 Project Overview	6
1.4 Project Dashboard	6
1.5 Summary of findings	15
1.6 Conclusion	17
2.FINDINGS REPORT	18
2.1 Critical	18
2.2 High	18
H-1 Bypassing payment fees through various methods	18
H-2 Hashflow RFQ-integration	20
H-3 Swaps through an ERC777 token can lead to DoS of these swaps	21
2.3 Medium	23
M-1 Facade allows any unauthorized Executor	23
M-2 Anyone can withdraw funds left on <code>SwapExecutor</code>	24
M-3 ETH value above limited value can be lost	26
M-4 A swap with permit can be blocked if a frontrunner swaps using copied permit	27
M-5 Unprotected access to <code>makeCheckpoint()</code> in SwapGuardV2	28
2.4 Low	29
L-1 <code>Ownable</code> can be upgraded to <code>Ownable2Step</code>	29
L-2 Simplifying <code>setFeeAndFeeRecipient()</code> logic for gas optimization	30
L-3 Missing zero-checks	31
L-4 SwapGuardV2 has multiple problems	32
L-5 Protection against accidental ETH sendings does not work	33
L-6 Not effective depositETH() and withdrawETH()	34
L-7 Use <code>encodeCall</code>	35
L-8 Missing caller verification in <code>uniswapV3SwapCallback()</code>	36
L-9 Funds stuck in <code>SwapFacade</code> cannot be withdrawn	37

L-10 A potential overflow due to unsafe math	38
L-11 Use <code>UniswapV2Router02</code> to avoid duplication of code	39
L-12 <code>SwapFacade.swap()</code> gas optimisations	40
2.5 Appendix	42
3. ABOUT MIXBYTES	43

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals

- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

The protocol is a participant of the CowSwap solver and allows producing exchanges of tokens in one transaction. Swaps might be various and are joined in batches, executed sequentially.

1.4 Project Dashboard

Project Summary

Title	Description
Client	BarterDAO
Project name	Argon
Timeline	March 1 2023 - March 6 2023
Number of Auditors	3

Project Log

Date	Commit Hash	Note
11.02.2023	22239aaccdbb78d4aa7ac5c4d0859a9b31c0fc00	Commit for the review
25.02.2023	8f4592bbcf1e9328c18632813fef80a68b465a74	Code with fixes for audit

Date	Commit Hash	Note
03.03.2023	0d364bc386e48a74ffe6499e13a77bf81343631	Code with fixes for re-audit
03.03.2023	592bb5fde1579773013e7a54e9842e4fa40572bf	Commit with all fixes

Project Scope

The audit covered the following files:

File name	Link
contracts/Constants.sol	Constants.sol
contracts/Errors.sol	Errors.sol
contracts/ProtocolHelper.sol	ProtocolHelper.sol
contracts/SwapExecutor.sol	SwapExecutor.sol
contracts/SwapFacade.sol	SwapFacade.sol
contracts/SwapGuardV2.sol	SwapGuardV2.sol
contracts/features/ContractOnlyEthRecipient.sol	ContractOnlyEthRecipient.sol
contracts/features/UniswapV3Executor.sol	UniswapV3Executor.sol
contracts/features/KyberExecutor.sol	KyberExecutor.sol
contracts/helpers/DodoV1Helper.sol	DodoV1Helper.sol
contracts/helpers/DssPsmHelper.sol	DssPsmHelper.sol
contracts/helpers/HashflowHelper.sol	HashflowHelper.sol
contracts/helpers/UniswapV2Helper.sol	UniswapV2Helper.sol
contracts/libs/LowLevelHelper.sol	LowLevelHelper.sol

File name	Link
contracts/libs/SafeERC20Ext.sol	SafeERC20Ext.sol
contracts/libs/TokenLibrary.sol	TokenLibrary.sol

Deployments

1. SwapExecutor

Address: [0xC298ca19313568f297FC5f322392968642Db7597](#)

Verified file	Source file	Comments
File 1 of 32: SwapExecutor.sol	SwapExecutor.sol	audited
File 2 of 32: ContractOnlyEthRecipient.sol	ContractOnlyEthRecipient.sol	audited
File 3 of 32: KyberExecutor.sol	KyberExecutor.sol	audited
File 4 of 32: UniswapV3Executor.sol	UniswapV3Executor.sol	audited
File 5 of 32: Constants.sol	Constants.sol	audited
File 6 of 32: RevertReasonParser.sol	RevertReasonParser.sol	trusted as part of 1inch
File 7 of 32: ISwapExecutor.sol	ISwapExecutor.sol	reviewed
File 8 of 32: SafeERC20Ext.sol	SafeERC20Ext.sol	audited
File 9 of 32: TokenLibrary.sol	TokenLibrary.sol	audited
File 10 of 32: LowLevelHelper.sol	LowLevelHelper.sol	audited
File 11 of 32: ReentrancyGuard.sol	ReentrancyGuard.sol	trusted as part of OpenZeppelin
File 12 of 32: Errors.sol	Errors.sol	audited
File 13 of 32: IERC20.sol	IERC20.sol	trusted as part of OpenZeppelin
File 14 of 32: SafeERC20.sol	SafeERC20.sol	trusted as part of 1inch

Verified file	Source file	Comments
File 15 of 32: SafeCast.sol	SafeCast.sol	trusted as part of OpenZeppelin
File 16 of 32: Kyber.sol	Kyber.sol	reviewed
File 17 of 32: draft-IERC20Permit.sol	draft-IERC20Permit.sol	trusted as part of OpenZeppelin
File 18 of 32: IDaiLikePermit.sol	IDaiLikePermit.sol	trusted as part of 1inch
File 19 of 32: IPermit2.sol	IPermit2.sol	trusted as part of 1inch
File 20 of 32: IWETH.sol	IWETH.sol	reviewed
File 21 of 32: RevertReasonForwarder.sol	RevertReasonForwarder.sol	trusted as part of 1inch
File 22 of 32: IUniswapV3Pool.sol	IUniswapV3Pool.sol	trusted as part of Uniswap
File 23 of 32: IUniswapV3SwapCallback.sol	IUniswapV3SwapCallback.sol	trusted as part of Uniswap
File 24 of 32: IUniswapV3PoolImmutables.sol	IUniswapV3PoolImmutables.sol	trusted as part of Uniswap
File 25 of 32: IUniswapV3PoolState.sol	IUniswapV3PoolState.sol	trusted as part of Uniswap
File 26 of 32: IUniswapV3PoolDerivedState.sol	IUniswapV3PoolDerivedState.sol	trusted as part of Uniswap

Verified file	Source file	Comments
File 27 of 32: IUniswapV3PoolActions.sol	IUniswapV3PoolActions.sol	trusted as part of Uniswap
File 28 of 32: IUniswapV3PoolOwnerActions.sol	IUniswapV3PoolOwnerActions.sol	trusted as part of Uniswap
File 29 of 32: IUniswapV3PoolEvents.sol	IUniswapV3PoolEvents.sol	trusted as part of Uniswap
File 30 of 32: StringUtil.sol	StringUtil.sol	trusted as part of 1inch
File 31 of 32: Address.sol	Address.sol	trusted as part of OpenZeppelin
File 32 of 32: SafeMath.sol	SafeMath.sol	trusted as part of OpenZeppelin

2. SwapFacade

Address: [0x4932D5ff00A91B725Ed89514FB5999BBE8a6A0b6](#)

Verified file	Source file	Comments
File 1 of 16: SwapFacade.sol	SwapFacade.sol	audited
File 2 of 16: SafeERC20.sol	SafeERC20.sol	trusted as part of 1inch
File 3 of 16: ContractOnlyEthRecipient.sol	ContractOnlyEthRecipient.sol	audited
File 4 of 16: ISwapExecutor.sol	ISwapExecutor.sol	reviewed
File 5 of 16: TokenLibrary.sol	TokenLibrary.sol	audited

Verified file	Source file	Comments
File 6 of 16: Errors.sol	Errors.sol	audited
File 7 of 16: Ownable2Step.sol	Ownable2Step.sol	trusted as part of OpenZeppelin
File 8 of 16: IERC20.sol	IERC20.sol	trusted as part of OpenZeppelin
File 9 of 16: draft-IERC20Permit.sol	draft-IERC20Permit.sol	trusted as part of OpenZeppelin
File 10 of 16: IDaiLikePermit.sol	IDaiLikePermit.sol	trusted as part of 1inch
File 11 of 16: IPermit2.sol	IPermit2.sol	trusted as part of 1inch
File 12 of 16: IWETH.sol	IWETH.sol	reviewed
File 13 of 16: RevertReasonForwarder.sol	RevertReasonForwarder.sol	trusted as part of 1inch
File 14 of 16: SafeMath.sol	SafeMath.sol	trusted as part of OpenZeppelin
File 15 of 16: Ownable.sol	Ownable.sol	trusted as part of OpenZeppelin
File 16 of 16: Context.sol	Context.sol	trusted as part of OpenZeppelin

3. ProtocolHelper

Address: [0x6f9508F7d373f42dF19aBb275eFB50E11892F5dE](#)

Verified file	Source file	Comments
File 1 of 17: ProtocolHelper.sol	ProtocolHelper.sol	audited
File 2 of 17: DodoV1Helper.sol	DodoV1Helper.sol	audited

Verified file	Source file	Comments
File 3 of 17: DssPsmHelper.sol	DssPsmHelper.sol	audited
File 4 of 17: HashflowHelper.sol	HashflowHelper.sol	audited
File 5 of 17: UniswapV2Helper.sol	UniswapV2Helper.sol	audited
File 6 of 17: DodoV1.sol	DodoV1.sol	reviewed
File 7 of 17: IERC20.sol	IERC20.sol	trusted as part of OpenZeppelin
File 8 of 17: SafeERC20.sol	SafeERC20.sol	trusted as part of 1inch
File 9 of 17: draft-IERC20Permit.sol	draft-IERC20Permit.sol	trusted as part of OpenZeppelin
File 10 of 17: IDaiLikePermit.sol	IDaiLikePermit.sol	trusted as part of 1inch
File 11 of 17: IPermit2.sol	IPermit2.sol	trusted as part of 1inch
File 12 of 17: IWETH.sol	IWETH.sol	reviewed
File 13 of 17: RevertReasonForwarder.sol	RevertReasonForwarder.sol	trusted as part of 1inch
File 14 of 17: DssPsm.sol	DssPsm.sol	reviewed
File 15 of 17: SafeMath.sol	SafeMath.sol	trusted as part of OpenZeppelin
File 16 of 17: Hashflow.sol	Hashflow.sol	reviewed
File 17 of 17: UniswapV2.sol	UniswapV2.sol	reviewed

4. SwapGuardV2

Address: [0xA6F0329fA07F7D8b4027e3d162428Dc29D28e8e6](#)

Verified file	Source file	Comments
File 1 of 13: SwapGuardV2.sol	SwapGuardV2.sol	audited
File 2 of 13: IERC20.sol	IERC20.sol	trusted as part of OpenZeppelin
File 3 of 13: SafeCast.sol	SafeCast.sol	trusted as part of OpenZeppelin
File 4 of 13: ContractOnlyEthRecipient.sol	ContractOnlyEthRecipient.sol	audited
File 5 of 13: TokenLibrary.sol	TokenLibrary.sol	audited
File 6 of 13: Errors.sol	Errors.sol	audited
File 7 of 13: SafeERC20.sol	SafeERC20.sol	trusted as part of 1inch
File 8 of 13: SafeMath.sol	SafeMath.sol	trusted as part of OpenZeppelin
File 9 of 13: draft-IERC20Permit.sol	draft-IERC20Permit.sol	trusted as part of OpenZeppelin
File 10 of 13: IDaiLikePermit.sol	IDaiLikePermit.sol	trusted as part of 1inch
File 11 of 13: IPermit2.sol	IPermit2.sol	trusted as part of 1inch
File 12 of 13: IWETH.sol	IWETH.sol	reviewed
File 13 of 13: RevertReasonForwarder.sol	RevertReasonForwarder.sol	trusted as part of 1inch

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	3
Medium	5
Low	12

ID	Name	Severity	Status
H-1	Bypassing payment fees through various methods	High	Fixed
H-2	Hashflow RFQ-integration	High	Acknowledged
H-3	Swaps through an ERC777 token can lead to DoS of these swaps	High	Fixed
M-1	Facade allows any unauthorized Executor	Medium	Acknowledged
M-2	Anyone can withdraw funds left on <code>SwapExecutor</code>	Medium	Acknowledged
M-3	ETH value above limited value can be lost	Medium	Acknowledged
M-4	A swap with permit can be blocked if a frontrunner swaps using copied permit	Medium	Fixed
M-5	Unprotected access to <code>makeCheckpoint()</code> in SwapGuardV2	Medium	Fixed
L-1	<code>Ownable</code> can be upgraded to <code>Ownable2Step</code>	Low	Fixed
L-2	Simplifying <code>setFeeAndFeeRecipient()</code> logic for gas optimization	Low	Fixed

L-3	Missing zero-checks	Low	Fixed
L-4	SwapGuardV2 has multiple problems	Low	Acknowledged
L-5	Protection against accidental ETH sendings does not work	Low	Fixed
L-6	Not effective depositETH() and withdrawETH()	Low	Fixed
L-7	Use <code>encodeCall</code>	Low	Fixed
L-8	Missing caller verification in <code>uniswapV3SwapCallback()</code>	Low	Fixed
L-9	Funds stuck in <code>SwapFacade</code> cannot be withdrawn	Low	Fixed
L-10	A potential overflow due to unsafe math	Low	Acknowledged
L-11	Use <code>UniswapV2Router02</code> to avoid duplication of code	Low	Acknowledged
L-12	<code>SwapFacade.swap()</code> gas optimisations	Low	Fixed

1.6 Conclusion

During the audit process, the developers spotted and acknowledged 3 HIGH, 5 MEDIUM, and 12 LOW severity findings. After working on the reported findings, all of them were acknowledged or fixed by the client.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

H-1	Bypassing payment fees through various methods
Severity	High
Status	Fixed in 8f4592bb

Description

SwapExecutor, a contract responsible for executing token swaps, is vulnerable to multiple methods of bypassing payment fees, allowing an attacker to perform swaps without paying the required fees.

The first method involves specifying a personal transfer of all tokens in the final swap, leaving only 2 wei of `tokenToTransfer` on the contract to avoid paying fees. This enables the attacker to evade payment and execute swaps at no cost.

- [SwapExecutor.sol#L146](#)

The second method involves passing a swap-path that ends with a poisonous token created by the attacker. The attacker pays fees in their own token, which is worthless, thus bypassing payment fees.

- [SwapFacade.sol#L35](#)

The third method involves copying the code of SwapExecutor and removing the payment logic, allowing the attacker to continue using the SwapFacade without paying any fees by passing the custom SwapExecutor to `SwapFacade.swap()`.

- [SwapFacade.sol#L29](#)

Overall, these vulnerabilities allow attackers to perform swaps without paying the required fees, leading to potential financial losses for the SwapExecutor owners.

Recommendation

One way to address the vulnerabilities in SwapExecutor is to take a fixed fee in a fixed token, such as Ether, in SwapFacade.

Client's commentary

Client's commentary: we will remove all fee related logics.

MixBytes: contracts do not take fees now, the issue disappeared.

H-2

Hashflow RFQ-integration

Severity

High

Status

Acknowledged

Description

Since we receive `HashflowQuote` before the call occurred, the information at the time of the transaction may not be up to date.

In that case, due to this code:

```
if (amount > quote.maxBaseTokenAmount) {
    emit AmountExceedsQuote(amount, quote.maxBaseTokenAmount);
    quote.effectiveBaseTokenAmount = quote.maxBaseTokenAmount;
} else {
    quote.effectiveBaseTokenAmount = amount;
}
```

part of the money may remain with the `SwapExecutor`.

- [HashflowHelper.sol#L24](#)

Recommendation

We recommend adding a revert if the input `amount` is not actual.

Client's commentary

Client's commentary: Won't fix. We are aware that sometimes we can get more than expected amount of funds. Reverting could lead to situation where an 1M trade with slight positive slippage would revert due to 0.01\$ surplus to hashflow quote. We do not expect that amount difference will be significant, and we can keep this small difference on contract because transferring it to user or somewhere else can be an unwanted and surprising behavior.

MixBytes: If `minReturn` is always actual and the transaction is completed quickly (using `deadline`), then this point can indeed be accepted.

H-3

Swaps through an ERC777 token can lead to DoS of these swaps

Severity High

Status Fixed in 0d364bc3

Description

ERC777 tokens allow hooks before and after balance changes.

Transfer `_from` and `to` can choose the exact receivers of these hooks via

```
Registry.setInterfaceImplementer():
```

- 0x1820a4B7618BdE71Dce8cdc73aAB6C95905faD24

Also, Executor allows anyone to call `executeSwap()` and decide on the following targets and calldata.

1. An attacker uses `executeSwap()` so that Executor calls `ERC1820Registry.setInterfaceImplementer(implementer=attacker)`.
2. User A swaps an ERC777 token to something with low `minReturn`. It must be at least one ERC777 among the list of tokens in swaps.
3. Executor receives ERC777 tokens from Facade (hook).
4. The attacker reverts on the hook.

As a result, any ERC777 token going through Executor will lead to DoS and swap will revert.

So, one of the attack flows can be like this:

1. Attacker uses `executeSwap()` so that Executor calls `ERC1820Registry.setInterfaceImplementer(implementer=attacker)`.
So, anytime Executor receives ERC777 tokens, a frontrunner contract will be called.
2. User A swaps the ERC777 tokens to something with low `minReturn`.
3. Executor receives the ERC777 tokens from Facade (hook).
4. Attacker contract enters `Executor.executeSwap()` and Executor thinks that current ERC777 tokens belong to Attacker. So, he can withdraw these tokens, or swap them to something.

Or at least Attacker can revert when receiving a call, not allowing a swap with this token at all.

Recommendation

We recommend that Facade be the only allowed caller for `SwapExecutor.executeSwap()` with the Reentrancy guard and be sure that `minReturn` is set even for small swaps.

Client's commentary

Client's commentary: Fixed

MixBytes: Fixed as SwapExecutor now reverts on call to the `0x1820...aD24` address, so it is not allowed to set a malicious contract as an interface implementer.

2.3 Medium

M-1	Facade allows any unauthorized Executor
Severity	Medium
Status	Acknowledged

Description

SwapFacade allows to choose any SwapExecutor address, even not authored by the team. Fees are charged on SwapExecutor. So, it is rational to fork SwapExecutor with zero fees and choose it as an executor for swaps.

- [SwapFacade.sol#L87](#)

Some other possible scenarios:

- users can choose out date executors
- users can choose executors with bugs

Recommendation

We recommend having a list of allowed executors and taking fees on SwapFacade.

Client's commentary

Client's commentary: Having fees on SwapFacade won't help either because it can be forked as well as SwapExecutor. Therefore, we remove all fee-related logics as inefficient and gas-consuming.

M-2

Anyone can withdraw funds left on `SwapExecutor`

Severity

Medium

Status

Acknowledged

Description

Funds left on the `SwapExecutor` can be withdrawn by anyone who specifies a transfer call in the swaps.

There are multiple reasons why funds may be left on the `SwapExecutor`:

1. A user may not set the `tokenRatio` to 100% in a swap.

```
uint256 poolSwapAmount = (balanceToSwap * swap.tokenRatio) / _ONE;

# sum (tokenRatio) > _ONE - is not risky, revert happens
# sum (tokenRatio) < _ONE - is risky, as some tokens
    are left not swapped and can be taken by anyone.
```

2. The `SwapExecutor` may run arbitrary calls to protocols that reward it with additional tokens. The `SwapFacade` only checks the `tokenToTransfer`, and a user may forget to transfer other tokens from the `SwapExecutor` immediately.
3. If a user transfers tokens to the `SwapExecutor` first and then runs the `SwapFacade.swap()` function in a separate transaction.

- [SwapExecutor.sol#L58](#)

Recommendation

To address this, a similar approach to that used in CowProtocol could be implemented: allowing only whitelisted managers to execute swaps in `SwapExecutor`.

It is also recommended to check that all `tokenRatio` used eventually sum up to 100%.

Client's commentary

Client's commentary: Won't fix. We are aware of this behavior and we have external checks on side that forms calldata to prevent this from happening. It is also possible to use this as part of bigger flow, i.e. let's consider for example integration with some service that expects to be paid 1% fee otherwise

transaction will revert. We are able to make last step transferring 1% of tokens to some address and this is an entire step. It doesn't sum to 1 as other steps but it's a valid use case we want to support.

M-3	ETH value above limited value can be lost
Severity	Medium
Status	Acknowledged

Description

SwapExecutor has the following lines for ETH:

```
if (value > valueLimit) {
    value = valueLimit;
}
(success, result) = target.call{ value: value }(data);
```

- [SwapExecutor.sol#L116-L120](#)

So, only a limited amount will be used if sent ETH is above valueLimit.

This ETH will be lost on the contract and anyone can withdraw these exceeded amounts.

Recommendation

We recommend redesigning valueLimit purpose so that no funds are lost in edge situations.

Client's commentary

Client's commentary: Won't fix. This is an ETH special case for 2.2.2 (Hashflow) or more generally speaking for any quoting mechanism. We expect this difference to only exist on market movements that can't be big enough (deadline check is in place in order to guarantee this). So the same logics as 2.2.2 applies here except it's about ETH not ERC20 tokens.

M-4	A swap with permit can be blocked if a frontrunner swaps using copied permit
Severity	Medium
Status	Fixed in 8f4592bb

Description

Facade does not check that msg.sender is the owner of a permit.

So, a frontrunner can use permits of other users available from mempool, so that their transactions would revert (as permit are used by the frontrunner).

- [SwapFacade.sol#L58-L64](#)

A user will receive revert and will have to build new calldata, without permit.

Recommendation

We recommend decoding permits to extract their owner, then check that msg.sender is this owner. Some example used by 1inch:

- [SafeERC20.sol#L158-L245](#)

M-5	Unprotected access to <code>makeCheckpoint()</code> in SwapGuardV2
Severity	Medium
Status	Fixed in 8f4592bb

Description

SwapGuardV2 functions are planned to be used as separate calls in a set of swaps. However, if during the swaps process a hacker manages to get a callback to their own contract (for example, if the victim operates with ERC-777 tokens), the hacker can call public accessible function `makeCheckpoint()` to modify the SwapGuardV2 state so that the `ensureCheckpoint()` function at the victim's end would not work correctly.

[SwapGuardV2.sol#L20](#)

Recommendation

It is recommended to revise the approach to using SwapGuardV2. It may be reasonable to leave only one function in SwapGuardV2, which receives a set of tokens, deltas and a callback. The function records the current token balances in memory instead of a storage, then calls the user's callback (which performs swaps), and then checks for changes in the balances.

Client's commentary

Client's commentary: We will add `msg.sender` check so only the person who initially called `makeCheckount` will be able to `ensureCheckpoint`.

2.4 Low

L-1	<code>Ownable</code> can be upgraded to <code>Ownable2Step</code>
Severity	Low
Status	Fixed in 8f4592bb

Description

SwapExecutor uses the `Ownable` functionality:

[SwapExecutor.sol#L24](#)

The `Ownable` contract can be upgraded to Open Zeppelin's `Ownable2Step`:

[Ownable2Step.sol](#)

`Ownable2Step` provides added safety due to its securely designed two-step process.

Recommendation

We recommend applying `Ownable2Step` instead of `Ownable`.

L-2Simplifying `setFeeAndFeeRecipient()` logic for gas optimization**Severity**

Low

Status

Fixed in 8f4592bb

Description

The current implementation manually packs and unpacks the fee and address from a uint256 value, which increases the likelihood of errors and may cause unnecessary gas usage.

- [SwapExecutor.sol#L48](#)
- [SwapExecutor.sol#L137](#)

Recommendation

To simplify the logic and optimize gas usage, it's recommended to store the fee and address separately in the contract's state using two separate variables:

```
uint160 feeAddress;  
uint96 fee;
```

Both variables will fit into a single storage slot. This will also allow more efficient retrieval of the fee-related data.

L-3	Missing zero-checks
Severity	Low
Status	Fixed in 8f4592bb

Description

There are no zero checks for `recipient` and `minReturn`:

- [SwapFacade.sol#L32](#)
- [SwapFacade.sol#L33](#)

Some programs may pass zero values for some arguments by default. A user may not notice this behaviour and lose funds.

Recommendation

It is recommended to add requirements which check that the `recipient` and `minReturn` are not zero.

L-4	SwapGuardV2 has multiple problems
Severity	Low
Status	Acknowledged

Description

1. Contracts that use SwapGuardV2 together with Facade+Executor are out of scope
2. All functions are public and anyone can makeCheckpoint()
3. This contract will not work with native ETH.
4. allowedLoss adds up with tokens having different decimals. Or tokensPrices must at least include decimals
5. Lengths of tokens, tokenPrices and balanceChanges are not checked to be the same
6. All checkpoints are written to storage, no need if it is used in one transaction

- [SwapGuardV2.sol#L20-L58](#)

Recommendation

We recommend using more efficient ways to check profitability of swaps.

Client's commentary

MixBytes: Points 2 and 3 are fixed, the other points are acknowledged. After an internal discussion with the customer, we concluded that the status is Acknowledged.

L-5

Protection against accidental ETH sendings does not work

Severity

Low

Status

Fixed in 8f4592bb

Description

Both Facade and Executor inherit ContractOnlyEthRecipient.sol with the following code.

```
/**
 * @title ContractOnlyEthRecipient
 * @notice Base contract that rejects any direct ethereum deposits.
 * This is a failsafe against users who can accidentally send ether
 */
abstract contract ContractOnlyEthRecipient {
    receive() external payable {
        // solhint-disable-next-line avoid-tx-origin
        if (msg.sender == tx.origin) {
            revert DirectEthDepositIsForbidden();
        }
    }
}
```

In fact, only Executor needs this inheritance, as it can receive ETH from external exchanges. But Facade should only receive ETH as msg.value at swap().

Also, in the comments developers stated that it is against accidental ETH sendings. By the way, it still allows to receive accidental ETH from users as contracts (like user wallets). Moreover, any accidental ETH on balances are bad (can be stolen), so it is better to remove options to receive accidental money when it's possible.

Recommendation

We recommend removing `recieve()` for `SwapFacade` and `SwapGuardV2`.

L-6	Not effective depositETH() and withdrawETH()
Severity	Low
Status	Fixed in 8f4592bb

Description

`depositETH()` and `withdrawETH()` are inherited and make calls to WETH.

It is used to swap between native ETH and WETH. But the implementation makes Executor call itself (external call), send ETH itself.

This step is useless and can be dropped.

```
function depositWeth(uint256 amount) external payable {
    if (amount != msg.value) {
        revert EthValueAmountMismatch();
    }
    weth.deposit{value: amount}();
}

function withdrawWeth(uint256 amount) external {
    weth.withdraw(amount);
}
```

Recommendation

We recommend removing these functions and use WETH as target directly.

L-7	Use <code>encodeCall</code>
Severity	Low
Status	Fixed in 8f4592bb

Description

`abi.encodeCall` is a safer way to avoid mistakes during compilation. The compiler checks that the types of args are compatible with a call.

- [UniswapV2Helper.sol#L31](#)
- [HashflowHelper.sol#L30](#)
- [DssPsmHelper.sol#L31](#)
- [DodoV1Helper.sol#L22](#)

Recommendation

We recommend changing `encodeWithSelector` to `encodeCall`.

L-8	Missing caller verification in <code>uniswapV3SwapCallback()</code>
Severity	Low
Status	Fixed in 8f4592bb

Description

The `uniswapV3SwapCallback()` function in the `IUniswapV3SwapCallback` interface, which is part of the Uniswap v3 protocol, is used to handle the results of a swap. However, the function in `UniswapV3Executor` does not include any caller verification, which can lead to potential vulnerabilities.

- [UniswapV3Executor.sol#L20](#)

Recommendation

The uniswap team recommends checking that the caller of the `uniswapV3SwapCallback()` is a `UniswapV3Pool` deployed by the canonical `UniswapV3Factory`:

- [IUniswapV3SwapCallback.sol](#)

Note that this increases the amount of gas used.

L-9	Funds stuck in <code>SwapFacade</code> cannot be withdrawn
Severity	Low
Status	Fixed in 8f4592bb

Description

ERC-20 token funds cannot be withdrawn from SwapFacade if a user accidentally sends tokens to the SwapFacade contract instead of SwapExecutor.

- [SwapFacade.sol#L16](#)

Recommendation

It is recommended to add a `sweep(token) onlyOwner` function with the onlyOwner modifier. This function will allow the owner of the contract to sweep any funds that are stuck in the contract and transfer them to a designated account. This will provide a safety net for users who accidentally send funds to the contract, preventing their funds from being lost or stolen.

L-10	A potential overflow due to unsafe math
Severity	Low
Status	Acknowledged

Description

In `SwapExecutor` this line `uint256 poolSwapAmount = (balanceToSwap * swap.tokenRatio) / _ONE` allows making overflow.

Example ([SwapExecutor.sol#L73](#)):

```
uint256 poolSwapAmount = (balanceToSwap * swap.tokenRatio) / _ONE;
```

Recommendation

We recommend removing unsafe math from critical functions.

Client's commentary

Client's commentary: Won't fix. We would like to reenale checked math for this multiplication but solidity provide no such tools. After considering possible fixes we decided on touchin this for following reasons: overflow would reult in drastical reduction of swap input on the step, which will lead to great reduction in output token. This should be handled by minReturn or other slippage tolerance techniques. Moreover, overflow can only happen if left argument is greater than $2^{(256-18)}$, since 2^{18} is a limit for tokenRatio. Considering most of tokens have total supply around 2^{30} it looks unrealistic to hit these bounds.

L-11	Use <code>UniswapV2Router02</code> to avoid duplication of code
Severity	Low
Status	Acknowledged

Description

`UniswapV2Helper` duplicates the existing code of `UniswapV2Router` from the official Uniswap repo.

- [UniswapV2Helper.sol#L18](#)

Recommendation

We recommend using the existing functionality to avoid potential mistakes.

Client's commentary

Client's commentary: Won't fix. We would like to not pay for external call, we checked that our implementation is identical and wish to stick with it.

MixBytes: Calling `swapUniswapV2` in Protocol Helper requires an external call too. There is no error here, so the status is Acknowledged.

L-12`SwapFacade.swap()` gas optimisations**Severity** Low**Status** Fixed in 8f4592bb**Description**

The logic in `SwapFacade.swap()` can be simplified to always call the `_permit()`.

The logic at lines 59-73

[SwapFacade.sol#L63](#)

```
uint256 currentBalance = sourceToken.balanceOf(address(executor));
if (currentBalance < amount)
{
    if (permit.length > 0) {
        _permit(address(sourceToken), permit);
    }
    uint256 approveAmount = sourceToken.allowance(
        msg.sender,
        address(this)
    );
    if (approveAmount < amount) {
        revert NotEnoughApprovedFundsForSwap(approveAmount, amount);
    }
    sourceToken.safeTransferFrom(msg.sender, address(executor), amount);
}
```

can be simplified down to just two lines (always call `_permit()`):

```
_permit(address(sourceToken), permit);
sourceToken.safeTransferFrom(msg.sender, address(executor), amount);
```

Note that the 1inch `_permit()` is already checking `permit.length` and does nothing in case it is zero:

```
contract Permitable {
    error BadPermitLength();

    function _permit(address token, bytes calldata permit)
    internal virtual {
        if (permit.length > 0) {
            bool success;
```

```

        if (permit.length == 32 * 7) {
            // solhint-disable-next-line avoid-low-level-calls
            (success,) = token.call(
                abi.encodePacked(IERC20Permit.permit.selector, permit)
            );
        } else if (permit.length == 32 * 8) {
            // solhint-disable-next-line avoid-low-level-calls
            (success,) = token.call(
                abi.encodePacked(IDaiLikePermit.permit.selector, permit)
            );
        } else {
            revert BadPermitLength();
        }
        if (!success) {
            RevertReasonForwarder.reRevert();
        }
    }
}

```

Recommendation

It is recommended to remove unnecessary conditions to save gas.

2.5 Appendix

1 Monitoring Recommendation

The project contains smart contracts that require active monitoring. For these purposes, it is recommended to proceed with developing new monitoring events based on Forta (<https://forta.org>) with which you can track the following exemplary incidents:

- Using an unexpected `Executor` during swap
- Either ERC20 tokens or ETH remain on the balance of contracts (SwapExecutor, SwapFacade)
- A wrong `minReturn` or `deadline` are passed

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts



https://github.com/mixbytes/audits_public



<https://mixbytes.io/>



hello@mixbytes.io



<https://twitter.com/mixbytes>