MesonFi Protocol Security Assessment

February 26, 2022

Prepared For:
[Undisclosed Recipients]

Prepared By:
[Undisclosed Authors]

Georgia Tech

Table of Content

Summary

From February 04, 2022 through February 26, 2022, Obsidian Labs engaged SSLab@Gatech to review the security of its MesonFi Protocol smart contacts. We conducted this assessment working with the above mentioned security researcher on commit hash **47836cbe71dce8e5c25390640bb204d4e81f5ca1** from the MesonFi/meson-contracts-solidity repository. The MesonFi protocol is a fast and low-cost exchange protocol for cross-chain stablecoins. The system uses off-chain cryptographic signatures and multi-step swaps to verify identities of exchanging parties and thus securely issuing cross-chain exchanges.

During the first week, the assessment focused on the Meson smart contracts. This includes a quick skim of the codebase to get ourselves familiar with its architecture and operating logic, as well as putting specific focus on core functions. During the second week, the assessment mainly focused on discussing the issues we found in the first week and communicating with the developer about fixes. The majority of the time has been spent in verifying the implicit timing order of exchange logics and secureness of the off-chain cryptographic signatures. We also briefly evaluated the javascript client-facing sdk during our second week.

Throughout the engagement, we identified 9 issues, ranging in severity from informational to high. We discovered:

- Issue related to incorrectly encoded shiting operations
- Issue related to cryptographic signatures
- Issue related to vulnerable high-volume exchange
- Issue related to swap locking logic
- Issue related to front running a swap
- Issue related to code readability and redundancy

Overall the code reviewed consists only of the onchain portion and the client facing SDK. We recommend that other security critical components such as the offline relayer should also be security reviewed.

Disclaimer

This audit report should not be used as investment advice.

We make best effort at finding security issues in smart contracts, however we do not provide any guarantees on eliminating all possible security issues. As a single audit-based assessment cannot and should not be considered comprehensive, we always recommend proceeding with several other independent audits and a public bug bounty program to ensure the security of smart contracts.

Project Background

MesonFi's cross-chain stable coin exchange is done in 5 steps.

1. Any party that is willing to match a swap needs to deposit their stablecoin assets in the **MesonPool** in advance. The party (referred to as the Liquidity Provider or simply LP) will receive its **providerIndex** upon successful registration. Note that the LP needs to register on all chains they are willing to swap.

2. An initiator of a swap generates an offline request signature.and posts a swap request via the off-chain relayer or directly calls **MesonSwap::postSwap()** function on the source chain along with other information required for the swap (such as requested swap amount, expiration time, swap src info, swap dest info, etc.) packed into an **encodedSwap**. The initiator will also need to provide its address for signature verification purposes.

   If this request is done through the off-chain relayer, then the request proxies to participating LPs. Among receiving the request the LP will also call **MesonSwap::postSwap** function on the source chain but additionally provide its registered **providerIndex**. Note that the initiator address should still be the initiating user's address in this case.
   If this request is done by the user, then they need to make sure that the **providerIndex** is set to 0 so that it will be detected and successfully bonded by a participating LP in 2.5

2.5 Upon finding a user posted swap (not through the relayer), a participating LP can bond the swap by calling **MesonSwap::bondSwap** function along with its **providerIndex**.

3. The LP then invokes the **MesonPool::lock** function on the destination chain to lock his funds matching the swap request.

4. The user generates an offline release signature and unlocks the LP's funds on the destination chain by calling **MesonPool::unlock**

5. The LP acquires the user's deposit via the same release signature provided by the initiator by calling **MesonSwap::executeSwap** on the source chain

Audit Scope

The assessment scope contains mainly the on-chain smart contract and the client-facing javascript SDK in MesonFi/meson-contracts-solidity at commit **47836cbe71dce8e5c25390640bb204d4e81f5ca1**. Other components such as the offchain relayer are out of scope for this assessment. We listed the files we have audited below.

Table 1 Smart Contract Audit Scope

Contract path
contracts/Pools/IMesonPoolsEvents.sol
contracts/Pools/MesonPools.sol
contracts/Swap/IMesonSwapEvents.sol
contracts/Swap/MesonSwap.sol
contracts/interfaces/IERC20Minimal.sol
contracts/libraries/LowGasSafeMath.sol
contracts/utils/MesonHelpers.sol
contracts/utils/MesonStates.sol
contracts/utils/MesonTokens.sol
contracts/Meson.sol
contracts/MesonConfig.sol
contracts/UpgradableMeson.sol

Table 2 SDK audit scope

File path
packages/sdk/src/MesonClient.ts
packages/sdk/src/SignedSwap.ts
packages/sdk/src/SwapSigner.ts
packages/sdk/src/SwapWithSigner.ts
packages/sdk/src/Swap.ts
packages/sdk/src/index.ts

Vulnerability Dashboard

Vulnerability Summary

Table 3 Vulnerability Summary

Severity	# of Findings
Undetermined	0
High	2
Medium	2
Low	2
Informational	3
Total	9

Category Summary

Table 4 Category Summary

Category	# of Findings
Code With No Effects	2
Improper Authentication	1
Improper Following of Specification by Caller	1
Improper Locking	1
Incorrect Behavior Order	1
Incorrect Calculation	1
Use of a Broken or Risky Cryptographic Algorithm	1
Uncontrolled Resource Consumption	1
Total	9

Key Findings and Recommendations

Table 5 Key Audit Findings

ID	Title	Severity	Type
01	Incorrect typing while calculating lockedSwap	**High**	Incorrect Calculation
02	Outdated Cryptographic Signature Used	**Informational**	Use of a Broken or Risky Cryptographic Algorithm
03	Arbitrary Bonding Unbonded Swaps	**Medium**	Improper Authentication
04	Vulnerable high-volume swapping	**High**	Incorrect Behavior Order
05	Late lock by LP allows initiator to steal LP's assets	**Medium**	Improper Locking
06	Redundant LowGasSafeMath.sol	**Informational**	Code With No Effects
07	Hardcoded shifts are error prone	**Informational**	Code With No Effects
08	Malicious LPs/users can prevent swaps	**Low**	Improper Following of Specification by Caller
09	Malicious users can front-run to permanently occupy swaps	**Low**	Uncontrolled Resource Consumption

Incorrect typing while calculating lockedSwap

Severity: **High**
Target: lock (MesonPools.sol)

Difficulty: Low
Type: Incorrect Calculation

Description

The `MesonPool` contract provides the `lock` function for the LP to lock his fund to match a swap request. At line 92, the registered `providerIndex` of LP is typed as `uint40` but left-shifted `160` bits, leading to an unexpected stored `providerIndex` of 0.

```
92 _lockedSwaps[encodedSwap] = (uint240(block.timestamp + LOCK_TIME_PERIOD) << 200)
93     | (providerIndex << 160)
94     | uint160(initiator);
```

Listing 1 MesonPool::lock

Impact

LPs permanently lose the balance when the lock expires since no initiator can execute the swap with the incorrect providerIndex of 0.

Recommendation

use the correct type for shifting, hence change line 93 to `(uint240(providerIndex) << 160)`

Outdated Cryptographic Signature Used

Severity: **Informational** Difficulty: High
Type: Use of a Broken or Risky Cryptographic Algorithm
Target: _checkRequestSignature, _checkReleaseSingature (MesonHelpers.sol)

Description

The request and release cryptographic signatures used in MesonFi Protocol is a signature based off an early draft of EIP-712 v1 (signTypedData)

Impact

Though no particular vulnerability exists in the current contract code, the signature itself is subject to signature malleability. When the contract in the future uses the signature data directly, for example as an unique Id , this could cause potential issues. The signature also misses important security updates introduced in later versions of EIP-712 (signTypedDatav3, signTypedDatav4), such as the absence of EIP-155 chainID in the signature.

Recommendation

Replace the signature function with the most up to date signature function and avoid implementing it on one's own.

Arbitrary Bonding Unbonded Swaps

Severity: **Medium**
Target: bondSwap (MesonSwap.sol)

Difficulty: Medium
Type: Improper Authentication

Description

The `MesonSwap` contract provides the `bondSwap` function for the LP to bond unbonded swaps. However, the `bondSwap` function lacks validation for `providerIndex`, making it possible for a malicious attacker to bond unbonded swaps to an arbitrary `providerIndex`, even invalid ones.

```
66 function bondSwap(uint256 encodedSwap, uint40 providerIndex) external {
67   uint200 postedSwap = _postedSwaps[encodedSwap];
68   require(postedSwap != 0, "Swap does not exist");
69   require(uint40(postedSwap) == 0, "Swap bonded to another provider");
70
71   _postedSwaps[encodedSwap] = postedSwap | providerIndex;
72   emit SwapBonded(encodedSwap);
73 }
```

Listing 2 MesonSwap::bondSwap()

Possible Attack Scenario

1. User Alice posts an on-chain swap request without specifying a `providerIndex`.
2. Attacker Bob bonds Alice's swap to an invalid `providerIndex`.
3. No legitimate LP can bond to Alice's swap at this point.
4. Alice has to cancel her swap.

In this case, Alice has to pay the gas fee for two transactions, i.e., postSwap + cancelSwap, while Bob only has to pay gas fee for the single bondSwap transaction. This can be considered as a form of DDoS attack to online swap requests.

Impact

Upon presence of malicious LPs, users cannot directly request on-chain swaps themselves, i.e. invoking `postSwap` without going through the relayer.

Recommandation

Validate `indexOfAddress[msg.sender] == providerIndex` in the `bondSwap()` function.

Vulnerable high-volume swapping

Severity: **High**
Target: MesonSwap.sol, MesonPools.sol

Difficulty: High
Type: Incorrect Behavior Order

Description

Any large-sum swap transaction can be vulnerable, especially on those chains with longer block time.

Possible Attack Scenario

1. User Alice attempts to exchange N (N is a huge number) stablecoins and initiates a swap on the source chain, which is then bonded by a malicious LP Bob. Bob then locks his assets of N stablecoins on the destination chain.
2. Alice sends a “release” transaction on the destination chain along with the release signature. This transaction has not been finalized on the chain, but exists in the (transaction) mempool.
3. Bob retrieves the “release” transaction from mempool, obtains Alice’s release signature, and invokes the `executeSwap` function to acquire Alice’s assets on the source chain.
4. Malicious LP Bob then floods the destination chain with front-runned idle transactions, such that Alice’s “release” transaction gets postponed. (e.g. *Bob’s transactions offer a much higher gas price than Alice's “release” transaction, thus miners are more favored toward mining Bob’s transactions)*
5. Alice’s transaction remains pending until the lock period expires. Bob then invokes `unlock` to retrieve his locked assets.

Impact

A malicious LP can acquire initiator’s swap funds without exchanging their assets at the cost of losing a considerable amount of gas fee for delaying the victim initiator’s “release” transaction.

How likely is this attack going to happen? What is a swap amount that can be considered profitable for the attacker? We answer these questions with the following hypothetical scenario.

We suppose the attack happens on BSC and we roughly estimate the total gas fee based on a most recent BSC block. The block size is 45,940 bytes and fee is approximated as 0.15 BNB. Considering a full block occupying 2MB, its fee would be approximated as 6.85 BNB. We assume for front-running Bob doubles their transaction’s gas price, and the gas fee becomes 13.7 BNB per block. We further assume there are only Bob and Alice on the chain (This is the upper bound scenario, when there are more users present on the chain, the required gas fee to execute the attack would drop).

BSC generates blocks every 3 seconds. To make the lock expire, Bob needs to delay the “release” transaction by 20 minutes, which is roughly equivalent to 400 blocks. As a result, Bob theoretically needs to pay a gas fee of 5480 BNBs. In reality the gas price fluctuates block by block (e.g. the gas fee may increase exponentially on ETH London fork). The actual attack expense also depends on how Alice adjusts her transaction, e.g. resending the transaction with a higher gas price.

On a chain with longer block time, the gas price would become much more stable (since the chain is able to finalize only a few blocks during the attack), rendering the attack more feasible.

Overall, the actual attack expense is hard to estimate. *But the essence remains the same, that is, the more valuable the swap is, the more vulnerable it is.*

Recommandation

1) Limit the max amount of each individual swap (e.g. bound it to 100K).
2) Based on the amount of requested swap, dynamically adjust the lock time to render the attack infeasible. The algorithm should at least depend on the base gas fee and chain block time.
3) Warn users to check their “release” transaction’s status in time when requesting for large-sum swaps.
4) Encourage other users / “release miners” to mine the available releases from relayers/mempools and award them with Meson tokens. This enables us to execute multiple same “release” transactions on the chain thus making it harder for the malicious LP to execute the attack.

Late lock by LP allows initiator to steal LP's assets

Severity: **Medium** Difficulty: Medium
Type: Improper Locking
Target: cancelSwap (MesonSwap.sol), lock (MesonPool.sol)

Description

When LP locks their assets later than `expireTs` - `lockTime`, it is likely that the swap initiator can first invoke the `cancelSwap` function and then acquire LP's locked assets.

Impact

LPs may lose their assets without acquiring the initiator's swapped in funds.

Recommandation

Add the checking
`block.timestamp + LOCK_TIME_PERIOD < ((encodedSwap >> 48) & 0xFFFFFFFFFF)`
in the `lock` function.

Redundant LowGasSafeMath.sol

Severity: **Informational**
Target: MesonSwap.sol, MesonPool.sol

Difficulty: N/A
Type: Code With No Effects

Description

The MesonFi project is using solidity compiler v0.8.6. Solidity compiler beyond v0.8 automatically checks for integer overflow. As a result, usage of LowGasSafeMath.sol is redundant.

Impact

Nonoptimal code practices.

Recommendation

Remove LowGasSafeMath.sol and its corresponding function calls.

Hardcoded shifts are error prone

Severity: **Informational**
Target: MesonSwap.sol, MesonPools.sol

Difficulty: N/A
Type: Code With No Effects

Description

MesonFi uses many custom bit-packed structures to save gas. When acquiring specific field value, hardcoded shifting operations are used. This is very error prone and affects both the readability and maintainability of the code.

Impact

Nonoptimal code practices.

Recommendation

Add pack and unpack function for each field to replace shifting operations. Solidity compiler beyond v0.8.2 supports inlining short pure functions at no extra gas cost.

Malicious LPs/users can prevent swaps

Severity: **Low** Difficulty: Low
Target: MesonSwap.sol, MesonPools.sol
Type: Improper Following of Specification by Caller

Description

It is a natural drawback of the *atomic swap*, which the Meson contract built upon. After bonding to a swap, a malicious LP is able to block the normal use of the swap by refusing to lock their assets. The victim user has no choice but to wait until the swap expires. A similar attack can be launched by a malicious user who refuses to sign for release, rendering the victim LP's assets locked until the lock period expires.

Impact

Freeze users' and LPs' assets for `MIN_BOND_TIME_PERIOD` and `LOCK_TIME_PERIOD`, respectively.

Recommendation

Build off-chain monitoring/detecting system and support on-chain blacklist for dishonest LPs and users.

Malicious users can front-run to permanently occupy swaps

Severity: **Low**
Target: MesonSwaps.sol

Difficulty: Low
Type: Uncontrolled Resource Consumption

Description

A successful postSwap request requires `postedSwaps_[encodedSwap]` to be `0`. Upon executing `executeSwap`, if the `encodedSwap` is executed fast enough (i.e. `block.timestamp <= expireTS - MIN_BOND_TIME_PERIOD`) the code at line 120 will set the state to `1` to prevent replay attack. However, this state is then never reset to `0` again, effectively disallowing `encodedSwap` to be posted ever again.

```
113  if (((encodedSwap >> 48) & 0xFFFFFFFFFF) < block.timestamp +
MIN_BOND_TIME_PERIOD) {
114      // Swap expiredTs < current + MIN_BOND_TIME_PERIOD
115     // The swap cannot be posted again and therefore safe to remove it
116      // LPs who execute in this mode can save ~5000 gas
117      _postedSwaps[encodedSwap] = 0;
118    } else {
119      // 1 will prevent the same swap to be posted again
120      _postedSwaps[encodedSwap] = 1;
121  }
```

Possible Attack Scenario

1. Malicious user observes victim user's `postSwap` request in mempool and frontruns the victim user to execute their `postSwap` with the same `encodedSwap` along with their own signature before victim user's `postSwap` finalizes.
2. At this point the `encodedSwap` is already occupied. Malicious user can then invoke `executeSwap` with `recipientHash` pointing to itself to transfer the fund back and thus permanently occupy the `encodedSwap` in `_postedSwaps`

Impact

This enables a malicious user to front-run another user's transaction and make the Meson service unusable to them, effectively DoS another user at the cost of two transaction gas fees.

Recommendation

Clean _postedSwap states frequently when the condition at line 113 is met.

Fix Log

Table 6 Fix Log

ID	Title	Severity	Status
01	Incorrect typing while calculating lockedSwap	High	Confirmed and Fixed
02	Outdated Cryptographic Signature Used	Low	Risk Accepted
03	Arbitrary Bonding Unbonded Swaps	Medium	Confirmed and Fixed
04	Vulnerable high-volume swapping	High	Confirmed and Fixed
05	Late lock by LP allows initiator to steal LP's assets	Medium	Confirmed and Fixed
06	Redundant LowGasSafeMath.sol	Informational	Confirmed and Fixed
07	Hardcoded shifts are error prone	Informational	Confirmed and Fixed
08	Malicious LPs/users can prevent swaps	Low	Confirmed
09	Malicious users can front-run to permanently occupy swaps	Low	Confirmed

All the aforementioned fixes have been conducted and reviewed on commit hash **5beb15ae50fbcd2be29b2c67a4e4c39b439d4576** from the MesonFi/meson-contracts-solidity repository.

Appendix

A. Vulnerability Classifications

Table i Vulnerability Classifications

Type	CWE ID
Incorrect Calculation	CWE-682
Use of a Broken or Risky Cryptographic Algorithm	CWE-327
Improper Authentication	CWE-287
Incorrect Behavior Order	CWE-696
Improper Locking	CWE-667
Irrelevant Code	CWE-1164
Improper Following of Specification by Caller	CWE-573

Table ii Severity Categories

Severity	Description
Undetermined	The extent of the risk can not be determined during this assessment
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Low	The risk is relatively small or is not a risk that the customer indicated as important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possibly legal implication for client
High	Affects large number of users, very bad for clients reputation or posses great financial or legal risk

Table iii Difficulty Levels

Difficulty	Description
Undetermined	The difficulty of the exploit is undetermined during this assessment
Low	Commonly exploited, existing tools can be leveraged or can be easily scripted
Medium	Attacker must write a dedicated exploit, or need in depth knowledge of a complex system
High	The attacker must have privileged insider access or need to know extremely complicated technical details or must discover other issues to exploit this