From: n-var CONSULTING

To: AltLayer

Security Review

Mach AVS (M2) - MachOptimismServiceManager

Disclaimer

Executive Summary

- **Review Period**: 1 + 4 + 1 days (pre-review + review + report)
- **Start**: 02 Apr 2024
- **Delivery**: 08 Apr 2024

Timeline & Scope

- **First Scope**: [19 Feb 2024] https://github.com/alt-research/avs/blob/master/src/mach/MachOptimismServiceManager.sol

Initial feedback shared with the client:

serviceManager (ex groth16 proof verification):

- duplicate / mixed imports
- initialize dbl-transfers ownership
- owner can unilaterally change contract params (frontrun; set new image for proofs, clear contract state)
- array indexing is inconsistent
- anyone can submit alerts (any operator, but anyone can register IIRC)
- alerts stored in array may be problematic
- checks for things that cannot happen (proverIndex > arrLength)
- a potential gas oob when iterating array length
- a potential frontrunning attack on alerts (i.e. malicious op frontrunning a legitimate alert witha bogus one)
- checks could be optimized

- **Actual Scope**: [02 Apr 2024] https://github.com/alt-research/mach-avs/blob/m2-dev/contracts/src/core/MachServiceManager.sol (Commit hash: `3c120d28425b3f03dd55cd03cda259829ccbbc7b`)

This iteration addressed findings shared with the client when initially scoping the engagement. Complexity much reduced. Code quality increased. Feedback addressed.

- **Remediation** [07 Apr 2024] Updated the initial report with remediation notes and relevant fixes.

Files in Scope:

- `MachServiceManager.sol`

Findings

Severity Medium

[MEDIUM] Missing input sanitization in `updateQuorumThresholdPercentage()`

Remediation Note: Addressed in https://github.com/alt-research/mach-avs/pull/101 by raising an alert if `> 100%`

The function `updateQuorumThresholdPercentage()` does not have any checks on the input parameter `thresholdPercentage`. The expected range of an `uint8` value is 0 to 255, which means that the `quorumThresholdPercentage` value could theoretically be set to any value within this range. However, as this value is used as a percentage, acceptable values should be limited to the range of 0 to 100.

If no checks are implemented, potential issues could occur. For instance, if the `thresholdPercentage` is set above 100, it might break the functions that use `quorumThresholdPercentage` as they might never reach the required quorum.

contracts/src/core/MachServiceManager.sol (Lines 147-150):

```
function updateQuorumThresholdPercentage(uint8 thresholdPercentage)
external onlyOwner {
    quorumThresholdPercentage = thresholdPercentage;
    emit QuorumThresholdPercentageChanged(thresholdPercentage);
}
```

Valid ranges are 0-100 as implied by this comment on the individual threshold percentages:

contracts/src/core/MachServiceManager.sol (Lines 222-224):

```
for (uint256 i = 0; i < alertHeader.quorumThresholdPercentages.length; i++)
{
    // we don't check that the quorumThresholdPercentages are not >100
because a greater value would trivially fail the check, implying
    // signed stake > total stake
```

It is recommended, to implement a sanity check to verify that the `thresholdPercentage` is within the valid range (0 to 100). One could even further restrict the range for better security. If this condition is not met, the function should revert the operation.

[MEDIUM] `removeAlert()` succeeds for invalid messageHashes and `enableAllowlist()`, `disableAllowlist()` should not emit events if the state is left unchanged

Remediation Note: Addressed with https://github.com/alt-research/mach-avs/pull/106 by checking if the alert was indeed removed and raising an error if not, and https://github.com/alt-research/mach-avs/pull/108 reverting on ineffective calls.

The `removeAlert()` function doesn't check whether `messageHash` exists before trying to remove it (i.e. by checking the `bool` return value of `Set.remove()`). Thus, the function will emit an 'AlertRemoved' event even if the `messageHash` didn't exist in the first place, or was already deleted.

Similarly, but less of a security problem, the function `enableAllowlist()` emits events if `allowListEnabled` is already set to `true`. Respectively,`disableAllowlist()` emits events if `allowListEnabled` is already `false`. This could be confusing and impact off-chain components as the state of `allowlistEnabled` will not change, yet the functions will still emit events.

It is therefore, recommended, that function `removeAlert()` checks the result of the the call to `_remove(messageHash)` and revert or skip emitting the event if it returns `false`.

[MEDIUM] `confirmAlert` - Inconsistent `referenceBlockNumber` Staleness Check

Remediation Note: Addressed with https://github.com/alt-research/mach-avs/pull/91.

When making sure that the stakes being checked against with are not stale, `confirmAlert` checks that `alertHeader.referenceBlockNumber` is not in the future, and at max the current `block.number`. However, the current block can already be considered in the future because this transaction is part of it. Alerts observed by outsiders and signed upon are very unlikely to be submitted for the current "in the making" block.

This can also be seen in the Documentation: BLSSignatureChecker.md where it is suggested that `referenceBlockNumber` must be less than `block.number`. Here's the relevant code part in `BLSSignatureChecker`:

https://github.com/Layr-Labs/eigenlayer-middleware/blob/dc9d01307a4e98d5d2529bd10ee38682c97e465c/src/BLSSignatureChecker.sol#L115

Since the signature check would fail with `referenceBlockNumber < uint32(block.number` it is suggested to adjust the check in `confirmAlert` to be the same as `BLSSignatureChecker`. Relevant code:

contracts/src/core/MachServiceManager.sol (Lines 207-211):

```
// make sure the stakes against which the Batch is being confirmed are not
stale
if (alertHeader.referenceBlockNumber > block.number) {
    revert InvalidReferenceBlockNum();
}
bytes32 hashedHeader = hashAlertHeader(alertHeader);
```

[MEDIUM] `confirmAlert` - Missing Array Length Check for `quorumNumbers`, `quorumThresholdPercentage`

Remediation Note: Addressed with https://github.com/alt-research/mach-avs/pull/88 by checking the array lengths.

`confirmAlert` is missing a check to enforce that `quorumNumbers.length == quorumThresholdPercentage.length`. The two arrays are connected as the threshold percent of quorumNumber[x] is at position quorumThresholdPercentage[x]. Note that without this check the two arrays would not be coupled at all. With this check, they are at least loosely coupled in length.

contracts/src/core/MachServiceManager.sol (Lines 199-228):

```
function confirmAlert(
    AlertHeader calldata alertHeader,
    NonSignerStakesAndSignature memory nonSignerStakesAndSignature
) external whenNotPaused onlyAlertConfirmer {
    // make sure the information needed to derive the non-signers and batch
is in calldata to avoid emitting events
    if (tx.origin != msg.sender) {
        revert InvalidSender();
    }
    // make sure the stakes against which the Batch is being confirmed are
not stale
    if (alertHeader.referenceBlockNumber > block.number) {
        revert InvalidReferenceBlockNum();
    }
    bytes32 hashedHeader = hashAlertHeader(alertHeader);

    // check the signature
    (QuorumStakeTotals memory quorumStakeTotals, bytes32
signatoryRecordHash) = checkSignatures(
        hashedHeader,
        alertHeader.quorumNumbers, // use list of uint8s instead of uint256
bitmap to not iterate 256 times
        alertHeader.referenceBlockNumber,
        nonSignerStakesAndSignature
    );

    // check that signatories own at least a threshold percentage of each
quourm
    for (uint256 i = 0; i < alertHeader.quorumThresholdPercentages.length;
i++) {
        // we don't check that the quorumThresholdPercentages are not >100
because a greater value would trivially fail the check, implying
        // signed stake > total stake
        // signedStakeForQuorum[i] / totalStakeForQuorum[i] *
THRESHOLD_DENOMINATOR >= quorumThresholdPercentages[i]
        // => signedStakeForQuorum[i] * THRESHOLD_DENOMINATOR >=
totalStakeForQuorum[i] * quorumThresholdPercentages[i]
        uint8 currentQuorumThresholdPercentages =
uint8(alertHeader.quorumThresholdPercentages[i]);
        if (currentQuorumThresholdPercentages < quorumThresholdPercentage)
{
```

[MEDIUM] `EnumerableSet` - Unchecked Return Values on `add/remove`

Remediation Note: Addressed with https://github.com/alt-research/mach-avs/pull/89 by checking for duplicate add on `confirmAlert` and adding more comments.

EnumerableSet.sol returns `false` on error. For example, if an existing element is added to the set or a non-existing element is removed, the function returns `false`.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/structs/EnumerableSet.sol#L169-L171

- In the case of the `confirmAlert()` this may allow the `alertConfirmer` role to add the same message, multiple times, as the return value of `add()` is not checked and its existence is therefore silently ignored. This will result in an event emission even though that event has fired before.

contracts/src/core/MachServiceManager.sol (Lines 240-241):

```
_messageHashes.add(alertHeader.messageHash);
```

- In the case of `registerOperatorToAVS()` it is even fine not to check the return value because the `AVSDirectory` will revert on duplicates. Nevertheless, it is recommended to add an inline comment that the unchecked return value is intentional.

contracts/src/core/MachServiceManager.sol (Lines 170-171):

```
_operators.add(operator);
emit OperatorAdded(operator);
```

Leave an inline comment making it obvious that the absence of return value checks are intended. For the `confirmAlert` case, it is recommended to revert if the element already exists in the set.

Severity Low

[LOW] `initialize` should call `__ServiceManagerBase_init(initialOwner)` instead of `transferOwnership`

Remediation Note: Addressed in https://github.com/alt-research/mach-avs/pull/83

`ServiceManagerBase` exposes an initialization chain. Use the `init` function to initialize `Base` for maximum compatibility.

src/ServiceManagerBase.sol (Lines 47-49):

```
function __ServiceManagerBase_init(address initialOwner) internal virtual
onlyInitializing {
    _transferOwnership(initialOwner);
}
```

```
function __ServiceManagerBase_init(address initialOwner) internal virtual
onlyInitializing {
    _transferOwnership(initialOwner);
}
```

[LOW] Lock Solidity to a recent Version

Remediation Note: Addressed in https://github.com/alt-research/mach-avs/pull/107

Contracts should be deployed with the same compiler version they have been tested with. Locking the pragma helps ensure that contracts do not accidentally get deployed and compiled using a different compiler version. Additionally, it indicates the compiler version that has been used to develop and test the contract when open-sourcing the code.

contracts/src/core/MachServiceManager.sol (Lines 9-9):

```
pragma solidity ^0.8.12;
```

It is recommended, to lock the Solidity version in the pragma declaration. Ensure you're using a recent version.

[LOW] Alerts Are Removed Completely Instead Of Them Being Invalidated (Audit Trail)

Remediation Note: Addressed with https://github.com/alt-research/mach-avs/pull/106

When removing alerts they are removed from the contract state completely. It would potentially make sense to keep that historic information instead and flag it as invalid only (audit trail), unless there is no need for that and all traces of the alert hash should be removed completely as per the system design.

For clarification, the client provided the following information:

> I think we prefer the second one. Once the alert is handled, should be no more useful. And we can always check event log if we want trace some specific log happens previously

It should be noted, that if alerts are removed from the contract, a corrupted/malfunctioning `alertObserver` could replay the aggregated signature which would still be valid, which would record the alert in the contract again, and re-emit an `AlertConfirmed()` event. This could potentially impact off-chain components.

The client provided the following statement with the remediation:

> The removeAlert() function doesn't check whether messageHash exists before trying to remove it (i.e. via checking the bool return value of Set.remove(). Thus, the function will emit an 'AlertRemoved' event even if the messageHash didn't exist in the first place, or was already deleted.

[LOW] Operators cannot be enumerated or queried

Remediation Note: Acknowledged.

Operators can register and deregister with the AVS. They are stored in a complex `EnumerableSet`. However, there is no functionality to access them. If there are no plans to add an enumeration of operators in the future, it might make sense to fall back to a normal `mapping address => bool` to track operators.

The client provided the following statement:

> Operators can be queried from EigenLayer middleware directly

Severity Info

[INFO] Check-Effect-Interaction violation - `registerOperatorToAVS()`

Remediation Note: Addressed in https://github.com/alt-research/mach-avs/pull/105.

The function`registerOperatorToAVS()` violates the Check-Effect-Interaction pattern. It would be preferable to first add the operator to the `_operators` set, and then call the external `_avsDirectory` contract even if it is trusted. There is no security impact in this case as `add` will silently add the operator to the set anyway. We are mentioning this at this point to foster a secure coding style.

[INFO] The `alertConfirmer` address cannot be changed

Remediation Note: Addressed with https://github.com/alt-research/mach-avs/pull/100 (note: centralization)

The `alertConfirmer` address cannot be changed as `_setAlertConfirmer()` is declared as internal, and there is no corresponding external function declaration. Thus, if the `alertConfirmer` account is compromised, there is no way to change it other than performing a contract upgrade.

[INFO] `hashAlertHeader()` and `convertAlertHeaderToReducedAlertHeader` should take a `calldata` argument

Remediation Note: Addressed with https://github.com/alt-research/mach-avs/pull/103

To optimize gas usage, it would be preferable to pass the `AlertHeader` argument as `calldata` instead of `memory` to the functions `hashAlertHeader()` and `convertAlertHeaderToReducedAlertHeader`.

[INFO] Style: Underscore Prefix for Non-external Functions

Remediation Note: Addressed with https://github.com/alt-research/mach-avs/pull/104

By convention, names of non-external Functions in Solidity should be prefixed with an _. Thus, to follow Solidity's Style Guide, it would be preferable to rename `convertAlertHeaderToReducedAlertHeader()` and `hashAlertHeader` to `_convertAlertHeaderToReducedAlertHeader()` and `_hashAlertHeader` respectively.

[INFO] Information About The Systems' Centralization

Remediation Note: Acknowledged by the client.

There's a slight centralization concern because `onlyOwner` can gate-keep (via allowList), tune parameters (`quorumThreshold`), and remove alerts the group of operators had signed up on, unilaterally. It should also be noted that the contracts are proxies and the proxy owner may initiate upgrades without prior notification.

Right now, for example, the `owner` can front-run sandwich `confirmAlert` setting `quorumThresholdPercentage` in a way it is impossible to pass the signature check, censoring operators trying to submit an alert.

This can easily be tackled by requiring the `owner` (and proxy owners, too) to be a democratic MultiSig or DAO that requires (a) announcement and vote delay (b) sufficient quorum. For `removeAlert` it should be considered to use the same consensus mechanism used in `confirmAlert` and allow the operators to remove invalid message hashes when they find consensus on it.

First and foremost, it should be the project's highest priority to clearly communicate the degree of centralization they want to have, communicating who can do what, if delays are required or if things can be changed unilaterally. It is up the the user-base to decide if they want to participate in a system based on the description of the degree of centralization.

The client provided the following clarification:

> There is some degree of centralization as we regard them as training wheels, which can be removed in future contract upgrade
>
> Alerts should be extremely rare, and if they happen, the chain we are securing will very likely need to be rolled back. In such a case, we expect the faulty chain to be first rolled back with manual inspection, followed by the contract owner manually removing the alert.
>
> Removing the alert will emit AlertRemoved and this should cause a concern if an alert was maliciously removed.
>
> We will also clearly document the privileges of privileged parties in our public documentation, e.g., gitbook, when it goes live.

[INFO] Information about Safe Deployment to prevent init front-running

Remediation Note: Acknowledged by the client.

For completeness, we notified the client that their contract must be deployed and initialized in the same transaction. The client confirmed that `MachServiceManagerDeployer.s.sol` is their deployment contract and deployment of the implementation (petrified) and proxy with `upgradeTo` and `initialize()` are performed in a safe way, in the same transaction.

[INFO] Information about Out-of-order Alert Submission

Remediation Note: Acknowledged by the client.

`confirmAlert` does not enforce that alerts are being submitted in order. The client provided the following clarification:

Out-of-order Alert Submission can be accepted so as long as it has reached the required quorum threshold. An out-of-order alert is still a valid alert and should be flagged

[INFO] Information about Alert Data Validation being Completely Off-Chain; Re-Orgs

Remediation Note: Acknowledged by the client.

We would like to note that `confirmAlert` does not validate `messageHash` at all. This `messageHash` hashes information about an alert on the L2/L3 chain. Responsibilities for validation are completely off-chain and up to the operators to find a consensus upon and sign it. Note that L2/L3 can re-org and operators might sign alerts for blocks that got invalidated. We assume it is the off-chain components' responsibility to deal with events like this. On the flipside, L1 might reorg too, which could impair off-chain components if they are reliant on these alerts. This issue arises because alerts can be submitted up to the current block, implying that alerts can be submitted for blocks that have not been confirmed yet. For a block that is ultimately not part of the finalized chain, these alerts will be lost.

The client provided the following statement:

Right, so the aggregator/confirmer who collects BLS signature from operators shall perform the validation. The honest operator should do that too. The validation not happen on contract side yet (for this version), it only relying on BLS signature and stake threshold

It is possible that L2/L3 can re-org and the invalid block can be invalidated due to re-org and the valid block become eventually finalised.
In such a case, if operators still manage to come to a consensus on the invalid block and the aggregator submits the signature\ on-chain, the alert will be for a block that has been re-org and off-chain services listening on the contract will need to handle this case.

[INFO] Unused Import

Remediation Note: Addressed with https://github.com/alt-research/mach-avs/pull/87 by removing the import.

The following Symbols are imported but not used within the Source Unit:

contracts/src/core/MachServiceManager.sol (Lines 32-32):

`NotAllowed`