Immunefi

Internal Audit Report

Vault System and Arbitration MVP

Jun 6, 2023

This report represents the results of an internal audit report that our team of triagers have conducted on Immunefi's Vault System and Arbitration MVP.

Technical Overview for the Audit

Code repository: https://github.com/immunefi-team/vaults-internal.
Commit for audit: 466c7f55f9ef220d60c5a21807936db55008f37f.
SLOC: ~1500 (excludes comments and empty lines).
In scope: all files inside the *src* folder, excluding the *mocks* folder.

Protocol Assumptions

- The *ARBITER_ROLE* role is granted to a single highly trusted third party, some entity specially selected by Immunefi and completely assumed to be neutral and collusion-resistant.
- There is an assumption whereby a *referenceId* in the context of the Arbitration component is corresponding to a specific bug report. A corollary is that the *referenceId* will be unique, and only one arbitration per *referenceId* can be called. Because the on-chain components don't have knowledge about which *referenceId*s correspond to valid open bug reports, we need a trusted entity to provide proof of it. This is the reason why there's an *inputValidationSigner*, an account responsible for signing the transaction input to validate the usage of the specific *referenceId* with a specific *whitehat*, etc. The *inputValidationSigner* account would essentially be a key controlled by the Immunefi dashboard. An effect of this is that arbitration can only be called through the Immunefi dashboard, in order to get that necessary *inputValidationSignature*.
- The architecture absolutely provides a great deal of control to Immunefi, and particularly to the *ProxyAdminOwnable2Step* owner. Immunefi is capable of changing the logic of *EXECUTOR_ROLE* contracts, thus it is capable of executing arbitrary operations on the protocol's Vault. Roles are also granted by an owner, so Immunefi could whitelist any

address as an EXECUTOR to run arbitrary operations through the ImmunefiModule, namely draining the contract.

Findings

Title: Arbitration fee not included in hashed message blob

Severity: Low
Line and commit hash:
https://github.com/immunefi-team/vaults-internal/blob/4eee8cd417f9bfcb12c96e3008967cde6056b9ab/src/Arbitration.sol#L110
Description: The `feeAmount` the whitehat is willing to pay is not included in the message blob. A whitehat's signature for arbitration could be held for a `feeAmount` they did not agree to.
Recommendation: Include arbitration fee in the signed message blob.
Acknowledgement: Acknowledged. We do transferFrom from the caller of the function, not from the whitehat, so the whitehat only signs to allow the calling of arbitration. The caller can either be the whitehat or another address (potentially a sponsor), which will cover for the fees. There is no permit, so the feeAmount needs to be previously approved in the feeToken.

Title: Tokens with fees are not supported

Severity: Informational
Line and commit hash:
https://github.com/immunefi-team/vaults-internal/blob/4eee8cd417f9bfcb12c96e3008967cde6056b9ab/src/Arbitration.sol#L234
Description: Tokens with transfer fees are not supported by arbitration since the value sent may deviate from tokens actually received.
Recommendation: Explicitly define what types of tokens are not supported, or take into account fees for transferred tokens.
Acknowledgement: Acknowledged, we will not take into account fees for transferred tokens.

Title: Centralization risk, arbiter and input validation signer

Severity: Informational
Line and commit hash:
https://github.com/immunefi-team/vaults-internal/blob/4eee8cd417f9bfcb12c96e3008967cde6056b9ab/src/Arbitration.sol
Description:

Arbiters have the power to close arbitration requests with arbitrary rewards to be enforced and sent out from vaults. This gives arbiters direct access to project vault funds with no limits. A compromise which results in the input validation signer and arbiter's private keys exposed would be catastrophic as all vaults can be drained.
Recommendation: Ensure these roles are given to separate trusted actors and key are managed individually. Consider assigning arbiters to individual cases or limiting the payout amount an arbiter can assign.
Acknowledgement: Acknowledged, it is 100% a big centralization factor, specially the input validation signer key (which can in theory also be a multisig for increased security). Still, this is the solution adopted for the MVP. We will hopefully reach a more matured and decentralized solution in future iterations of the Arbitration component.

Title: Missing token retrieval in arbitration escrow

Severity: Informational
Line and commit hash:
https://github.com/immunefi-team/vaults-internal/blob/4eee8cd417f9bfcb12c96e3008967cde6056b9ab/src/ArbitrationEscrow.sol
Description: Funds that accidentally send to the escrow contract, will be stuck in the arbitration escrow contract and cannot be retrieved, Or if a whitehat didn't want to go through arbitration, and the vault already initiate the arbitration through `requestArbVault()`, the vault funds will be stuck in the escrow, and the arbiter can't retrieve it too, because the status of the arbitration still in the `WaitingForWhitehat`.
Recommendation: Add token retrieval logic
Acknowledgement: Acknowledged. We will not create withdrawal logic for any accidental tokens getting sent to the escrow. The same goes for any other contract in the protocol, since in theory any contract can be sent accidental tokens. As for the permanent lock of vault funds, it is an issue for sure and we are aware of this. In this iteration, we are working with the assumption that the whitehat will agree to call for mediation, since design decisions were not yet done regarding an arbitration expiration time. But this will need to be resolved in the next protocol iteration.

Title: whitehat can DOS vault funds, if the vault is the one that initiates the arbitration through `requestArbVault()`

Severity: Informational
Line and commit hash:
https://github.com/immunefi-team/vaults-internal/blob/4eee8cd417f9bfcb12c96e3008967cde6056b9ab/src/Arbitration.sol#L200

Description: When Vault forward their funds to the escrow contract through `requestArbVault()`, a whitehat or other user can call `openVaultRequestedArb()` to open the arbitration status. However, the `IsValidSignatureNow()` will make an external `isValidSignature()` static call to the first arg of the function, which gives control to the whitehat whether to pass the failed signature or not.

Attacker contract:

```
pragma solidity =0.8.7;

interface IERC1271 {
	function isValidSignature(bytes32 hash, bytes memory signature) external view returns
(bytes4 magicValue);
}

contract test3 {

	bool Passed;

function isValidSignature(bytes32 hash, bytes memory signature) external view returns(bytes4){
	if (Passed) {
	return (IERC1271.isValidSignature.selector);
	}
	require(false, "Intentionally revert");
	}

function setPassed(bool input)external {
	Passed = input;
}
}
```

Recommendation: check if the whitehat is a contract or not.

Additional note: If the vault can request an arbitrary reference Id with its valid `inputValidationSig`, the vault can DOS the arbitration contract by continuously calling `requestArbWhitehat()`, with a whitehat arg set as an attacker contract. Which only cost n * feeAmount, n as in the total report, Because the funds only being forwarded to the escrow contract through the `safeTransferFrom()`. (Assuming a vault can be any contract)

Acknowledgement: Acknowledged. EIP1271 is only checked if the whitehat is a contract, which is already done by the SignatureCheckerUpgradeable. And the whitehat absolutely has the power, if it is a contract, to mark the signature as invalid. It's the equivalent of the whitehat continuously DoSing itself. Not a bug.