Internal Audit Report

Splitter.sol contract for the new Vault Systems

Feb 3, 2023

This report represents the results of an internal audit report that our team of triagers have conducted on the new Vault System, specifically the Splitter.sol contract, that Immunefi is launching for their customers.

[Informational] Natspec comments do not correspond to the code logic

One the line 80 of Splitter.sol the natspec comment mentions “@dev Only callable by owner” whereas the function is not onlyOwner . Please update the natspec comments to reflect the logic of the code.

Immunefi response:
`Fixed by the team`

[Low to Medium] Potential drain of leftover funds from the Splitter contract if someone sends more native tokens through payment to the whitehat during payWhitehat()

payWhitehat() function is payable, and it's expected that the function caller will send native currency through *msg.value* and split it between the whitehat and the fee receiver. However, there's no usage of msg.value. As a result of that, callers can send more funds on *msg.value* than *nativeTokenAmt+fee*. This will make it probable that the contract will be holding funds in the native currency, and those funds can be drained by anybody by calling *payWhitehat(randomNumber, attackerAddress, emptyList, someAvailableBalanceAmount, someGas)*. This problem is not present on the *ERC20* part of the function, because funds go directly from the caller to the wh and fee receiver.

Immunefi response:
`Fixed by the team`

[Low, low likelihood of happening] Possibility of paying lower Immunefi fee once when fee increases in the future.

There's a front-running possibility to pay lower fees IF Immunefi decides to increase the fee from 10% to a higher number. Projects wanting to pay lower fees through our system could frontrun the *setFee()* function and call payWhitehat before *setFee()* will be executed. It's worth noting this scenario is very unlikely to happen.

Immunefi response:
`Fixed by the team`

[Informational] gas should be capped

The gas parameter should be capped, to prevent the whitehat consuming all the gas when receiving the native token in case the project mistakenly set the gas parameter to a big amount. (consuming without reverting the transaction, can be done with manually setting up a return data in the receive function, using `return` opcode)

Immunefi response:
Acknowledged.