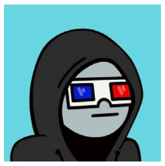

By
0xth0mas

Contract Audit
Issue Date
11/01/2023

Overview

The following is a review of the GasliteSplitter smart contract, performed by 0xth0mas. GasliteSplitter is a gas efficient native and ERC20 token splitter that distributes tokens sent to the contract to recipients specified at the time of the contract's deployment.

The purpose of this audit is to identify security vulnerabilities, potential gas savings, general best practice recommendations, as well as offer input from a design perspective to minimize community trust requirements.

The audit is based on the GitHub Gist `3873bb951919931832189bc9d383cad0`.

This audit includes a best effort review of potential vulnerabilities, but it is not a guarantee of security. Findings should be used in conjunction with other audits (internal or external), as well as independent verification.

Lastly, this audit is simply a technical review and not an endorsement or dismissal of GasliteSplitter.

Design

The user deploying a GasliteSplitter contract specifies the recipients, number of shares each recipient receives, and a flag specifying if the person calling the release functions receives a 0.1% royalty payment as a reward for executing the transaction. Recipient shares are summed up to get a total share count that is used for releases.

Recipients, shares, and the release royalty flag cannot be modified after deployment.

When the release function is called for either native tokens or ERC20 tokens the contract checks the token balance, subtracts and sends 0.1% of the balance to the caller if the release royalty flag is set to true, sends tokens to each recipient using the formula payment = (balance * shares) / totalShares, and emits a SplitReleased event with the recipient addresses and amounts sent.

Findings

#1 accumulatedShares can overflow at contract deployment

Severity: Low

At contract deployment each recipient's share count is summed up to a total number of shares. The summation occurs inside an assembly block that does not check for overflow.

Impact – Deployers that set share values near the maximum value of uint256 for one or more recipients can overflow the accumulatedShares variable which is assigned to totalShares. This can result in unexpected behavior and release transactions to fail.

Suggested Resolution – Set a maximum value for each recipient's share amount to an amount that would not allow the accumulatedShares amount to overflow.

#2 totalShares can be zero

Severity: Low + Gas Optimization

There is no check that any recipient has a share count greater than zero.

Impact – If totalShares is zero the only tokens that could be dispersed from the contract would be the 0.1% royalty if that flag is set. Additionally, any recipient with a zero share count incurs unnecessary gas costs on deployment and each release despite having no shares.

Suggested Resolution – Add a check that each recipient has a share amount greater than zero.

#3 releaseRoyalty is a storage variable

Severity: Gas Optimization

The releaseRoyalty flag cannot be changed after contract deployment and should be set as an immutable variable.

Impact – Additional gas cost of 20,000 gas units for the SSTORE at time of deployment, 2,100 gas units on each release to SLOAD the flag from storage.

Suggested Resolution – Change releaseRoyalty from a storage variable to an immutable variable.

#4 Recipient and share amount are stored in two separate storage slots

Severity: Gas Optimization

Storage slots are 256 bits and an address only uses 160 bits, leaving 96 bits available for other data. A recipient's share amount could be limited to 96 bits and packed with the recipient address to save gas on deployment and each release.

Impact – Additional gas cost of 20,000 gas units per recipient to SSTORE in two slots instead of one at time of deployment, 2,100 gas units per recipient to SLOAD from two slots instead of one on each release.

Suggested Resolution – Pack recipient share with recipient address.

#5 Shares copied to memory for each release

Severity: Gas Optimization

On each release the shares data is copied from storage to memory incurring a cost for memory expansion, writing the data to memory and then reading the data from memory.

Impact – Additional gas costs that will vary depending on the number of recipients.

Suggested Resolution – Load the share amount from storage onto the stack to perform the calculation for a recipient's amount.

#6 Two for loops iterating over shares and recipients for each release

Severity: Gas Optimization

The release functions iterate over shares to calculate amounts then iterate over addresses to send the amounts to each address.

Impact – Unnecessary gas cost for second loop iteration.

Suggested Resolution – Send the payment to the recipient in the first loops when their token amount is calculated.

#7 Solidity has to recopy recipient and amount memory arrays to emit SplitReleased log

Severity: Gas Optimization

Solidity ABI encodes the recipient and amount memory arrays to emit the SplitReleased log which requires them to be copied to a new area of free memory to add the pointer for each array as the first two values.

Impact – Additional gas costs at deployment for the runtime bytecode to copy arrays, additional gas fees at each release for copying arrays.

Suggested Resolution – Layout the recipient addresses and amounts in memory with the ABI encoding and emit the event with an assembly log function.

#8 Event emission on receive function

Severity: Gas Optimization

The receive function emits a PaymentReceived event on each native token transfer to the splitter. It is questionable whether this is necessary.

Impact – Additional gas fees for payments being sent to the splitter contract.

Suggested Resolution – Consider removing the event.

#9 SplitReleased event does not include token address for ERC20 releases

Severity: Informational

The SplitReleased event includes recipients and amounts but there is no indication if it is a native token release or an ERC20 token being released.

Impact – A user retrieving SplitReleased logs would need to pull additional transaction information to determine what was released.

Suggested Resolution – Add the token address for ERC20 releases.

Conclusion

GasliteSplitter was evaluated for potential reentrancy attacks and other exploits that could allow a user to obtain more value than their share amount. Caching the contract balance and reverting if the token transfers fail guards against reentrancy and ensures that recipients only receive their share of tokens.

Findings #1 and #2 could allow a deployer to create a contract that locks funds, these were deemed low risk as the deployers and recipients of a deployed splitter should be checking the shares / totalShares amounts prior to using the splitter to receive funds.