HashCloak

Security assessment and code review

Delivery: March 29, 2022 and August 29, 2022

Prepared for:
Railgun

Prepared by:
Mikerah Quintyne-Collins | HashCloak Inc

Table Of Contents

Executive Summary

Railgun is a privacy protocol that aims to provide Ethereum users with a cheap way to make private transfers and swaps. HashCloak had been engaged by Railgun to look at its v2 circuits and at its voting and treasury Solidity smart contracts. From March 22, 2022 to March 29, 2022, the HashCloak team reviewed the Railgun v2 circuits. From August 22, 2022 to August 29, 2022, HashCloak reviewed the Voting.sol and

Treasury.sol contracts. The review was done over 2 person weeks with 1 auditor. The scope of the audit was the following :

- https://github.com/Railgun-Privacy/circuits-v2/tree/main/src at commit 67cd4ce7f49afd1dfae67c8e2d59ddf87ec3de43
- https://github.com/Railgun-Privacy/contract/blob/main/contracts/treasury/GovernorRewards.sol and https://github.com/Railgun-Privacy/contract/blob/main/contracts/governance/Voting.sol at commit b74eeb69ca2614212c8060a3460fd05c28bb17e3

We found 7 issues which range from High to informational. The Railgun team has promptly resolved all the issues found during the security review.

Severity	Number of Findings
Critical	0
High	3
Medium	0
Low	1
Informational	3

Summary of Findings

Unused variables in `joinsplit.circom`

On lines 64 and 68 both have unused variables that are not used within `joinsplit.circom`. Instead the corresponding variables have been refactored into `public-input-hash.circom`

Missing `nullifierCheck.random` assignment in `joinsplit.circom`

On line 84 , a `nullifierCheck` component is initialized but the `random` input signal is not set. The following snippet rectifies this by setting the random input signal:

```
component inExtractors[nInputs];
for (i=0; i<nInputs; i++) {
     inExtractors[i].in <== packedIn[i]
}
// Since in nullifiersCheck, random is the same across hashes
// should suffice to simply any of the inExtractors for the given
input note
nullifiersCheck.random <== inExtractors[0].random
```

Simplification of `PublicInputHash.circom` is possible

The circuit PublicInputHash.circom can be simplified as follows:

```
var size = nInputs + nOutputs + 2;
component poseidon = Poseidon(size);
poseidon.inputs[0] <== merkleRoot;
poseidon.inputs[1] <== boundParamsHash;


for(var i=2; i<nInputs; i++) {
    poseidon.inputs[i] <== nullifiers[i-2];
}

for(var i=nInputs; i<nOutputs; i++) {
    poseidon.inputs[i] <==
commitmentsOut[i-nInputs];
}
```

SetVotingKey in Voting.sol can be set by anyone

`SetVotingKey` in Voting.sol is meant to add a voting key to the Voting contract in order to allow anyone with a voting key to participate in governance. However, in its current implementation, anyone can add their own voting key. Hence, a malicious entity can create many fake sybil voting keys in order to pass/reject any proposal they want.
Recommendation: Limit who can call the `SetVotingKey` function to an admin or those that have staked in Railgun.

CallVote in Voting.sol can be called by anyone

`CallVote` sets in motion the voting period for a proposal. However, currently, anyone can call `CallVote` at any time that they want. Combined with the previous issue, a malicious entity can pass through proposals as quickly as they want.
Recommendation: Limit who can call the `CallVote` function to an admin

ExecuteProposal in Voting.sol can be called by anyone

`executeProposal` executes a proposal once it has been voted on. However, anyone can call this function. Combined with the previous two issues, a malicious entity can execute any proposal they want
Recommendation: Limit who can call the `executeProposal` to an admin.

Votes can be front-run

Votes are all publicly executed on Ethereum. As such, front-runners can front-run these votes as they see fit.
Recommendation: Employ a commit-reveal scheme in order to minimize the possibility of votes being front-ran.