Introduction

A security review of the IdentityStaking V2 contracts for Gitcoin, done by Backseats focusing on the gas and security aspects of their smart contract.

Disclaimer

This security review is not a guarantee of a lack of vulnerabilities. It is my best effort within the time spent to identify, reproduce, and report errors and improvements that can be made to the project or protocol. As always, the more eyes on the project the better so I encourage others to give this protocol a security review as well.

About Backseats

Backseats is the founder of Backseats Studio and Solidity developer. We help people build and launch products and protocols. He is also the founder of ContractReader.io, a site to read and understand smart contracts that includes live data right inline with the code to give you a better picture of what's happening onchain. You can reach out to him on Twitter @backseats.

About IdentityStaking

`IdentityStaking` is a protocol developed by Gitcoin that allows the staking of Gitcoin (GTC) on one's own address or on someone else's. It is a successor to their current `IdentityStaking` protocol though wholly independent. It does not rely on the prior contracts and introduces the idea of slashing should someone commit a slashable offense. Slashes can be appealed within a minimum period of 90 days upon which time the outcome can be the GTC is 1) held by the contract 2) burned or 3) returned to the original owner.

You can find more, directly from them, on their README.

Observations

`IdentityStaking` is built on top of well-work OpenZeppelin contracts: `UUPSUpgradeable`, `Initializable`, `AccessControlUpgradeable`, and `PauseUpgradable`. It has no additional dependencies other than the `IERC20` interface for interacting with Gitcoin.

It will be important when considering upgrading the contract that storage variables remain in the same order and with the same names so that the proxy upgrade can happen effectively while not losing any storage data from the previous version(s).

Finally, `IdentityStaking` emits a myriad of events that allow for their team or interested observers to pay attention to the changes in the protocol to monitor it accordingly.

Security Roles and Actors

`IdentityStaking` employs three key roles which are managed well: `SLASHER_ROLE`, `RELEASER_ROLE`, and `PAUSER_ROLE`. The contract does not mix and match roles nor do roles subsume one another. The use of the `onlyRole` modifier is clear and explicit and prevents certain roles from taking unnecessary action or holding too much power.

The `lockAndBurn` function can be called by anyone to keep the protocol in check.

Severity Classification

- Critical – Leading to significant financial loss and reputational risk if attacked
- High – A very bad but not entirely devastating vector of attack
- Medium – Significant but again, not devastating
- Low – Minimal damage to be done by exploiting

Scope

The `IdentityStaking.sol` file was in scope of this audit.

`UUPSUpgradeable.sol`, `Initializable.sol`, `AccessControlUpgradeable.sol`, and `PauseUpgradable.sol` were not in scope as they've been rigorously audited by OpenZeppelin and the Solidity community.

Findings

The following issues were found, categorized by severity:

- Critical: 0 issues
- High: 0 issues
- Medium: 0 issues
- Low: 2 issues

ID	Title	Severity	Status
[L-01]	Ensure the validity of parameters of functions	Low	Fixed
[L-02]	Use `storage` variables	Low	Fixed

[L-01] Ensure the validity of parameters of functions

A small note to ensure the validity of various parameters of functions such as the `release` function just to check that addresses aren't `address(0)` or the `amountToRelease` isn't 0. Better to explicitly revert early when data passed into a function is non-compliant with the goals of the protocol.

[L-02] Use `storage` variables

An early version of the contract had many references to mappings that looked like the below:

```
if (
        selfStakes[staker].slashedInRound != 0 &&
        selfStakes[staker].slashedInRound != currentSlashRound
      ) {
        if (selfStakes[staker].slashedInRound == currentSlashRound - 1) {
          // If this is a slash from the previous round (not yet burned), move
          // it to the current round
          totalSlashed[currentSlashRound - 1] -=
```

```
selfStakes[staker].slashedAmount;
          totalSlashed[currentSlashRound] += selfStakes[staker].slashedAmount;
        } else {
          // Otherwise, this is a stale slash and can be overwritten
          selfStakes[staker].slashedAmount = 0;
        }
      }
```

It's generally both good Solidity practice to write this as `Staker storage staker = selfStakes.staker` and then read and write to the struct accordingly. It also comes with some gas savings.

Conclusion

Overall the `IdentityStaking` contract was well documented, well structured and performed the functions that it set out to do. The Gitcoin team provided a thorough README with intentions, descriptions, and diagrams that help the user, reader, or developer, to understand the intention and execution of the building of this protocol.