Security Audit

November 2023

Possum: Portals, Version One

Audited by
Ethan Bennett

Contents

About Darklinear Solutions

Darklinear Solutions provides unrivaled security for blockchain applications, from the bytecode to the browser. With years of experience in smart contract development and traditional software engineering, we find the bugs that others miss. Learn more at darklinear.com.

Introduction

Possum's Portals allow users to deposit yield-earning assets to receive their yield upfront. This review consists of an issue found in the Hats Finance competitive audit for Possum's Portals V1 in November 2023. It does not represent a full and exhaustive audit of the protocol.

Finding

Users can bypass `portalEnergy` calculations when unstaking to steal extra yield from the contract

Severity: High risk

Description: An attacker can inflate their `portalEnergy` infinitely by exploiting the rounding error on L314 of `Portal.sol` — since the division will round down, any `_amount` small enough to make `_amount * maxLockDuration` less than `SECONDS_PER_YEAR` will return `0`. Any logic that uses the raw `_amount` proceeds as intended, so the attacker still receives their unstaked funds like they normally would. But they also keep all the `portalEnergy` they built up in the process.

Proof of concept: It is possible that this could be exploited more efficiently or with greater impact than the loop in this PoC, but this attack is already significant enough to consider it high-risk: `portalEnergy` has real value within `Portal`, and that value can be stolen without limit by any attacker.

Recommendation: The simplest solution would likely be instituting a minimum for the `_amount` of principal tokens in both `stake` and `unstake`. Note that this minimum is a function of the `maxLockDuration`: it should not be referenced explicitly, since it will change with `maxLockDuration` (which, itself, is designed to change). As is, the minimum number of principal tokens with which the contract operates properly is `5`.

You could check for this dynamically with:

```
require (_amount * maxLockDuration >= SECONDS_PER_YEAR)
```

The reason I suggest this over adjusting the math is that rounding up instead of down could present a different risk in this context. However, if it is important to retain this functionality for small `_amounts`, the math could be reworked instead.