SMART CONTRACT AUDIT REPORT

Radicle Drips

Author - Satyam Agrawal

License

Disclaimer

ABOVE AUDIT IS NOT A SECURITY WARRANTY, INVESTMENT ADVICE, OR AN ENDORSEMENT OF THE RADICLE.

THIS AUDIT DOES NOT PROVIDE A SECURITY OR CORRECTNESS GUARANTEE FOR THE AUDITED SMART CONTRACTS. THE STATEMENTS MADE IN THIS DOCUMENT SHOULD NOT BE INTERPRETED AS INVESTMENT OR LEGAL ADVICE, NOR SHOULD ITS AUTHORS BE HELD ACCOUNTABLE FOR DECISIONS MADE BASED ON THEM.

SECURING SMART CONTRACTS IS A MULTISTEP PROCESS. ONE AUDIT CANNOT BE CONSIDERED ENOUGH. WE RECOMMEND THE RADICLE TEAM ORGANISE A BUG BOUNTY PROGRAM TO ENCOURAGE FURTHER ANALYSIS OF THE SMART CONTRACT BY OTHER THIRD PARTIES.

Introduction

Goals of this Report

Author has been engaged by Radicle Foundation to perform a security audit of the Radicle drips hub contract codebase.

The audit's focus was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped into the following three categories:

Security: Identifying security-related issues within each contract and the system of contracts.

Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity Improving scalability
- Quantity and quality of test coverage

Scope for the Audit

The audit has been performed on the following GitHub repositories:

Repository	Commit hash
https://github.com/radicle-dev/drips-contracts	`835656a99015e3cc28ee1003924654e5071f3d00`

Severity Classification

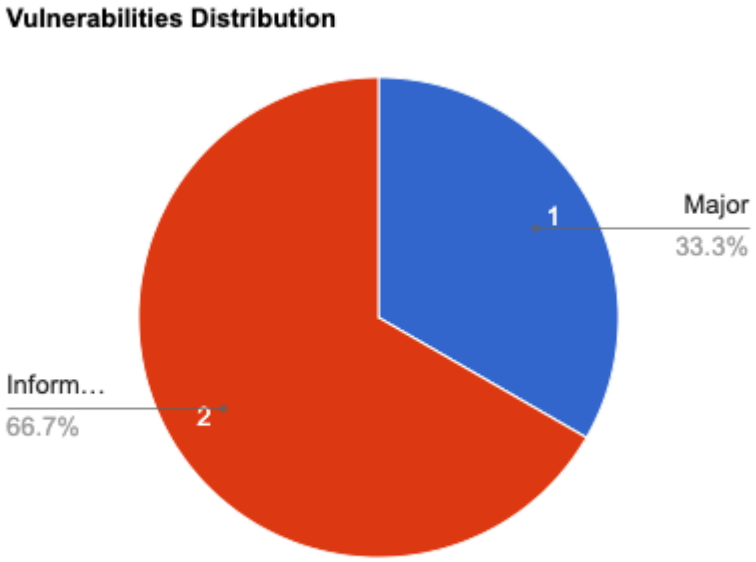
This report classifies the issues found into the following severity categories:

Severity	Description
Critical	These are issues that we managed to exploit. They compromise the system seriously. We suggest fixing them **immediately**.
Major	These are potentially exploitable issues. We did not manage to exploit them, and maybe they can not be exploited right now, or the impact is not clear, but they represent a security risk that can arise problems in the near future. We suggest fixing them as soon as possible.
Minor	These issues represent problems that are relatively small or difficult to exploit but can be used in combination with other issues. These kinds of issues do not block deployments. They should be taken into account and fixed eventually.
Informational	These kinds of findings do not represent a security risk. They are best practices that we suggest implementing.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Summary of Findings

TYPE	CRITICAL	MAJOR	MINOR	INFORMATIONAL
Open	0	0	0	0
Acknowledged	0	1	0	2
Closed	2	0	0	0



Detailed Findings

1. Set Splits would be lost the splittable value for old split receivers.

Severity: *Major*

Context: DripsHub.sol#L459

`setSplits` function can be called independently of the split function, but if there is already some value that has been ready to split but `split` function would not be explicitly called on-chain. If the user again calls `setSplits` with new receiver sets, then old receivers will not be able to receive split funds anymore.

Recommendation

We recommend calling the `split` function before calling `setSplits`.

Status: *Acknowledged*

Client Comment: This is the designed and expected characteristic of the protocol.

2. Inefficient mathematical operations

Severity: *Informational*

Context: Drips.sol#274

The drips-receiving procedure will be facilitated by frequent calls to `_receiveDripsResult()`. For a frequent drips receiver user, even tiny gas savings can add to significant monetary savings. Because `receivableCycles` and `toCycle` cannot go underflow, it is unnecessary to waste gas doing inherent underflow and overflow checks.

Multiple places can be found in the codebase to save gas like this.

Recommendation

We recommend using `unchecked` blocks to avoid inherent underflow and overflow checks on mathematical operations.

Status: *Acknowledged*

Client Comment: Worth looking into across the entire protocol.

3. Inconsistent naming convention used in the codebase.

Severity: *Informational*

Context: DripsHub.sol#239

The term `userId` is used throughout the codebase to identify the setter or receiver of drips and when the person is doing an operation on its receivable drips. While the `receiver` variable name is used to specify the recipient of drips. But in this situation, `userId` is used for the recipient and `senderId` for the person who initiated the drips. As a result code readability is reduced and creates confusion.

Recommendation

It is preferable to have uniformity across the codebase. Use `userId` for the person who set the drips and `receiver` or `receiverId` for the person for whom the drips are set.

Status: *Acknowledged*

Client Comment: We need some time to decide whether we want to alter the convention or not.