Drips
Security Review

Cantina Managed review by:
Deadrosesxyz, Lead Security Researcher
Sujith somraaj, Security Researcher
J4x, Associate Security Researcher

September 4, 2024

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Medium Risk . . . 4
  - 3.1.1 `commonFunds` can be burned by anyone . . . 4
- 3.2 Low Risk . . . 4
  - 3.2.1 All repo's ownership can be removed if Github/Gitlab goes down for 5 seconds . . . 4
- 3.3 Gas Optimization . . . 5
  - 3.3.1 `withdrawUserFunds` could be optimized to avoid unnecessary function call . . . 5
  - 3.3.2 `_cancelAllGelatoTasks` function could be optimized . . . 6
- 3.4 Informational . . . 6
  - 3.4.1 Do not add `Multicall` to future versions of RepoDriver . . . 6
  - 3.4.2 Replace magic numbers with named `constants` to improve code quality . . . 7

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Drips is a protocol and app built on Ethereum that enables organizations and individuals to directly and publicly provide funding to the free and open source software projects they depend on the most.

Drips also includes gas-optimized and integrated primitives for streaming and splitting tokens, allowing users and web3 apps to stream and split funds by the second with continuous settlement for use cases like contributor payments, vesting and subscription memberships.

From Jul 23rd to Jul 26th the Cantina team conducted a review of drips-monorepo on commit hash 78ce9efb. The team identified a total of **6** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 1
- Low Risk: 1
- Gas Optimizations: 2
- Informational: 2

3 Findings

3.1 Medium Risk

3.1.1 `commonFunds` can be burned by anyone

Severity: Medium Risk

Context: RepoDriver.sol#L309-L325

Description: The Drips protocol allows users to request updates to the ownership of a repo through the `requestUpdateOwner()` function. This will trigger a call through Gelato, which will result in a fee being charged to the contract. This fee can be paid from two different sources:

1. The users' deposited funds.
2. The `commonFunds`.

The protocol will first use the users deposited funds, and if there are none, the common funds are used.

```
if (userFundsUsed >= amount) {
    userFundsUsed = amount;
} else {
    require(commonFunds() >= amount - userFundsUsed, "Not enough funds");
}
if (userFundsUsed != 0) {
    _gelatoStorage().userFunds[payer] -= userFundsUsed;
    _gelatoStorage().userFundsTotal -= userFundsUsed;
}
Address.sendValue(gelatoFeeCollector, amount);
```

The problem is that there is no restriction on requesting these callbacks from Gelato. So a malicious user could generate countless callbacks which would drain the whole `commonFunds` without any guards in place to stop him,

Recommendation: We recommend only allowing updates through trusted identity (e.g. a whitelist) to prevent malicious actors from burning the `commonFunds` every time new ones get deposited.

Drips: Acknowledged. This is the intended behavior, we want to subsidize user calls. When the common funds run out, the users can still use the protocol, it just won't be subsidized anymore. The recommendation introduces a trusted identity which is against the trustless design of the protocol and brings security risks around maintenance of the whitelist.

Cantina Managed: Acknowledged.

3.2 Low Risk

3.2.1 All repo's ownership can be removed if Github/Gitlab goes down for 5 seconds

Severity: Low Risk

Context: index.ts#L69

Description: The Drips protocol uses an asynchronous Gelato task to update repos ownership. Users will call the `RepoDriver` contract to emit an `OwnerUpdateRequested` event. Based on this task, the gelato task will trigger, and the set typescript file will be run. In this file, the code generates a URL for the path to the repos `Funding.json` file and tries to retrieve the raw file from Github/Gitlab.

```
try {
  repoName = toUtf8String(name);
  let url: string;
  switch(forge) {
    case 0:
      url = `https://raw.githubusercontent.com/${repoName}/HEAD/FUNDING.json`;
      break;
    case 1:
      url = `https://gitlab.com/${repoName}/-/raw/HEAD/FUNDING.json`;
      break;
    default:
      throw Error(`Unknown forge ${forge}`);
  }
  const funding: any = await ky.get(url, { timeout: 25_000, retry: 10 }).json();
  owner = getAddress(funding.drips[chain].ownedBy);
} catch (error_) {
  error = error_;
}
```

As one can see, the code will try to be retrieved ten times from Github/Gitlab with a timeout of 25 seconds. This timeout will never trigger, as the call will already fail after 5 seconds based on the framework. If Github stays down longer than that, the call will fail, and the `catch` block will continue. The problem is that the function call to `updateOwnerByGelato()` will still be made in this case:

```
if(error) console.log("Error:", error)

return { canExec: true, callData: [functionCall] };
```

This function call will use the never overwritten default value for `owner`, which gets set before.

```
let owner = AddressZero;
```

This allows an attacker to wait until Github/Gitlab has an outage and then quickly calls the `requestUpdateOwner()` function for any repo he wants. This repo's ownership will then be overwritten with the `address(0)`, and the owner will only be able to reclaim ownership once he realizes this and Github is back up.

Recommendation: We recommend not calling the contract if the try/catch block returns an error. This way, the old assignment will persist.

Drips: Acknowledged, won't fix. The current behavior of not assuming ownership unless it can be actually looked up is a safety measure. When the user needs to quickly update the ownership, e.g. because their wallet was compromised, a GitHub/GitLab outage shouldn't slow down cutting off the attacker. An `address(0)` ownership can be easily fixed when the service gets restored, but stolen funds can't.

Cantina Managed: Acknowledged.

3.3 Gas Optimization

3.3.1 `withdrawUserFunds` could be optimized to avoid unnecessary function call

Severity: Gas Optimization

Context: RepoDriver.sol#L362

Description: In the `withdrawUserFunds` function, we can reduce gas consumption by directly accessing the storage variable instead of calling the `userFunds` function. This will reduce gas consumption by at least 20 GAS in all possible paths.

Recommendation: Replace the `userFunds(user)` function call with direct access to the storage variable:

```
function withdrawUserFunds(uint256 amount, address payable receiver)
    public
    whenNotPaused
    returns (uint256 withdrawnAmount)
{
    address user = _msgSender();
    uint256 maxAmount = _gelatoStorage().userFunds[user];
    // ... rest of the function
}
```

Drips: Acknowledged, won't fix. This is almost certainly an inlining bug or deficiency in the optimizer, it either is or probably will be fixed in the newer Solidity versions, especially with via-IR. I don't think that 20 gas per call is worth the slightly decreased readability and less DRY code.

Cantina Managed: Acknowledged.

3.3.2 `_cancelAllGelatoTasks` function could be optimized

Severity: Gas Optimization

Context: RepoDriver.sol#L224

Description: The `_cancelAllGelatoTasks` internal function cancels all pending tasks before creating a new one. This admin-only function doesn't cache the loop length and uses `i++` instead of `++i`, which can save gas costs.

Recommendation: Consider optimizing the function as recommended below,

```
    function _cancelAllGelatoTasks() internal {
        // `IAutomate` interface doesn't cover `getTaskIdsByUser.`
        bytes32[] memory tasks = IAutomate2(address(gelatoAutomate)).getTaskIdsByUser(address(this));
        uint256 len = tasks.length;

        for (uint256 i; i < len; ++i) {
            gelatoAutomate.cancelTask(tasks[i]);
        }
    }
```

This optimization reduced the gas costs from `195418` GAS to `195402` GAS for two cancellations and will exponentially reduce more if the length is considerable.

Drips: Acknowledged, won't fix. The recommended code is more verbose and this optimization is already done automatically in the newer version of Solidity with via-IR. The saving of 16 gas (0,0082% or about 0,067$ on Ethereum and a few orders of magnitude less on L2s) in a function that will be run a few times in the entire protocol lifetime isn't worth it.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Do not add `Multicall` to future versions of RepoDriver

Severity: Informational

Context: RepoDriver.sol#L29

Description: It must be noted that the current version of `RepoDriver` is incompatible with `Multicall`. If `Multicall` is ever added to the contract the following issues will occur:

- Attacker will be able to impersonate calls from other users due to the usage of `ERC2771Context`.
- Attacker will be able to inflate their balance (and later withdraw it) by multicalling deposit, which uses the `msg.value`.

Recommendation: A fix is not needed. Protocol should simply be aware to not add `Multicall` to future versions of the protocol

Drips: Acknowledged. We aren't planning switching `Caller` to `Multicall`.

Cantina Managed: Acknowledged.

3.4.2 Replace magic numbers with named `constants` to improve code quality

Severity: Informational

Context: RepoDriver.sol#L159, RepoDriver.sol#L184, RepoDriver.sol#L189

Description: The `RepoDriver` contract contains several instances of magic numbers - literal values used directly in the code without explanation. These magic numbers reduce code readability and make maintenance more difficult. Replacing these with named constants would improve code quality.

```
function calcAccountId(Forge forge, bytes calldata name)
    public
    view
    returns (uint256 accountId){
    // ...
    if (name.length <= 27) {
        // ...
    }
    // ...
    accountId = (accountId << 8) | forgeId;
    // ...
    accountId = (accountId << 216) | nameEncoded;
}
```

Recommendation: Identify all magic numbers in the contract and declare them as named constant with descriptive names.

```
uint8 private constant MAX_NAME_LENGTH = 27;
uint8 private constant FORGE_ID_BITS = 8;
uint16 private constant NAME_ENCODED_BITS = 216;

function calcAccountId(Forge forge, bytes calldata name)
    public
    view
    returns (uint256 accountId){
    // ...
    if (name.length <= MAX_NAME_LENGTH) {
        // ...
    }
    // ...
    accountId = (accountId << FORGE_ID_BITS) | forgeId;
    // ...
    accountId = (accountId << NAME_ENCODED_BITS) | nameEncoded;
}
```

Drips: Acknowledged. I disagree that turning these magic numbers into constants would improve the code quality. I tried it before and it resulted in code that was more difficult to read and reason about. In the current code the magic numbers are highly local, they are always used once, they are documented with comments, and it's immediately clear what their values are. Introducing constants would move away these values from their context, their documentation would need to be verbose because it would need to be readable in isolation, and it wouldn't be immediately clear where exactly these numbers are used in code, but it wouldn't make the code more DRY because they would be used only once anyway.

`MAX_NAME_LENGTH` probably should be split into `MAX_NAME_LENGTH_GITHUB` and `MAX_NAME_LENGTH_GITLAB` because these forges have entirely independent ID spaces, the magic number `27` repeats by coincidence, not by design.

Cantina Managed: Acknowledged.