PUBLIC

Code Assessment

of the Protego

Smart Contracts

October 17, 2024

Produced for

MAKER

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help MakerDAO with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Protego according to Scope to support you in forming an opinion on their security risks.

MakerDAO implements Protego, a framework for standardizing the deployment of Emergency Spells that drop proposals queued for execution.

The most critical subject covered in our audit is functional correctness. The general subjects covered are trustworthiness and documentation. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Protego repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	30 May 2024	fc1f94c1a874634af82a339259bac171c8b7b0e9	Initial Version
2	10 June 2024	d18899d4a3aa3818351908d71c7dc171093f8951	After Intermediate Report
3	16 Oct 2024	eef9ec0e92f64fe7dcd821369fd9a0b46a85463c	Update nextCastTime

For the solidity smart contracts, the compiler version `0.8.16` was chosen.

The following contracts are in the scope of this review:

```
src/
    EmergencyDropSpell.sol
    Protego.sol
```

As well as the deployment scripts:

```
scripts/
    ProtegoDeploy.s.sol
    dependencies/
        ProtegoDeploy.sol
        ProtegoInstance.sol
```

2.1.1 Excluded from scope

Any file not explicitly listed above including tests are out of the scope of this review.

2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

At the end of this report section we have added subsections for each of the changes accordingly to the versions.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

MakerDAO implemented Protego, a tool facilitating and standardizing the deployment of emergency spells for dropping plans scheduled in `MCD_PAUSE` should the need arise. Protego implements another mode of operation, intended for more extreme scenarios, where the Protego contract can be elected as the `hat` of the `DsChief` allowing anyone to permissionelessly drop any scheduled plan.

Technically, executive proposals in MakerDAO work as follows:

1. A spell is deployed.
2. This spell is voted for in the `DsChief`.
3. Upon receiving sufficient votes, the spell can be lifted to become the `hat` in the DsChief (`DSChief.lift()`). This enables the spell to schedule itself in `MCDPause` (`DSPause.plot()`).
4. After the wait time has elapsed, anyone can execute (`DSPause.exec()`).

Access control in `MCDPause` is based on `DSAuth` with the `DSChief` set as `authority`. `canCall()` of the `DSChief` allows the `hat` (if the Chief is live) to call any function. It is expected for spells to become the `hat` to receive the required privileges.

A spell has the following attributes:

- `usr`: spell implementation address
- `tag`: spell codehash
- `fax`: spell calldata
- `eta`: earliest spell execution time

Protego

Protego is an immutable contract facilitating the dropping of scheduled spells in `MCD_PAUSE` to prevent them from executing. It offers two main functions, one for both modes of operation:

- `deploy()`: Deploys a new EmergencyDropSpell (described below).
- `drop()`: allows to directly drop any specified spell in `MCD_PAUSE`. Requires the Protego contract to be the `hat` in the `DSChief` (to be used in extreme scenarios only).

Furthermore, the following view/pure functions are provided:

- `planned(address _usr, bytes32 _tag, bytes memory _fax, uint256 _eta)` and `planned(bytes32 _id)`: Returns `True/False` depending if the spell is planned in `MCD_PAUSE`.
- `id()`: Helper to calculate the `id` (keccak256) of a spell based on the attributes.

EmergencyDropSpell

While the naming suggests that the contract is a spell, it operates differently from regular ones. Most importantly, instead of being scheduled and being executed after a time delay in the context of the `MCD_PAUSE_PROXY`, it drops the defined spell (`DSPause.drop()`) using its privileges after being elected as `hat` in `DsChief`.

Upon deployment, the spell to be dropped (called original spell) is fixed by immutables set in the constructor (`usr`, `tag`, `fax` and `eta`).

The following functions are available:

- `schedule()` - Alias for `drop()`.
- `drop()`- Drops the original spell in the `MCD_PAUSE`.
- `description()` - Returns the description of the spell in the format "MakerDAO Drop Spell: <ID>".
- `done()` - Returns the inverse of `planned()`, i.E. `True` if the original spell is no longer planned.
- `planned()` - Returns `True/False` depending if the original spell is planned.

Furthermore, there are getters to read the contract information:

- `protego()` - returns the address of the Protego contract.
- `pause()` - returns the address of `MCD_PAUSE`.
- `action()`, `tag()`, `eta()`, `sig()` - returns the respective information about the **original** spell.

For interface compatibility with `DssExec`, `cast()` (implemented as a no-op) as well as getters for `log` (address of Chainlog), `officeHours` (`False`), `expiration` (`type(uint256).max`) and `nextCastTime` (`block.timestamp` at deployment) have been made available but are not intended to be used.

Deployment

Protego instances are deployed with `MCD_PAUSE` as the constructor parameter. Since the contract does not need any special privileges in the system normally, no spell needs to be executed. Note that Protego is not added to the Chainlog.

2.2.1 Changes in Version 3

In this version, `nextCastTime` has been updated to `uint256.max` to improve semantics, since an emergency spell will be executed during schedule phase, without the need to be cast.

2.2.2 Roles & Trust Model

Protego is an immutable contract with no special privileges by default. There are no privileged roles in this contract, any functionality is permissionless.

Deployed EmergencySpells must be voted for and be backed by a sufficient amount of MKR in the `DsChief`. If successful, the spell can be lifted to become the `hat` which is the required role to drop a plan from `MCD_PAUSE`.

Should the need arise, MKR holders can elect the Protego contract to be the `hat` in `DsChief` to enable the use of function `drop` allowing dropping anything that has been planned on `MCD_Pause`.

The majority of MKR token holders is trusted to act honestly and correctly at all times and to vote for such a proposal when needed.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0
Informational Findings	1

- Inaccurate NatSpec Code Corrected

6.1 Inaccurate NatSpec

Informational Version 1 Code Corrected

CS-MPRO-001

The contracts' NatSpec describes the ETA as the expiry date. However, it corresponds to the earliest execution date. That is the case for the following functions:

1. `Protego.deploy`
2. `Protego.id`
3. `Protego.planned`
4. `Protego.drop`
5. `EmergencySpellLike.eta`
6. `EmergencyDropSpell.constructor`

Further, the NatSpec of `EmergencyDropSpell.done` specifies that it returns `true` in case the original spell has been dropped. However, it will return `true` if the spell is not planned (e.g. has never been planned and thus never been dropped).

Code corrected:

The specification has been adjusted.