PUBLIC

Code Assessment

of the Dss Proxy Smart Contracts

June 10, 2022

Produced for

by

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 Dss Proxy according to Scope to support you in forming an opinion on their security risks.

DssProxy implements a replacement for DSProxy, a proxy contract for users to use with the ProxyActions contracts of the Maker applications such as Oasis.app.

The most critical subjects covered in our audit are security and functional correctness.

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	4
• Code Corrected	4

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 Dss Proxy 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	4 April 2022	d7a6348eebe7f4f412b9969ca31b213f4c7a694a	Initial Version
2	4 May 2022	3b46093e41f43486f5dbff832c787729bb347066	After Intermediate Report
3	8 June 2022	10b871627ab8861e7a85644d7c4233c5fe4717e0	Third Version

For the solidity smart contracts, the compiler version `0.8.13` was chosen. For the third version, compiler version `0.8.14` is used.

Contracts DssProxy and DssProxyRegistry are in scope of this review.

2.1.1 Excluded from scope

All contracts not listed above.

2.2 System Overview

DssProxy is a replacement for the currently used DSProxy. It`s a simple proxy contract which allows the owner to call any address via Delegatecall. Delegatecall executes the code at the target address within the environment (storage) of the caller's account (the DssProxy).

When an address interacts with any Maker application (e.g. the Oasis.app) for the first time, the user is asked to deploy such a proxy contract. Using this proxy contract, the user then interacts with the Maker application by executing the code of so-called ProxyAction contracts, smart contracts aggregating logic.

DssProxy offers the following functionality:

- `execute(address target_, bytes memory data_)`: Main proxy function allowing a caller able to pass the `auth` modifier to execute the code of the target address as Delegatecall. The target cannot be address `0x0`. If the call is successful its response is returned. If the Delegatecall fails, this function reverts, passing along the response.
- `setOwner()` bearing the `auth` modifier, allows changing the `owner` of the DssProxy.
- `setAuthority()` bearing the `auth` modifier, allows changing the `authority` of the DssProxy.

The proxy contract has an `owner` and an `authority` stored in the storage of the contract. As `execute()` may execute arbitrary code as Delegatecall, storage slots of the DssProxy may be overwritten. Generally, the executed code should only contain logic and not access/write to the contract's storage.

Access control is implemented using the `auth` modifier: This modifier allows the `owner` to call any function and additionally, if the `authority` is set to a non-zero address, `canCall(msg.sender, address(this), msg.sig)` is called on on the authority contract in order to determine whether access should be allowed.

Contract DssProxyRegistry serves as a factory to deploy new DssProxy contracts for individual users. Anyone who currently doesn't own a registered DssProxy may deploy one using the `build` function. The contract keeps track of the deployed DssProxies and their owners. As ownership of a DssProxy may be transferred, the `claim` function allows the new owner of the DssProxy to update the entry in the registry.

2.2.1 Trust model & Roles

DssProxy:

- `owner`: owner of the proxy, fully trusted
- `authority`: DSAuth-like authority, if set it can determine whether a caller can call a function of the DssProxy

The caller of the `execute` function is trusted to only target contracts which do not access/modify storage. Otherwise, storage slots of the DssProxy might be overwritten, in particular this may affect the stored `owner` and `authority`.

New owners of a DssProxy (after a transfer of ownership) must be very aware of the authority role and it's privilege. These privileges may include accessing funds of the DssProxy or to retake the ownership.

DssProxyRegistry:

Fully trustless. Anyone may deploy a new DssProxy using the `build` function. New owners of a DssProxy may update the entry in the registry using the `claim()` function. Users claiming a proxy they became owner of are trusted to understand the power of the `authority` if set for their DssProxy.

2.2.2 Changes in Version 3

The following changes have been made:

- The `authority` and `owner` variables are now ordered as in the old DSProxy contract.
- The `auth` modifier allows now that the DssProxy calls itself.

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:

- Design: Architectural shortcomings and design inefficiencies

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	4

- DssProxy Constructor Does Not Emit SetOwner Event Code Corrected
- Optimization of delegatecall Success Check Code Corrected
- Possible Failure of create2 Code Corrected
- Possible Optimization in Proxy Check of Registry Code Corrected

6.1 DssProxy Constructor Does Not Emit SetOwner Event

Design Low Version 1 Code Corrected

The constructor of DssProxy does not emit a SetOwner event. Consider emitting an event here to reflect this important storage change.

```
constructor(address owner_) {
    owner = owner_;
}
```

Code corrected:

The constructor now emits the `setOwner` event.

6.2 Optimization of `delegatecall` Success Check

Design Low Version 1 Code Corrected

The success check in the `execute` function of the DssProxy contract is as follows:

```
assembly {
    let succeeded := delegatecall(/*...*/)
    /*...*/
    switch iszero(succeeded)
```

```
    case 1 {
        revert(add(response, 0x20), size)
    }
}
```

However, as `delegatecall` can only return 0 or 1, the `iszero` is unnecessary. Instead, one can simply check for `case 0`. With optimization enabled, this change saves 9 gas and 4 bytes of bytecode.

Code corrected:

The optimization has been implemented.

6.3 Possible Failure of `create2`

Design | Low | Version 1 | Code Corrected

It is possible for the `create2` operation to fail, in which case the returned address will be 0. This failure is not checked, which would result in `isProxy[0]` being set to 1. Additionally, the owner's seed would be incremented despite not having deployed a contract.

```
assembly {
    proxy := create2(/*...*/)
}
proxies[owner_] = proxy;
isProxy[proxy] = 1;
```

Code corrected:

The code now ensures that the DssProxy has been successfully created:

```
require(proxy != address(0), "DssProxyRegistry/creation-failed");
```

6.4 Possible Optimization in Proxy Check of Registry

Design | Low | Version 1 | Code Corrected

In the `claim` function of the DssProxyRegistry, the following check is made:

```
require(isProxy[proxy] == 1, "DssProxyRegistry/not-proxy-from-this-registry");
```

The `isProxy` mapping only contains the values 0 or 1. Hence, checking the condition `== 1` is functionally equivalent to checking the condition `!= 0`. The latter check is more efficiently compiled, it saves 6 gas and reduces bytecode by 3 bytes.

Code corrected:

The optimization has been implemented.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 No Event on `execute`

Note Version 1

Integrations must be aware that compared to the DSProxy it replaces, DssProxy no longer emits a event on `execute()`.

7.2 `isProxy` Might Point to Addresses Without Code

Note Version 1

When creating a proxy with the `build` function, an entry is created in `isProxy`, which maps the address of the new proxy to 1 :

```
function build(address owner_) external returns (address payable proxy) {
    /*...*/
    isProxy[proxy] = 1;
}
```

This entry cannot be modified. Hence, a proxy that has been selfdestructed would still appear in `isProxy` like a valid proxy.

A selfdestructed proxy in the `isProxy` mapping would have prevented creation of a new proxy for the owner using `DssProxyRegistry.build()`: Retrieving the owner would have reverted. The implementation of `DssProxyRegistry.build()` has been changed to handle this case.