PUBLIC

Code Assessment

of the UniV2 Migration Deployment Scripts

October 11, 2023

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 UniV2 Migration according to Scope to support you in forming an opinion on their security risks.

MakerDAO implements a migration script that moves MakerDAO's DAI/MKR Uniswap v2 LP position to a new NST/NGT pool.

The most critical subjects covered in our audit are functional correctness, access control and frontrunning resistance.

Security regarding all 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	1
• Code Corrected	1

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 UniV2 Migration repository based on the documentation files.

The following deployment scripts are part of the scope of this review:

1. deploy/UniV2PoolMigratorInit.sol

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	18 September 2023	8036feb8f139938ca85329c6e43933241bd80727	Initial Version
2	10 October 2023	3b7fa42cfdaae8669ca235080111efad218c8336	Version 2

For the solidity smart contracts, the compiler version `>=0.8.0` was chosen.

2.1.1 Excluded from scope

Any other file not explicitly mentioned in the scope section. In particular tests, scripts, external dependencies, and configuration files are not part of the audit scope.

2.2 System Overview

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

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

MakerDAO offers *UniV2PoolMigratorInit*, a library to migrate MakerDAO's MKR/DAI liquidity position on Uniswap V2 to an NGT/NST liquidity position on Uniswap V2.

MakerDAO maintains MKR/DAI liquidity in a Uniswap V2 pool and periodically adds system surplus to the liquidity. As part of Endgame, MakerDAO plans to rebrand DAI and MKR, releasing two new equivalent tokens provisionally called NST (New Stable Token) and NGT (New Governance Token). NST will be exchangeable with DAI at a 1:1 ratio, while MKR will be exchangeable with NGT at a 1:1200 ratio. The *UniV2PoolMigratorInit* library allows Maker governance to migrate the DAI/MKR position to a new NST/NGT pool.

Liquidity is first withdrawn from the DAI/MKR pool by burning the LP tokens held by the PauseProxy. The DAI and MKR amounts transferred to the PauseProxy are then exchanged respectively for NST, in the *DaiNgt* contract, and for NGT in the *MkrNgt* contract, and finally the new amounts are used to provide liquidity to a yet empty UniswapV2 pool of NST/NGT.

To ensure the NST/NGT Uniswap V2 pool is empty, the two tokens have to be initialized in the same governance spell as the one which handles the migration.

2.2.1 Roles & Trust Model

The script is called by the PauseProxy after a governance vote. The migration is permissionless.

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:

- Correctness: Mismatches between specification and implementation

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	1

- Wrong Chainlog Identifiers Code Corrected

6.1 Wrong Chainlog Identifiers

Correctness Low Version 1 Code Corrected

CS-UV2M-001

`UniV2PoolMigratorInit.init()` retrieves the `DaiNst` and `MkrNgt` addresses from the chainlog the following way:

```
DaiNstLike daiNst = DaiNstLike(dss.chainlog.getAddress("DAINST"));
MkrNgtLike mkrNgt = MkrNgtLike(dss.chainlog.getAddress("MKRNGT"));
```

The correct strings (as set by the corresponding deployment scripts), however, are "DAI_NST" and "MKR_NGT" respectively.

Code corrected:

Labels `DAINST` and `MKRNGT` have been replaced with the correct ones: `DAI_NST` and `MKR_NGT`.

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 Note on the Safety of Uniswap Operations

Note **Version 1**

Two main considerations have been made when assessing the security of the script's Uniswap operations with regard to potential unwanted value extraction by adversaries.

1. Let's consider a Uniswap V2 pool with token amounts (x, y), and we split it into two pools with amounts $(\alpha x, \alpha y)$ and $((1-\alpha)x, (1-\alpha)y)$. This is equivalent to creating two smaller pools at the original price $\frac{x}{y}$, as happens when removing liquidity and creating a new pool with that liquidity. Then, the $\Delta\hat{y}$ amount obtained by trading an amount Δx in any combination in the two pools is at most equal to the amount Δy obtained when trading Δx in the original pool. This can be seen by examining the function $\Delta\hat{y}(t)$, which returns the output amount when trading $t\Delta x$ in the $(\alpha x, \alpha y)$ pool and $(1-t)\Delta x$ in the $((1-\alpha)x, (1-\alpha)y)$ pool, with $t \in (0, 1)$, and observing that the function is at a maximum when $t = \alpha$ as can be seen from its 0 derivative in that point and its negative second derivative over $(0, 1)$. This shows that splitting a big pool into two smaller pools, as the script does, does not expose value that can be extracted by back-running.

2. When presented with a pool with amounts (x, y) and another pool with amounts (x, ky), if Δy is obtained by trading Δx in the first pool, then $k\Delta y$ is obtained by trading Δx in the second pool. This shows that when replacing a token with a scaled version, such as *MKR* and *NGT*, the Uniswap operations safely scale in the same way.

In conjunction, these considerations make it so that:

1. no extractable value is exposed when splitting the pool (removing liquidity and creating a new pool) without slippage checks.
2. no extractable value is exposed when converting the *MKR* tokens to *NGT* for the new pool.