PUBLIC

Code Assessment

# of the GemJoin9 for PAXG Smart Contracts

January 10, 2023

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 GemJoin9 for PAXG according to Scope to support you in forming an opinion on their security risks.

MakerDAO implements a novel join adapter (GemJoin9) to be used with the PAXG token, an ERC20 token with fees on transfers.

The most critical subjects covered in our audit are functional correctness and access control. Security regarding all the aforementioned subjects is high.

Please note that the PAXG token is upgradable. Furthermore the current implementation of the PAXG token features functionality that allows the admin to seize/freeze assets of any address.

The general subjects covered are gas efficiency and error handling. 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	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 GemJoin9 for PAXG repository based on the documentation files. `src/join-9.sol` is the only file in scope.

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

V	Date	Commit Hash	Note
1	16 May 2022	d25ccb6a631e7ede90dde358a02ab2726a1da019	Initial Version
2	6 December 2022	06df176390860ed48caf4a2a63c0905e2d5415c1	Second Version

For the solidity smart contracts, the compiler version `0.6.12` was chosen.

2.1.1 *Excluded from scope*

All files not listed above.

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 implements a novel join adapter (GemJoin9) to be used with the PAXG token, an ERC20 token with fees on transfers.

2.2.1 *GemJoin9*

GemJoin9 implements the following functionality:

- `join(usr, wad)` which pulls tokens from `msg.sender` using `transferFrom()` (increases the contract's surplus balance) and joins with `join(usr)`.
- `join(usr)` The surplus balance is computed using the current balance and the tracked balance. This allows to determine the amount of tokens received by the contract with the transfer fees taken in to account. Based on this amount the accounting in the VAT, the gem balance of this ilk, for this user is updated. The tracked balance in the Join adapter is updated to the current value.
- `exit(usr, wad)` exits the amount of tokens: The account in the VAT is updated by calling `Vat.slip()` and the tokens are transferred. Note that due to the transfer fees less tokens arrive at the receiver. The tracked balance in the Join adapter is updated to the current value.

The PAXG tokens remain locked at the join adapter while the funds are accounted for in the VAT.

For priviledged roles:

- Managing access control through functions `rely()` and `deny()` and enforcing it with the `auth` modifier.
- Restricted function `cage` to set `live` to 0. Used during shutdown or to offboard this collateral.

Considerations for Liquidations and VAT shutdown:

Buyers of auctions and participants of the shutdown step 9) `cash()` should be aware of the transfer fees when exiting the bought gem. Furthermore the considerations of the Trust Model regarding the PAXG token applies.

2.2.2 Roles and Trust Model

`auth` roles are fully trusted. Can trigger `cage` and set the `live` to 0. Expected to be the Maker Governance and the End contract.

Token addresses are fully trusted to behave correctly. Moreover, tokens are assumed to be standard ERC20 tokens with fees - for the purposes of this review the PAXG token only has been considered.

The PAXG Token is upgradable. Furthermore the current implementation features functionality to freeze or seize assets from accounts. This may negatively affect or break this join adapter.

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	1

- Tokens With More Than 18 Decimals Not Caught in Constructor Code Corrected

6.1 Tokens With More Than 18 Decimals Not Caught in Constructor

Design Low Version 1 Code Corrected

The constructor querries and stores the tokens decimals:

```
dec = gem.decimals();
```

Recent JOIN adapters, e.g. GemJoin7 have an explicit `require` statement preventing tokens with decimals > 18 to initialized. This is done as additional safety measure as some integrations, e.g. proxy actions are incompatible with / break for token with > 18 decimals. No such check is present in GemJoin9.

Code corrected:

Similar to other recent GemJoin adapters, the code in the constructor of GemJoin9 now ensures the token decimals do not exceed 18.

6.2 PAXG Token Used for Tests Does Not Match Actual PAXG Contract

Informational Version 1 Code Corrected

The PAXG token contract added to the repository and used for tests does not match the actual PAXG contract implementation. Testing hence is done with a different token implementation than the intended token the join adapter interacts with in production which is not ideal and might hide bugs.

Code corrected:

Forked mainnet tests have been added in the gemjoins integration tests. This ensures the code is tested with the actual PAXG token on chain.

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 PAXG Token Is Upgradable

Note Version 1

The PAXG Token is upgradable and an upgrade could break the integration with `gemjoin9`.

7.2 `wad` Amount in `Join` Event

Note Version 1

Function `function join(address usr, uint256 wad)` will transfer `wad` tokens from `msg.sender`. However, fees are going to be deducted. Hence, the actually joined amount will be the amount that arrived at the join adapter. Since the return value of the internal function `_join` is ignored and, hence, the event is emitted using the `wad` parameter, the emission will be incorrect. Also, any excess tokens will be ignored for the emitted join amount.

MakerDAO responded that this behavior is in line with the other gemJoins (e.g. join-3 or join-7). It's always the input amount that is logged.