Zap Smart Contract Audit

Kwenta, 18 January 2024

PUBLIC
hello@iosiro.com

1. Introduction

iosiro was commissioned by Kwenta to conduct a smart contract audit of their Zap smart contract. The audit was performed by two auditors on 17 January 2024, using two resource days.

This report is organized into the following sections.

- **Section 2 - Executive summary:** A high-level description of the findings of the audit.
- **Section 3 - Audit details:** A description of the scope and methodology of the audit.
- **Section 4 - Design specification:** An outline of the intended functionality of the smart contracts.
- **Section 5 - Open findings:** Detailed descriptions of audit findings that remained present at the conclusion of the audit.
- **Section 6 - Closed findings:** Detailed descriptions of audit findings that were addressed during the audit.

The information in this report should be used to understand the smart contracts' risk exposure better and as a guide to improving the security posture of the smart contracts by remediating the issues identified. The results of this audit reflect the in-scope source code reviewed at the time of the audit.

The purpose of this audit was to achieve the following:

- Identify potential security flaws.
- Ensure that the smart contracts function according to the documentation provided.

Assessing the off-chain functionality associated with the contracts, for example, backend web application code, was outside of the scope of this audit.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered high risk from cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. Strategies that should be used to encourage secure code development include:

- Security should be integrated into the development lifecycle, and the level of perceived security should not be limited to a single code audit.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed where possible.

2. Executive summary

This report presents the findings of an audit performed by iosiro on the Kwenta Zap smart contract.

The code was found to be of a high standard. The code had a high degree of readability, was well documented, and comprehensive fork tests were used to validate interactions with Synthetix as an external dependency.

PUBLIC
hello@iosiro.com

3. Audit details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from all other files was considered to be out-of-scope. Out-of-scope code that interacts with in-scope code was assumed to function as intended and not introduce any functional or security vulnerabilities for the purposes of this audit.

3.1.1 Smart contracts

- **Project name:** Zap
- **Commits:** d3daab2
- **Files:** Zap.sol, ZapErrors.sol, ZapEvents.sol

3.2 Methodology

The audit was conducted using a variety of techniques described below.

3.2.1 Code review

The source code was manually inspected to identify potential security flaws. Code review is a useful approach for detecting security flaws, discrepancies between the specification and implementation, design improvements, and high-risk areas of the system.

3.2.2 Dynamic analysis

The contracts were compiled, deployed, and tested in a test environment, both manually and through the test suite provided. Manual analysis was used to confirm that the code was functional and discover security issues that could be exploited.

PUBLIC
hello@iosiro.com

3.2.3 Automated analysis

Tools were used to automatically detect the presence of several types of security vulnerabilities, including reentrancy, timestamp dependency bugs, and transaction-ordering dependency bugs. Static analysis results were reviewed manually and any false positives were removed. Any true positive results are included in this report.

Static analysis tools commonly used include Slither, Securify, and MythX. Tools such as the Remix IDE, compilation output, and linters could also be used to identify potential areas of concern.

3.3 Summary of findings

The table below provides an overview of the audit's findings. Detailed write-ups are provided in Section 5 and Section 6.

#	Issue	Risk	Status
6.4.1	Missing referrer value	Informational	Closed
6.4.2	Missing contract validation	Informational	Closed

Each issue identified during the audit has been assigned a risk rating. The rating is determined based on the criteria outlined below.

- **High risk**: The issue could result in a loss of funds for the contract owner or system users.
- **Medium risk**: The issue resulted in the code specification being implemented incorrectly.
- **Low risk**: A best practice or design issue that could affect the security of the contract.
- **Informational**: A lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.

In addition to a risk rating, each issue is assigned a status:

- **Open**: The issue remained present in the code as of the final commit reviewed and may still pose a risk.
- **Closed**: The issue was identified during the audit and has since been satisfactorily addressed, removing the risk it posed.

4. Design specification

The following section outlines the intended functionality of the system at a high level. This specification is based on the implementation in the codebase. Any perceived points of conflict should be highlighted with the auditing team to determine the source of the discrepancy.

4.1 Zap

Zap was an abstract contract intended to be inherited by systems that need to convert between USDC and sUSD with zero fees. The contract consisted of two functions.

4.1.1 Zap In

The `_zapIn` function would transfer a caller-specified amount of USDC from the `msg.sender` to `this`. The USDC amount would first be approved to Synthetix's spot market; then it would be wrapped to sUSDC. The sUSDC was then approved to the spot market and sold for sUSD. The sUSD was kept in the contract at the end of the function call.

4.1.2 Zap Out

The `_zapOut` function would first approve a specified amount of sUSD to the spot market and use it to buy sUSDC. The sUSDC was approved to the spot market and then unwrapped to USDC. As USDC is commonly 6 decimals and sUSDC and sUSD are 18 decimals, up to 12 decimals of precision can be lost during the conversion process. The USDC was kept in the contract at the end of the function call.

5. Open findings

The following section details the findings of the audit which remained open at the conclusion of the final review.

5.1 High risk

No identified high-risk issues were open at the conclusion of the review.

5.2 Medium risk

No identified medium-risk issues were open at the conclusion of the review.

5.3 Low risk

No identified low-risk issues were open at the conclusion of the review.

5.4 Informational

No identified informational issues were open at the conclusion of the review.

6. Closed findings

6.1 High risk

No identified high-risk issues were closed at the conclusion of the review.

6.2 Medium risk

No identified medium-risk issues were closed at the conclusion of the review.

6.3 Low risk

No identified low-risk issues were closed at the conclusion of the review.

6.4 Informational

6.4.1 Missing referrer value

https://github.com/JaredBorders/zap/blob/1f209abb7f93f0ef242fed4ae164b7cc6e160ea6/src/Zap.sol#L155, *https://github.com/JaredBorders/zap/blob/1f209abb7f93f0ef242fed4ae164b7cc6e160ea6/src/Zap.sol#L208*

Description

`referrer` values are not passed in when calling Synthetix's `buy` and `sell` functions. Zero fees generated, so no referrer fees are lost. However, it may still be desirable to pass in `referrer` values to ensure Synthetix properly accounts for volume generated by Kwenta.

Update

Acknowledged with no change.

Previously, a referrer was included. However, it was discovered that the referrer is unnecessary since no fees are taken. Furthermore, when a referrer address is not specified (i.e., address(0)), the call requires less computation to complete, thus reducing gas spent by the user.

If desired, retroactive volume tracking can still be achieved via the ZappedIn and ZappedOut events. Or, given the functions are virtual, other means of tracking volume can be easily added.

6.4.2 Missing contract validation

https://github.com/JaredBorders/zap/blob/1f209abb7f93f0ef242fed4ae164b7cc6e160ea6/src/Zap.sol#L72

Description

Validation can be added in the constructor to require that `_DECIMALS_FACTOR != 1e18` to ensure a valid USDC address is passed in.

Update

Acknowledged with no change.