macro

Sapience A-2

Security Audit

April 2, 2026
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Sapience's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from March 26 to March 29, 2026.

The purpose of this audit is to review the source code of certain Sapience Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
Low	1	-	-	1
Code Quality	1	-	-	1

Sapience was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Discord with the Sapience team.
- Technical documentation available in the repository.

Source Code

The following source code was reviewed during the audit:

- **Repository:** sapience
- Commit Hash **(initial)**: `cfbaaa8045706ed96a604cb4484b55f11809553b`

We audited the following contracts with **f2b6bd9231ae9915f9ba88f6bef02e7a914e6288** commit hash:

Source Code	SHA256
packages/protocol/src/SecondaryMarketEscrow.sol	`1bec137aef2a1f962c40ae22f12e754e4cb958cf131a3e4db024662569acbcc9`
packages/protocol/src/interfaces/ISecondaryMarketEscrow.sol	`25b8666dc38cba5f9c6f5077e31c07aa07f76a12ae11fa8291f421238f56ff25`

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~L-1~~ Add ability to view `tradeHash` to verify parameters being signed

TOPIC	STATUS	IMPACT	LIKELIHOOD
Verification	Fixed	Medium	Low

Users can view their trade approval hash via `getTradeApprovalHash()` to verify the data hash being to be signed by the user. This requires inputting a `tradeHash` which contains a hash of important trade data including the token and collateral types, amount and price. Allowing users to be able to input this data and generate the trade hash used in the trade approval hash would help verify what they are actually signing, since a malicious hash could result in negative outcomes for the user.

Remediations to Consider

Add the ability to generate the trade hash with its input parameters to more easily verify what exactly is being signed.

~~Q-1~~ Duplicate code used to generate `TradeApprovalHash`

TOPIC	STATUS	QUALITY IMPACT
Duplicate code	Fixed	Low

--- | --- | | **Impact** | low |

The trade approval hash which is signed and verified in trades is generated in the public view function `getTradeApprovalHash()`, as well as independently in both `_isTradeApprovalValid()` and `_isTradeApprovalValidWithEIP1271Fallback()`. Although the result is the same valid hash, and no

branch calculates it multiple times, it is best practice to reduce duplicate code where possible to prevent issues creeping up if changes are made in the future.

Remediations to Consider

Use `getTradeApprovalHash()` wherever the approval hash is required to prevent duplicate code.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Sapience team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.