Sapien

zokyo

August 8th 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING SAPIEN AI CORP

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: 0 points

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 1 High issue: 1 resolved = 0 points deducted
- 2 Medium issues: 1 resolved and 1 acknowledged = - 3 points deducted
- 3 Low issues: 2 resolved and 1 acknowledged = - 1 point deducted
- 2 Informational issues: 1 resolved and 1 acknowledged = 0 points deducted

Thus, 100 - 3 - 1 = 96

TECHNICAL SUMMARY

This document outlines the overall security of the Sapien AI Corp smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Sapien AI Corp smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Sapien AI Corp put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Sapien AI Corp repository:
Repo: https://github.com/sapien-io/sapien-contracts

Last commit - 5100c6b0ab7ee26f7bb902cf9178608e14953a14

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- ./SapienVault.sol
- ./utils/Constants.sol
- ./SapienRewards.sol
- ./BatchRewards.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Sapien AI Corp smart contract.i. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract/s by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

Sapien AI Corp is a decentralized platform that creates a reputation-based ecosystem for AI contributions using blockchain technology. The platform operates around a native utility token (SAPIEN) with a 1 billion token maximum supply, where users can stake tokens in a sophisticated vault system with flexible lockup periods (30-365 days) to earn reputation multipliers up to 1.50x. The core workflow involves users staking tokens to build reputation, contributing to AI tasks and content, having their work quality-assessed through an EIP-712 signature-based QA system that can impose financial penalties for poor contributions, and earning rewards distributed through cryptographic signature verification to prevent fraud. The platform features a two-phase unstaking mechanism with cooldown periods to prevent gaming of the QA penalty system, integrated treasury management for penalty collection, and batch reward claiming functionality for both native SAPIEN tokens and USDC payments, all designed to incentivize high-quality AI contributions while maintaining platform integrity through economic alignment.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Sapien AI Corp team and the Sapien AI Corp team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Maximum Stake Cap Bypass via increaseAmount May Let Users Exceed Protocol Limits	High	Resolved
2	Users' tokens that have passed the cooldownTime can be penalized unfairly via processQAPenalty(	Medium	Acknowledged
3	Missing Lower-Bound Check in setMaximumStakeAmount May Lead to Staking Freeze (Denial of Service)	Medium	Resolved
4	Unrestricted emergencyWithdraw Parameter in SapienVault May Lead to Unauthorized Token Drain and Accounting Inconsistency	Low	Resolved
5	Lack of Pause Control in batchClaimRewards May Lead to Inconsistent Claim Behavior During Emergencies	Low	Resolved
6	Missing zero address in claimRewardFor function of SapienRewards contract	Low	Acknowledged
7	Loose Lock-up Validation in isValidLockUpPeriod May Lead to Lockup Gaming and Reduced Commitment	Informational	Resolved
8	Centralization could lead to bricking the contracts entirely	Informational	Acknowledged

Maximum Stake Cap Bypass via increaseAmount May Let Users Exceed Protocol Limits

Description:

The validation helper for increaseAmount checks only the additional amount against maximumStakeAmount, not the combined total. Thus, a user with an existing stake can top-up past the cap:

```
function _validateIncreaseAmount(uint256 additionalAmount, UserStake
storage userStake) private view {
    if (additionalAmount == 0) revert InvalidAmount();
            if (additionalAmount > maximumStakeAmount) revert
StakeAmountTooLarge();
    if (userStake.amount == 0) revert NoStakeFound();
    // Missing: userStake.amount + additionalAmount <= maximumStakeAmount
}
```

Impact Scenario:

- maximumStakeAmount = 2 500.
- User stakes 2 000.
- They call increaseAmount(600) → passes (600 < cap).
- Final stake = 2 600 > cap, violating protocol invariants.

Recommendation:

Also enforce the sum check:

Resolved in commit :`9aa946b37ff47e6f1db1b168ff2475b157cbe535`

Users' tokens that have passed the cooldownTime can be penalized unfairly via processQAPenalty()

Description

In the SapienVault contract, the processQAPenalty() function allows the SapienQA contract to levy a penalty on user's stakes by transferring user's percentage of stake to the treasury. This penalty is applied by first calculating the actualPenalty via _calculateApplicablePenalty() which calculates the maximum penalty applicable based on the user's tokens currently staked. This staked amount also includes cooldownAmount so the calculation is done only on the userStake.amount. But the issue is that if a user has some amount of tokens in cooldownAmount that was initiated for unstake via initiateUnstake(), and if these tokens have passed the COOLDOWN_PERIOD, the user's could still lose them if the SapienQA contract calls the processQAPenalty() function.

In short, stakes could be slashed post cooldown which are ready to withdraw but not yet withdrawn by a user.

```
   function _calculateApplicablePenalty(UserStake storage userStake,
uint256 requestedPenalty)
        internal
        view
        returns (uint256)
    {
        // Only use amount as the maximum penalty, since cooldownAmount
is already counted within amount //
      uint256 totalAvailable = userStake.amount;
        return requestedPenalty > totalAvailable ? totalAvailable :
requestedPenalty;
    }
```

For example, let's say Alice has staked 1000 tokens via stake() function for a period of 30 days. Let's say that after the completion of 30 days, she initiates unstaking of 800 of her tokens via the initiateUnstake() function. A week passes by and Alice forgets to withdraw her tokens via unstake() function, which were already available to unstake and claim after 2 days which is currently the COOLDOWN_PERIOD. But as she tries to claim it she finds that she is able to withdraw only 500 tokens, as processQAPenalty() was called which slashed 500 of her tokens, which included the 200 which she has staked, plus the 300 out of 800 which was available to her after the cool down time.

Moreover, the same thing can happen if a user had initiated early unstaking via initiateEarlyUnstake() and those tokens had passed the COOLDOWN_PERIOD. Ideally this should not happen as the users will be losing their funds which were already initiated for unstaking AND had also completed the cooldown time which would be unfair to them.

Impact
The penalties will lead to users not just losing their staked tokens, but also their tokens initiated for unstaking that have sufficiently passed the cool down time. This may also unnecessarily discourage users from staking their tokens if they find that their tokens, that was initiated for unstaking and had passed the cooldown time sufficiently, to be slashed and penalized.

Proof of Concept (PoC): https://gist.github.com/shanzson/6bbfa22440073d9a14ced62e69e0bfb0

Recommendation:
It is advised to prevent levying penalty on the funds of users that have been initiated for unstaking that have sufficiently passed the cool down time. One of the ways that this can be done is by updating the _calculateApplicablePenalty() such that the max penalty levied excludes the funds that have passed the cooldown time but are yet to be withdrawn as follows:

```
  function _calculateApplicablePenalty(UserStake storage userStake,
uint256 requestedPenalty)
        internal
        view
        returns (uint256)
    {
        uint256 totalAvailable = userStake.amount;

        // SECURITY FIX: Protect completed cooldown tokens from penalties
        uint256 protectedAmount = 0;
```

```
        // Check if normal cooldown has completed - protect those tokens
        if (userStake.cooldownStart > 0 &&
            block.timestamp >= userStake.cooldownStart +
Const.COOLDOWN_PERIOD) {
            protectedAmount += userStake.cooldownAmount;
        }

        // Check if early unstake cooldown has completed - protect those
tokens
        if (userStake.earlyUnstakeCooldownStart > 0 &&
            block.timestamp >= userStake.earlyUnstakeCooldownStart +
Const.COOLDOWN_PERIOD) {
            protectedAmount += userStake.earlyUnstakeCooldownAmount;
        }

        // Reduce available amount by protected tokens
        uint256 penalizableAmount = totalAvailable > protectedAmount ?
            totalAvailable - protectedAmount : 0;

        return requestedPenalty > penalizableAmount ? penalizableAmount :
requestedPenalty;
    }
```

Comment: The client said that they were not going to implement this fix because it allowed a user to withdraw the said collateral and bypass quality assurance procedures. The team added that if a user had a stake collateral amount that is not subject to quality assurance, the user might be eligible to access tasks or provide contributions that do not pass quality standards and immediately remove their stake collateral, which would negate quality assurance guarantees.

Missing Lower-Bound Check in setMaximumStakeAmount May Lead to Staking Freeze (Denial of Service)

Description

(SapienVault:setMaximumStakeAmount):
The function setMaximumStakeAmount(uint256 newMaximumStakeAmount) only checks newMaximumStakeAmount != 0 and does not enforce newMaximumStakeAmount >= Const.MINIMUM_STAKE_AMOUNT.

Impact:

If an admin sets the cap below the protocol's minimum stake, all subsequent stake(...) calls revert, freezing new deposits and halting staking activity.

Proof of Concept:

1. Const.MINIMUM_STAKE_AMOUNT is 100 SAPIEN.
2. Admin calls setMaximumStakeAmount(50).
3. User calls stake(100, 30 days) → reverts StakeAmountTooLarge.
4. No new stakes until cap is corrected.

Recommendation:

Add a lower-bound requirement:

```
- if (newMaximumStakeAmount == 0) revert InvalidAmount();
+   if   (newMaximumStakeAmount   <   Const.MINIMUM_STAKE_AMOUNT)   revert
InvalidAmount();
```

Resolved in commit : `06afdfe2214c19f5d251ab1ab42cec98fd618755`

Unrestricted emergencyWithdraw Parameter in SapienVault May Lead to Unauthorized Token Drain and Accounting Inconsistency

Description (SapienVault:emergencyWithdraw):
The function emergencyWithdraw(address token, address to, uint256 amount) allows any ERC-20 token (and ETH) to be withdrawn by an admin without adjusting totalStaked. As a result, even the staking token itself can be removed without updating the vault's accounting.

Impact:
A compromised or malicious admin can drain SAPIEN or other tokens from the vault while leaving totalStaked unchanged, enabling theft and disrupting all staking and reward logic.

Proof of Concept:
1. Vault holds 1 000 SAPIEN, totalStaked == 1000.
2. Admin calls emergencyWithdraw(sapienToken, attacker, 1000).
3. Vault balance becomes 0, but totalStaked remains 1000.
4. All front-ends and reward calculations believe tokens still exist, blocking withdrawals or enabling further exploits.

Recommendation:
Either forbid withdrawing the staking token entirely, or if necessary allow it only with an update to totalStaked:

```
- if (token == address(0)) { ... }
+ if (token == address(sapienToken)) revert InvalidToken();
  if (token == address(0)) {
    // ETH withdraw
  } else {
    IERC20(token).safeTransfer(to, amount);
  }
+ totalStaked -= (token == address(sapienToken)) ? amount : 0;
```

Resolved in commit : `dc31e62adfc2f668cc74afff5aaaaa182ec0f445`

Lack of Pause Control in batchClaimRewards May Lead to Inconsistent Claim Behavior During Emergencies

Description (BatchRewards:batchClaimRewards):
The function batchClaimRewards(...) is not pausable, so emergency pause in underlying reward contracts can cause partial or confusing failures.

Impact:
No global emergency stop for batch claims; if one reward engine is paused, the entire batch reverts or fails unpredictably, causing poor UX and potential fund lockups.

Proof of Concept:
1. SapienRewards is paused.
2. User calls batchClaimRewards(...).
3. First claim reverts (paused), aborting the entire batch—even if other rewards could succeed.

Recommendation:
Make BatchRewards pausable and protect the batch function:

```
+ contract BatchRewards is ReentrancyGuard, Pausable {
+   function pause() external onlyOwner { _pause(); }
+   function unpause() external onlyOwner { _unpause(); }

-  function batchClaimRewards(...) public nonReentrant {
+  function batchClaimRewards(...) public whenNotPaused nonReentrant {
     if (sapienAmount > 0) sapienRewards.claimRewardFor(...);
     if (usdcAmount   > 0) usdcRewards.claimRewardFor(...);
  }
```

Resolved in commit : `819b00fa91263ca0a5fe02ebcfa1c162dfcd43ab`

Missing zero address in claimRewardFor function of SapienRewards contract

Description (BatchRewards:batchClaimRewards):
The claimRewardFor() function of the SapienRewards contract can be called by BATCH_CLAIMER_ROLE to claim a reward on behalf of another user. But there is missing zero address check for user parameter which could lead to a call being made to claim rewards for address(0). Although it is highly unlikely to happen, there is still a possibility that rewards could be sent to a zero address if a valid signature is passed for address(0) as user address as the first parameter to the claimRewardFor() function. This would work only when a burnable token is used as rewardToken with code that explicitly bypasses the SafeERC20's safetransfer() checks.

Impact:
It is highly unlikely to happen with many cases to come true in order for this kindof issue being exploited and is a griefing attack case where the attacker won't actually gain anything, but is still advised to add a zero address check as a best practice.

Recommendation:
Consider adding a zero address check as follows:

```
require(user != address(0))
```

Loose Lock-up Validation in isValidLockUpPeriod May Lead to Lockup Gaming and Reduced Commitment

Description (SapienVault:isValidLockUpPeriod):
The function isValidLockUpPeriod(uint256 lockUpPeriod) returns true for any period between minimum and maximum, rather than restricting to 30, 90, 180, or 365 days.

Impact:
Users can choose marginally longer durations (e.g. 31 days) to approximate the multiplier of a 30-day lockup with shorter commitment, undermining incentive alignment.

Proof of Concept:
1. User stakes 100 tokens with lockUpPeriod = 31 days.
2. They receive ~1.05× multiplier but only wait 31 days.
3. They repeat the cycle every 31 days, avoiding the full 30-day wait in practice.

Recommendation:
Whitelist only intended durations:

```
- return lockUpPeriod >= Const.MINIMUM_LOCKUP_INCREASE && lockUpPeriod <=
Const.LOCKUP_365_DAYS;
+ return lockUpPeriod == 30 days
+     || lockUpPeriod == 90 days
+     || lockUpPeriod == 180 days
+     || lockUpPeriod == 365 days;
```

Client comment: Task availability for tiers is subject to the amount staked. We have decided that discrete lockup tiers is problematic when combining multiple stake amounts and loose tier validation is optimal with less complexity. Will not implement this suggested fix.

Centralization could lead to bricking the contracts entirely

Description

The functions such as processQAPenalty() can be used to heavily penalize the stakes currently staked by users leading to their full staked amounts being lost. Although this is as per the design and should not be a problem as processQAPenalty() can only be called via SapienQA contract due to onlySapienQA, extra care must be taken that this is not changed to another address that is being granted SAPIEN_QA_ROLE accidentally or due to private key loss or compromise of DEFAULT_ADMIN_ROLE address. Compromise of DEFAULT_ADMIN_ROLE can also actually lead to the complete contract being compromised where attacker can upgrade it to any arbitrary code and potentially withdraw funds.

Impact:

It is unlikely to happen, but there still exists a possibility for mishaps to happen such as compromising of the DEFAULT_ADMIN_ROLE that could lead to upgrading the contract to any malicious code and users losing their funds.

Recommendation:

It is highly recommended to use multisig wallet for DEFAULT_ADMIN_ROLE with at least 5/9 or 6/11 configuration.

	./SapienVault.sol ./utils/Constants.sol ./SapienRewards.sol ./BatchRewards.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Sapien AI Corp team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Sapien AI Corp team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo