33AUDITS & CO.

Seraph Audit Report

Introduction

A time-boxed security review of the protocol was done by 33Audit & Company, focusing on the security aspects of the smart contracts. This audit was performed by 33Audits.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any vulnerabilities. Subsequent security reviews, bug bounty programs, and on-chain monitoring are recommended.

About 33 Audits & Company

33Audits LLC is an independent smart contract security researcher company and development group. We conduct audits a as a group of independent auditors with various experience and backgrounds. We have conducted over 15 audits with dozens of vulnerabilities found and are experienced in building and auditing smart contracts. We have over 4 years of Smart Contract development with a focus in Solidity, Rust and Move. Check our previous work here or reach out on X @solidityauditor. A time-boxed security review of the Seraph Daily Check In protocol was done by 33Audit & Company, focusing on the security aspects of the smart contracts.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any vulnerabilities. Subsequent security reviews, bug bounty programs, and on-chain monitoring are recommended.

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Findings Summary

ID	**Title**	**Severity**	**Status**
[M-1]	Unbounded Loop in setAllMessages Could Lead to DOS	Medium	
[M-2]	Single-Step Ownership Transfer Risk	Medium	
[L-1]	Inefficient Storage Pattern in setAllMessages	Low	
[L-2]	Use of require Instead of Custom Errors	Low	

ID	Title	Severity	Status
[L-3]	Missing Events for Critical State Changes	Low	

Medium Risk Findings

M-1: Unbounded Loop in setAllMessages Could Lead to DOS

Description

The `setAllMessages` function contains an unbounded loop that iterates through the input array. If a large array is provided, the function could exceed the block gas limit, causing the transaction to fail.

```
uint256 public constant MAX_MESSAGES = 100;

function setAllMessages(string[] calldata newMessages) public onlyOwner {
    if (newMessages.length > MAX_MESSAGES) {
        revert MaxMessagesExceeded(newMessages.length, MAX_MESSAGES);
    }
    // ... rest of function
}
```

Impact

- Contract functionality could become unusable if too many messages are added
- Owner could accidentally or maliciously make the contract inoperable
- Gas costs increase linearly with array size

Resolution

Add a maximum array length constant and check against it using a custom error. Implement a MAX_MESSAGES constant and revert if the input array exceeds this limit.

M-2: Single-Step Ownership Transfer Risk

Description

The contract inherits from OpenZeppelin's `Ownable` instead of `Ownable2Step`, allowing immediate ownership transfers that could result in permanently locked contracts if transferred to an incorrect address.

```
import "@openzeppelin/contracts/access/Ownable2Step.sol";

contract DailyCheckIn is Ownable2Step {
    // ... rest of contract
}
```

Impact

- Potential permanent loss of contract ownership
- No way to recover if wrong address is provided
- Critical admin functions could become inaccessible

Resolution

Replace `Ownable` with `Ownable2Step` from OpenZeppelin's library to implement a two-step ownership transfer pattern.

Low Risk Findings

L-1: Inefficient Storage Pattern in setAllMessages

Description

The current implementation uses `delete messages` followed by individual pushes, which is gas inefficient compared to direct array assignment.

```
function setAllMessages(string[] calldata newMessages) public onlyOwner {
    messages = newMessages;
}
```

Impact

- Higher gas costs for message updates
- Unnecessary storage operations

Resolution

Use direct array assignment instead of delete followed by pushes.

L-2: Use of require Instead of Custom Errors

Description

The contract uses require statements with string messages instead of custom errors, which is less gas efficient and provides less informative error messages.

```
error IndexOutOfBounds(uint256 index, uint256 length);

function checkIn(uint256 index) public {
    if (index >= messages.length) {
        revert IndexOutOfBounds(index, messages.length);
    }
    // ... rest of function
}
```

Impact

- Higher gas costs for error cases
- Less detailed error information for users
- Larger contract size due to string error messages

Resolution

Replace require statements with custom errors for better gas efficiency and more detailed error reporting.

L-3: Missing Events for Critical State Changes

Description

The `setAllMessages` function modifies contract state without emitting an event, making it difficult to track changes off-chain.

```
event MessagesUpdated(address indexed owner, uint256 messageCount);

function setAllMessages(string[] calldata newMessages) public onlyOwner {
    messages = newMessages;
    emit MessagesUpdated(msg.sender, newMessages.length);
}
```

Impact

- Reduced contract transparency
- Difficulty in tracking message updates
- Poor UX for dapp integrations

Resolution

Add events for message updates to allow proper off-chain tracking of state changes.