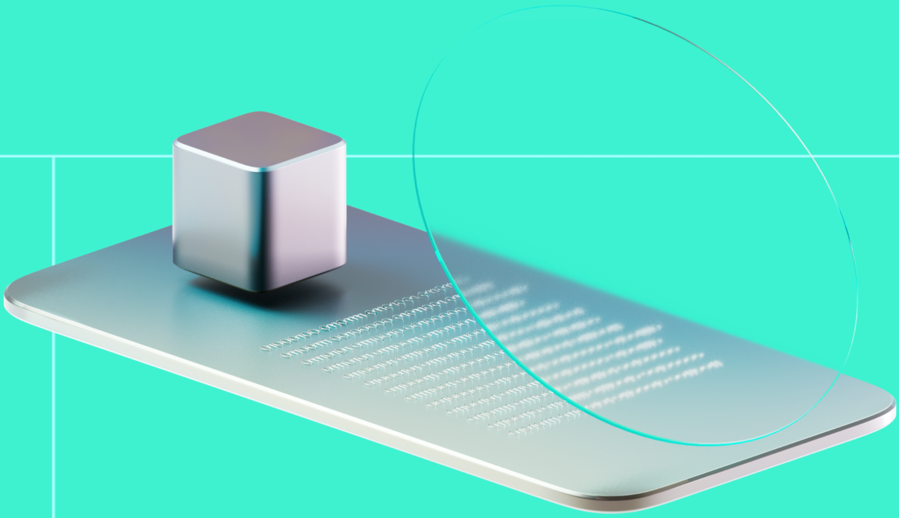
Smart Contract Code Review And Security Analysis Report



Customer: Sock

Date: 5 Dec, 2023

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

We thank Sock for allowing us to conduct a Smart Contract Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

Sock is an ERC-4337 compliant contract implementation, and is designed to allow for self-custody storage of cryptocurrencies in a safe and controlled space.

Platform: EVM

Language: Solidity

Tags: Authorization, ERC-4337

Timeline: 24.11.2023 - 05.12.2023

Methodology: Link

Last review scope

Repository	https://github.com/SockFinance/sock-account
Commit	0e95242dceab8815a44767f6ef2b20a693765e56
Remediation 1	79c848d82a59e9d99d4722b7f939e94c9a95df44
Remediation 2	0ba8d366dc7f386539f7e11037fda8ee7ce90223

View full scope

Audit Summary

10/10	10/10	100%	10/10
Security score	Code quality score	Test coverage	Documentation quality score

Total: 10/10

🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩

The system users should acknowledge all the risks summed up in the risks section of the report.

2	2	0	0
Total Findings	Resolved	Acknowledged	Mitigated

Findings by severity	Findings Number	Resolved	Mitigated	Acknowledged
Critical	0	0	0	0
High	1	1	0	0
Medium	0	0	0	0
Low	1	1	0	0

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Document

Name	Smart Contract Code Review and Security Analysis Report for Sock
Audited By	Arda Usman | SC Lead Auditor at Hacken OÜ Kornel Światłowski| SC Auditor at Hacken OÜ
Approved By	Grzegorz Trawiński | SC Audits Expert at Hacken OÜ
Website	https://www.sock.app/
Changelog	27.11.2023 – Preliminary Report 05.12.1223 – Final Report

Introduction

Hacken OÜ (Consultant) was contracted by Sock (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

Sock is an ERC-4337 compliant contract implementation, and is designed to allow for self-custody storage of cryptocurrencies in a safe and controlled space. It has the following contracts:

- SockRegistryAccessManager - Contract module that provides access control mechanisms over the sock function registry.
- SockFunctionRegistry - A registry contract to manage allowed and disallowed functions and their properties
- SockUserPermissions - Abstract contract module to handle user permissions
- SockOwnable - Provides access control mechanisms with two types of owners - 'owner' and 'sockOwner'. The 'owner' and 'sockOwner' can have distinct rights and the distinction allows functions to be restricted to either of them or both.

Privileged roles

- Owner: Maximal amount of control with the ability to change all other roles within the contract.

- SockOwner: Can execute functions that fall within the Sock Function registry.
- RegistryOwner: Can change the Owner of the contract only if recovery is enabled.

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.

Code quality

The total Code Quality score is **10** out of **10**.

Test coverage

Code coverage of the project is **100.00%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

Security score

As a result of the audit, the code contains **no** issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10.0**. The system users should acknowledge all the risks summed up in the risks section of the report.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Missing Constructor And Assignment Of Admins In SockOwnable

Impact	High
Likelihood	Medium

The *SockOwnable* entails a specialized implementation featuring three administrative roles: *_sockOwner*, *_owner*, and *_recoveryOwner*. However, the *SockOwnable*, *SockRegistryAccessManager* or *SockUserPermissions* contracts lack a *constructor()* and the assignment of these roles, along with the necessary functions for such assignments. For contracts intending to implement *SockOwnable*, the responsibility falls upon them to manually assign these roles. Failure to execute this assignment during deployment renders it impossible afterward. Consequently, default values of the address (0×0) will be assigned to these roles, rendering the ownable functionality unused.

Proof of Concept:

1. *SockUserPermissions* contract is implemented in *ExampleSmartContract*

2. *ExampleSmartContract* is missing an assignment of *_sockOwner*, *_owner*, and *_recoveryOwner* roles.
3. Mentioned roles have a default value of 0×0 and can not be changed.
4. The role related operations will not be able to be performed by the contract creator and the roles will not be assigned to new addresses because the current authorized address is 0×0. This will create a lock situation.

Recommendation: It is recommended to implement a *constructor()* in *SockOwnable,* *SockRegistryAccessManager* or *SockUserPermissions*. and assigning the provided addresses to the admin variables and conducting checks against addresses with the value 0×0.

Path: ./contracts/sock-account/SockOwnable.sol

Found in: 0e95242dceab8815a44767f6ef2b20a693765e56

Status: Fixed (Revised commit: 79c848d8)
Remediation: Constructor with assignments of *_sockOwner* and *_owner* is now added to the *SockOwnable* contract.

Medium

No medium severity issues were found.

Low

L01. Missing Zero Address Validation

Impact	Low
Likelihood	Low

Address parameters are being used without checking against the possibility of 0×0.

This can lead to unwanted external calls to 0×0.

Path: ./contracts/sock-account/SockRegistryAccessManager.sol: _transferSockFunctionRegistry()

./contracts/sock-account/SockOwnable.sol: _transferSockOwnership(), _transferOwnership()

Recommendation: Implement zero address checks.

Found in: 0e95242dceab8815a44767f6ef2b20a693765e56

Status: Fixed (Revised commit: 79c848d8)

Remediation: The zero address check is currently added into the *_transferOwnership()* function. However, absence of zero address checks in the remaining mentioned functions is desired to facilitate the revocation of sock admin.

Informational

I01. Floating Pragma

The project uses floating pragmas ^0.8.17.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Paths: ./contracts/registry/SockFunctionRegistry.sol

./contracts/sock-account/SockUserPermissions.sol

./contracts/sock-account/SockRegistryAccessManager.sol

./contracts/sock-account/SockOwnable.sol

./contracts/interfaces/ISockFunctionRegistry.sol

./contracts/interfaces/ISockUserPermissions.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: 0e95242dceab8815a44767f6ef2b20a693765e56

Status: Fixed (Revised commit: 79c848d8)
Remediation: Pragma is now set to 0.8.17.

I02. Higher Deployment Cost Due To Complex Implementation Of _getFunctionSig()

The present implementation of the *_getFunctionSig()* function contributes to the escalation of deployment costs for the *SockFunctionRegistry* contract. Achieving the same functionality can be accomplished with significantly reduced deployment costs and complexity by implementing the following formula: *return bytes4(func[:4]);*.

```
function _getFunctionSig(bytes calldata func) internal pure returns (bytes4) {
        return bytes4((uint32(uint8(func[0])) << 24) | (uint32(uint8(func[1])) << 16) |
(uint32(uint8(func[2])) << 8) | uint32(uint8(func[3])));
   }
```

Path: ./contracts/registry/SockFunctionRegistry.sol: _getFunctionSig()

Recommendation: Consider using the mentioned formula to lower deployment costs and complexity of the contract.

Found in: 0e95242dceab8815a44767f6ef2b20a693765e56

Status: Fixed (Revised commit: 79c848d8)

Remediation: The suggested solution is used and deployment cost is now reduced.

I03. Gas Inefficient Counter

In Solidity version 0.8 and above, arithmetic operations automatically include checks for underflows and overflows. Although these checks are useful for preventing calculation errors, they consume additional gas, leading to higher transaction costs.

In scenarios where underflows and overflows are not possible, the additional checks introduced by Solidity 0.8 can be bypassed to save gas. This can be done by placing the increment operation inside an unchecked{} block. This block enables developers to perform arithmetic operations without the automatic underflow and overflow checks, thus conserving gas when they are not needed.

Path: ./contracts/sock-account/SockUserPermissions.sol:
initializeUserPermissions()

Recommendation: To improve gas efficiency, consider placing the post-iteration increment operation at the end of the loop inside an *unchecked{}* code block. This avoids the standard overflow checks, thereby conserving gas. Ensure that this technique is only employed in cases where an overflow is not possible.

Found in: 0e95242dceab8815a44767f6ef2b20a693765e56

Status: Fixed (Revised commit: 0ba8d366dc7f386539f7e11037fda8ee7ce90223)

Remediation: The suggested solution is used and deployment cost is now reduced.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope details

Repository	https://github.com/SockFinance/sock-account
Commit	0e95242dceab8815a44767f6ef2b20a693765e56
Whitepaper	-
Requirements	Link
Technical Requirements	Link

Contracts in Scope

./contracts/registry/SockFunctionRegistry.sol
./contracts/sock-account/SockUserPermissions.sol
./contracts/sock-account/SockRegistryAccessManager.sol
./contracts/sock-account/SockOwnable.sol
./contracts/interfaces/ISockFunctionRegistry.sol
./contracts/interfaces/ISockUserPermissions.sol

Scope details

Repository	https://github.com/SockFinance/sock-account
Commit	79c848d82a59e9d99d4722b7f939e94c9a95df44
Whitepaper	-
Requirements	Link
Technical Requirements	Link

Contracts in Scope

./contracts/registry/SockFunctionRegistry.sol
./contracts/sock-account/SockUserPermissions.sol
./contracts/sock-account/SockRegistryAccessManager.sol
./contracts/sock-account/SockOwnable.sol
./contracts/interfaces/ISockFunctionRegistry.sol
./contracts/interfaces/ISockUserPermissions.sol

Scope details

Repository	https://github.com/SockFinance/sock-account

Commit	0ba8d366dc7f386539f7e11037fda8ee7ce90223
Whitepaper	-
Requirements	Link
Technical Requirements	Link

Contracts in Scope

./contracts/registry/SockFunctionRegistry.sol
./contracts/sock-account/SockUserPermissions.sol
./contracts/sock-account/SockRegistryAccessManager.sol
./contracts/sock-account/SockOwnable.sol
./contracts/interfaces/ISockFunctionRegistry.sol
./contracts/interfaces/ISockUserPermissions.sol