# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: Sock
**Date**: 13 Oct, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io



## Document

| | |
|---|---|
| **Name** | Smart Contract Code Review and Security Analysis Report for Sock |
| **Approved By** | Przemyslaw Swiatowiec \| Lead Solidity SC Auditor at Hacken OÜ |
| **Auditors** | David Camps Novi \| SC Auditor at Hacken OÜ |
| **Tags** | Signatures; Staking; Proxy; Factory; ERC4337; Account Abstraction; |
| **Platform** | EVM |
| **Language** | Solidity |
| **Methodology** | [Link] |
| **Website** | https://www.sock.app/ |
| **Changelog** | 27.09.2023 - Initial Review<br>13.10.2023 - Second Review |

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

# Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

# 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 ERC4337 compliant contract implementation designed to allow self-custody storage of cryptocurrencies in a safe and controlled space with the following contracts:

- ***SockAccountFactory*** – creates and manages instances of SockAccount via proxy pattern.
- ***SockAccount*** – ERC4337 compliant implementation, customized for the Sock ecosystem.
- ***SockOwnable*** – provides access control mechanisms to differentiate two types of owners: *owner* and *sockOwner*.
- ***SockRegistryAccessManager*** – extends the functionality of SockOwnable, providing access control mechanisms for a specific registry named sock function registry. The sock owner has the exclusive right to change the referenced sock function registry within the system.
- ***SockRegistryImplementer*** – builds upon the functionality provided by SockRegistryAccessManager. Ensures that functions are allowed to be executed based on the rules defined in the sock function registry.
- ***SockFunctionRegistry*** – management tool for keeping track of specific allowed functions.
- ***UniswapV3SockProxy*** – gateway for interacting with UniswapV3, facilitating seamless ETH and ERC20 swaps.
- ***ERC20SockProxy*** – proxy that offers direct interactions with ERC20 token methods.
- ***SockFeeManager*** – converts fee tokens into a designated cashout token through Uniswap V3 and sends them to a specific cashout address.
- ***SockFeeWhitelist*** – manages a whitelist of users who are exempt from "sock fees".

## Privileged roles

- **SockAccount Owner**: Has the ability to whitelist functions that can be executed from *SockAccount* by the account owner and *SockOwner*. Add, unlock, and withdraw stakes in ERC4337 flow.
- **SockOwner**: Can execute whitelisted functions (defined by *SockAccount* owner in the *SockFunctionRegistry*).
- **RecoverOwner**: Can change the owner of the contract if recovery is enabled.

- **Protocol Owner**: Set fees generated by using the *UniswapV3SockProxy* contract. Change fees *cashOut* tokens, withdraw fees, and Whitelist addresses that fees do not apply to.

# 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.
- NatSpec is sufficient.
- Description of the development environment is present.

## Code quality

The total Code Quality score is **9** out of **10**.

- Solidity Style Guides are violated.
- Floating Pragmas are present.
- There are missing validations.
- Inefficient Gas model is present.
- The development environment is configured.

## Test coverage

Code coverage of the project is **93.44%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

## Security score

As a result of the audit, the code contains **2** low severity 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: **9.5**. The system users should acknowledge all the risks summed up in the risks section of the report.

1 2 3 4 5 6 7 8 9 10

The final score

***Table. The distribution of issues during the audit***

| Review date | Low | Medium | High | Critical |
|---|---|---|---|---|
| 27 September 2023 | 5 | 4 | 3 | 2 |
| 13 October 2023 | 2 | 0 | 0 | 0 |

## Risks

- Incorrect registry management: The SockAccount owner (protocol user) is responsible for whitelisting operations that can be performed by *SockOwner*. It is up to the user to manage the risk of whitelisting proper operations and smart contracts using the least privilege principle and choose a trusted *SockOwner* operator.
- Insecure ERC4337 infrastructure: The *SockAccount* relies on ERC4337 infrastructure provided by an external company (Stackup). Protocol users should verify if the *Entrypoint* contract assigned to *SockAccount* is secure. Malicious *Entrypoint* contracts may lead to unexpected results, for example, incorrect signature replay attacks in the case of incorrect *nonce* implementation.
- Roles management: The *SockAccount* is a self-custody account. Users are responsible for securing *owner* and *recoverOwner* private keys.
- UUPS proxy: The *SockAccount* contract is implemented using UUPS proxy, which allows contract code to change. It is necessary to secure the upgrade process and verify each contract change.

## Findings

### ■■■■ Critical

#### C01. Missing Access Control in UUPS Upgradable Pattern Leads to SockAccount Takeover

| | |
|---|---|
| **Impact** | High |
| **Likelihood** | High |

Due to the lack of access control in *_authorizeUpgrade*, an external account can upgrade the implementation of the *SockAccount* contract to a malicious one and steal all funds.

What is more, to comply with OpenZeppelin's UUPSUpgradeable contract, it is necessary to initialize *UUPSUpgradeable*, by calling *__UUPSUpgradeable_init()* within the *_initialize()* function of the *SockAccount* contract.

**Path:** ./contracts/sock-account/SockAccount.sol: _initialize(), _authorizeUpgrade().

**Recommendation**: Initialize the UUPSUpgradeable contract and add access control to *_authorizeUpgrade()*.

**Found in:** 8009b2e

**Status:** Fixed (Revised commit: 6616162)

**Remediation**: Proper access control was introduced to *_authorizeUpgrade* - only owner can upgrade the *SockAccount* contract.

#### C02. SockOwner Role Can Hijack SockAccount

| | |
|---|---|
| **Impact** | High |
| **Likelihood** | High |

The main goal of the system is to allow users to delegate the predefined actions to *sockOwner* role. Users can do that by whitelisting function signature and smart contract that can be invoked using *SockFunctionRegistry* contract.

It was observed that *sockOwner* can swap the *SockFunctionRegistry* address used in user *SockAccount*. Consider following scenario:

1. A user deploys his own *SockAccount* and allows *SockOwner* to only transfer and set *allowance* for USDC tokens.
2. *SockOwner* can create malicious *SockFunctionRegistry* that allows him to transfer other tokens stored in *SockAccount*.

3. Then using *setSockFunctionRegistry* in *SockRegistryAccessManager*, *SockOwner* can change an existing registry to a malicious one.
4. As a result, *SockOwner* can whitelist any actions in a user's *SockAccount* and steal all funds.

**Path:** ./contracts/sock-account/SockFunctionRegistry.sol: setSockFunctionRegistry(),

**Recommendation**: Only users (*SockAccount* *owner*) should be allowed to modify *SockFunctionRegistry* instance address.

**Found in:** 8009b2e

**Status:** Fixed (Revised commit: 6616162)

**Remediation**: Proper access control was introduced to *setSockFunctionRegistry()* only the SockAccount owner can change address to the *SockFunctionRegistry*.

## High

### H01. Temporary Freezing of Funds due to Missing Parameter Update

| | |
|---|---|
| **Impact** | High |
| **Likelihood** | Medium |

The function *addAllowedFunction()* does not update *_allowedFunctions*' *payable* status.

As a result, the account owner is not able to call *payable* functions, which seriously reduces the number of operations that can be performed by *SockAccount*. For example, the account owner is not able to withdraw any native ETH stored in *SockAccount*. To withdraw native ETH user has to:

1. Go to the registry and set *sockOwner* to themself.
2. Allow *sockOwner* to withdraw (call payable function)
3. Withdraw as *sockOwner*
4. Change *sockOwner* to the old one.

**Path:** ./contracts/registry/SockFunctionRegistry.sol: addAllowedFunction().

**Recommendation**: Update the *payable* parameter in *addAllowedFunction().*

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: The *SockAccount* owner should execute transactions using newly added *executeOwner()* function. All functions regarding owner permissions including *addAllowedFunction()* were removed from the *SockFunctionRegistry*.

### H02. Requirements Violation - ERC20SockProxy Cannot be Used as Proxy

| | |
|---|---|
| **Impact** | Medium |
| **Likelihood** | High |

According to the requirements, everybody can interact with the standalone *ERC20SockProxy* contract to transfer/approve their tokens according to the token address given. However, the implementation will not execute that since the transfer function transfers tokens from inside the *ERC20SockProxy* contract and not from the users themselves.

The function implementation requires the token holders to first transfer their tokens to the *ERC20SockProxy* contract. If a malicious user calls transfer at this point before the original token holder, they can effectively withdraw the funds, so it is not recommended for any user to interact with the contract in this way.

**Path:** ./contracts/proxies/ERC20SockProxy.sol

**Recommendation**: Review the requirements for the functionality of this contract. Either update the documentation or redesign the contract functionality.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: The *ERC20SockProxy* was removed.

### H03. Signature Replay Attack

| | |
|---|---|
| **Impact** | High |
| **Likelihood** | Medium |

The Sock project allows calls to *execute()* from *EntryPoint*, the *owner* or the *sockOwner*. If the function is called from the *owner* or the *sockOwner*, the *nonce* value is not updated and the function becomes vulnerable to signature replay attacks.

The *execute()* function should only be called by the *EntryPoint* contract, following EIP-4337.

However, a new flow form should be created to be called from the owner and *sockOwner*. This new flow should implement the EIP-712 standard for signature validation.

**Path:** ./contracts/sock-account/SockAccount.sol: execute().

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

**Recommendation**: Follow the EIP-4337 specification and create additional flow to allow execution for owner and sockOwner that follow EIP-712.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: The *SockAccount* owner should execute transactions using newly added *executeOwner()* function without extra signature validation (owner check). The *sockOwner* can execute transactions only using *EntryPoint* with nonce update implemented.

## Medium

### M01. Best Practice Violation; Usage of SafeERC20

| | |
|---|---|
| **Impact** | High |
| **Likelihood** | Low |

The *transfer()* function of the *ERC20SockProxy.sol* contract checks the *return* value of the token transfers manually and does not use *SafeERC20* library for checking the result of ERC20 token transfers.

Some tokens may not follow ERC20 standard and may not *return* *false* in case of transfer failure or they might not *return* any value at all. An example for such a type of token would be the BNB token.

This may lead to unexpected behavior if the interacted token is not ERC20 compliant.

**Path:** ./contracts/proxies/ERC20SockProxy.sol : transfer()

**Recommendation**: Use SafeERC20 library to interact with tokens safely.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: The Uniswap *TransferHelper* library is used, which implements safe transfer features.

### M02. Unlimited Parameter Allows Abusive Fees

| | |
|---|---|
| **Impact** | High |
| **Likelihood** | Low |

In *setSockFee()*, the *_sockFee* can be set without limits.

Thus, such a percentage *fee* can reach 100% and become abusive. The percentage value could exceed 100%, which would cause *underflow* and cause the *_deductSockFee()* function to revert.

If the *_deductSockFee* function reverts, all swap operations in the *UniswapV3SockProxy* contract will fail.

**Path:** ./contracts/proxies/UniswapV3SockProxy.sol: setSockFee().

**Recommendation**: Add limits to the settable fee.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: Limit of maximum 3% fee is implemented.

### M03. Risk of Incorrect Slippage During Swap

| | |
|---|---|
| **Impact** | High |
| **Likelihood** | Low |

In *_atteptCashOut()*, the *fees* can be withdrawn via *swapRouter*.

To calculate the amount of tokens to withdraw, the method uses the *balance* of the contract:

```
uint256 balance = cashOutParams.tokenIn.balanceOf(address(this));
ISwapRouter.ExactInputSingleParams({
      ...
      amountIn: balance,
      ...});
```

For such *amountIn*, a minimum amount of tokens is set as *amountOutMinimum: cashoutParams.amountOutMinimum* in order to define a reasonable *slippage*.

However, if the token *balance* increases after *amountOutMin* is calculated off-chain and inputted in *_attemptCashOut()*, the *slippage* would dramatically increase above desired.

**Path:** ./contracts/sock-infra/SockFeeManager.sol: _attemptCashOut().

**Recommendation**: Set *amountIn* manually instead of relying on the contract balance.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: The *sqrtPriceLimitX96* parameter was introduced to the *UniswapV3SockProxy* swap functions and amountOutMinimum parameter in the *_attemptCashOut()* to protect against slippage during swap.

### M04. Fees Cannot be Cashed Out if Fee Token and Cash Out Token Are the Same

| | |
|---|---|
| **Impact** | Medium |
| **Likelihood** | Medium |

In the *SockFeeManager* contract, all fees are cashed out using Uniswap. There is no functionality to directly withdraw tokens.

As a result, it is not possible to withdraw fees collected in cash out token as Uniswap swap operation reverts if the *tokenIn* parameter and *tokenOut* parameter are the same (Uniswap does not allow the creation of pools with the same token):

1. Protocol fee cash out token is USDC.
2. User performs an operation that swaps USDC for WETH using *UniswapV3SockProxy.* Contract deducts X USDC fees for the protocol.
3. Fees manager cannot withdraw USDC fees as collected fees token is the same as cash out token and *_attemptCashOut* is reverted because of Uniswap error.

**Path:** ./contracts/sock-infra/SockFeeManager.sol : _attemptCashOut()

**Recommendation**: Implement a way to directly withdraw fees without using a swapping protocol if the collected fees token is the same as cash out token.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: The *transferCashOutToken* was introduced to directly withdraw fees denominated in the *cashOut* token.

## Low

### L01. Missing Zero Address Validation

| | |
|---|---|
| **Impact** | Low |
| **Likelihood** | Low |

Address parameters are being used without checking against the possibility of *0x0*.

This can lead to unwanted external calls to *0x0*.

**Paths:**
./contracts/sock-infra/SockFeeManager.sol: constructor(), changeCashOutToken().
./contracts/sock-account/SockRegistryAccessManager.sol: setSockFunctionRegistry().
./contracts/sock-account/SockOwnable.sol: transferSockOwnership(), transferRecoveryOwnership().
./contracts/sock-account/SockAccountFactory.sol: constructor(), createAccount().
./contracts/proxies/UniswapV3SockProxy.sol: constructor(), transferSockFeeRecipient().
./contracts/sock-account/SockAccount.sol: constructor(), execute(), executeBatch(), withdrawDepositTo(), _initialize(), _authorizeUpgrade().

**Recommendation**: Implement zero address checks.

**Found in:** 8009b2e

**Status**: Reported

**Remediation**: No changes regarding missing validation against zero addresses.

### L02. Unused Imports

| | |
|---|---|
| **Impact** | Low |
| **Likelihood** | Medium |

The contract *SockFeeWhitelist* imports the unused contracts *SafeERC20*, *IERC20*, *TransferHelper* and *ISwapRouter*. However, they are not used, incrementing the deployment cost of *SockFeeWhitelist*.

**Paths:**
./contracts/sock-infra/SockFeeWhitelist.sol

**Recommendation**: Remove the unused imports.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: The unused imports were removed.

### L03. Improper Event Data Emission

| | |
|---|---|
| **Impact** | Low |
| **Likelihood** | Medium |

The function *_transferRecoveryOwnership()* emits the *event RecoveryOwnershipTransferred()* with the same data twice instead of the old and new *owner*.

**Paths:**
./contracts/sock-account/SockOwnable.sol:
_transferRecoveryOwnership().

**Recommendation**: Use a *memory* variable to emit the old *_recoveryOwner*.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: The event *RecoveryOwnershipTransferred()* emits correct data.

### L04. Inefficient Gas Model due to Missing Require Check

| | |
|---|---|
| **Impact** | Low |
| **Likelihood** | Low |

When calling *_requireOnlyAllowedFunctions()*, *sockFunctionRegistry* is checked against *address(0)* but it is not reverted in such cases.

As a consequence, there is an unnecessary expense in terms of Gas.

**Path:** ./contracts/sock-account/SockRegistryImplementer.sol:
_requireOnlyAllowedFunctions().

**Recommendation**: It is recommended to revert the function if *sockFunctionRegistry == address(0)*.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: The *_requireOnlyAllowedFunctions()* reverts if *sockFunctionRegistry* is a zero address.

### L05. SockAccount Entrypoint Contract Address Is Set by Sock Team

| | |
|---|---|
| **Impact** | Low |
| **Likelihood** | Low |

When initializing the *SockAccount* contract using *SockAccountFactory*, the *entryPoint* address is predefined by the Sock team.

*EntryPoint* contract plays a crucial role in EIP-4773 flow. Setting an incorrect or malicious *entryPoint* could lead to catastrophic results. Sock team is declaring to use the [Stackup](Stackup) infrastructure. However, as *entryPoint* is set dynamically during *SockAccountFactory* deployment, it's not possible in the audit process to verify that the correct address would be used.

**Path:** ./contracts/sock-account/SockAccount.sol

**Recommendation**: To increase decentralization of the system and increase users trust, it is recommended to allow users to set the *entryPoint* address during initialization of *SockAccount* in *SockAccountFactory*. Alternatively, hardcode the correct *entryPoint* address in the *SockAccountFactory*.

**Found in:** 8009b2e

**Status**: Reported

**Remediation**: No changes regarding *entryPoint* address were observed.

## Informational

### I01. Solidity Style Guides Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be ordered and grouped by their *visibility* as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)
- External functions
- Public functions
- Internal functions
- Private functions

Within each grouping, *view* and *pure* functions should be placed at the end.

**Paths:**
./contracts/sock-account/SockAccount.sol
./contracts/sock-account/SockOwnable.sol
./contracts/sock-account/SockRegistryAccessManager.sol
./contracts/sock-infra/SockFeeManager.sol

**Recommendation**: Consistent adherence to the official *Solidity style guide* is recommended.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: Contracts were refactored to match the official *Solidity style guide.*

### I02. Missing Events for Critical Value Updates

*Events* should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

**Paths:**
./contracts/sock-infra/SockFeeManager.sol: changeCashOutToken(), constructor() → _cashOutToken.
./contracts/sock-infra/SockFeeWhitelist.sol: addAllowedAddresses(), removeAllowedAddresses().
./contracts/proxies/UniswapV3SockProxy.sol: constructor() → _sockFee, setSockFee().

**Recommendation**: Consider emitting *events* in said functions.

**Found in:** 8009b2e

**Status**: Reported (Revised commit: 6616162)

**Remediation**: Events were added to *changeCashOutToken()* and *setSockFee()* function. However, events in *addAllowedAddresses()* and *removeAllowedAddresses()* are still missing.

### I03. Public Functions That Should Be External

Functions that are meant to be exclusively invoked from external sources should be designated as *external* rather than *public*. This is essential to enhance both the Gas efficiency and the overall security of the contract.

**Paths:**
./contracts/sock-account/SockRegistryAccessManager.sol: sockFunctionRegistry(), setSockFunctionRegistry().
./contracts/sock-account/SockOwnable.sol: transferSockOwnership(), transferOwnership(), transferRecoveryOwnership().
./contracts/sock-account/SockAccountFactory.sol: createAccount().
./contracts/registry/SockFunctionRegistry.sol: addAllowedFunction(), addAllowedSockFunction(), removeAllowedFunction(), removeAllowedSockFunction().
./contracts/proxies/ERC20SockProxy.sol: allowance(), approve().
./contracts/sock-account/SockAccount.sol: transferSockOwnership().

**Recommendation**: Consider updating functions which are exclusively utilized by external entities from their current *public* visibility to *external* visibility.

**Found in:** 8009b2e

**Status**: Reported (Revised commit: 6616162)

**Remediation**: Following functions were still not marked as *external*:

- sockFunctionRegistry()
- transferRecoveryOwnership()
- transferSockOwnership() (in SockAccount.sol)

### I04. NatSpec Contradiction in cashOut Function

The *NatSpec* provided for the *cashOut()* function does not correspond to its functionality.

This inconsistency can cause confusion and make it harder for auditors and developers to understand the code. Additionally, even small inconsistencies can accumulate over time and make the codebase harder to maintain.

**Paths:**
./contracts/sock-infra/SockFeeManager.sol: cashOut().

**Recommendation**: Update the *NatSpec* to match the method's functionality.

**Found in:** 8009b2e

**Status**: Fixed (Revised commit: 6616162)

**Remediation**: NatSpec for the *cashOut()* function was fixed.

### I05. Floating Pragma

As stated in SWC-103, contracts should be deployed with the same *compiler* version and flags that they have been tested with thoroughly. Locking the *pragma* helps to ensure that contracts do not accidentally get deployed using, for example, an outdated *compiler* version that might introduce bugs that affect the contract system negatively.

Some contracts use *Solidity 0.8.18* features, such as mapping key/values names and will not be compatible with previous versions.

**Paths:**
./contracts/*.sol

**Recommendation**: Lock the *pragma* version in all contracts.

**Found in:** 8009b2e

**Status**: Reported

**Remediation**: No changes regarding floating pragma were observed.

# 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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

# 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 won't have a significant impact on code execution, don’t 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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

## Appendix 2. Scope

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

### Initial review scope

| | |
|---|---|
| **Repository** | https://github.com/SockFinance/sock-account |
| **Commit** | 8009b2e |
| **Whitepaper** | - |
| **Requirements** | Link |
| **Technical Requirements** | Link |
| **Contracts** | File: dependencies/callback/TokenCallbackHandler.sol<br>SHA3: fea222523a4ea48ed7a7aaa2c6794c741c895580fbbb7d426f98ebfca68f6cce<br><br>File: dependencies/core/BaseAccount.sol<br>SHA3: a60d1021c129e4afc2958d4c323d951e03f7de7bf17a65c19e07a29c9030b23c<br><br>File: dependencies/core/BasePaymaster.sol<br>SHA3: f51b693232ebdb335c8f5ae60cc6de107aa06c5a8402cdb029e22907032b5a6e<br><br>File: dependencies/core/EntryPoint.sol<br>SHA3: 322ba9499c07230834bad57ea8afa21e2990ee2c536b00049c62515add1d089a<br><br>File: dependencies/core/Helpers.sol<br>SHA3: c2b71e1bf5b964260af7a1ff314921a9bd12fba114a5a7e92662ed1197372689<br><br>File: dependencies/core/NonceManager.sol<br>SHA3: ef7045540db39ec973cdc845f658a937a22239f8e620e44bc56831ebffc552c9<br><br>File: dependencies/core/SenderCreator.sol<br>SHA3: 29e94632b90139e9ab53a9377afbd866c23f57f9331aec2138b67e2a5f4e44b6<br><br>File: dependencies/core/StakeManager.sol<br>SHA3: 6d190fe5b33840968e8768b8d296de0abf636b547fa28768360edf532042620c<br><br>File: dependencies/interfaces/IAccount.sol<br>SHA3: ab8890365269704fe4a40d7390f17963d3cb9a54595f7b1b46b8ae16a505aef3<br><br>File: dependencies/interfaces/IAggregator.sol<br>SHA3: 5b1a4877c8c368eb611818d615c5334f119a9634e0516448108c524c8a712638<br><br>File: dependencies/interfaces/IEntryPoint.sol<br>SHA3: e1873d9ddd84235b20f20adb2fa92047fd884bed8cb52efbb8618be139207bcc<br><br>File: dependencies/interfaces/INonceManager.sol<br>SHA3: 83400003c207d7a80d88cd4860361d441de8098c61a452989250044454818 3be<br><br>File: dependencies/interfaces/IPaymaster.sol<br>SHA3: 597b3f407f83747d367661723f4de979d74cb102826893146a62216d9d9b7b89<br><br>File: dependencies/interfaces/IStakeManager.sol<br>SHA3: 8c3bed35eea885979a5a07dd391332c036c1b435cfe4d3dc5f200cd5cb9c89d2<br><br>File: dependencies/libraries/UserOperation.sol |

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

| | |
|---|---|
| | SHA3: c32ce4a6506a9a49a4efeaf021bb88477eebb9d24550e33787169361e955bbb2 |
| | File: dependencies/utils/Exec.sol<br>SHA3: be1350248e4c3a3c927ed43312f081daeec462a5613f22310a9d5b24159bc6f7 |
| | File: interfaces/IERC20SockProxy.sol<br>SHA3: 46ea84cf906f9f978a856aca5a95f69ff7799f2f98abc9f5439496fff2141c2f |
| | File: interfaces/ISockFeeManager.sol<br>SHA3: 21bd4d806fa7e2e9e27bc80dfe00c7e05d2843a39ec97b11a7d8843209dca18b |
| | File: interfaces/ISockFeeWhitelist.sol<br>SHA3: 9606570d37144faeaa3d874ab0209494419590301d6cdbcbba10a2fe0c8b2d91 |
| | File: interfaces/ISockFunctionRegistry.sol<br>SHA3: 7c50f6e02988a9413552fc2ce515412c4a685d2f5d878553802fe3d8870657f6 |
| | File: interfaces/IUniswapV3SockProxy.sol<br>SHA3: ddafd7c3c7a2e4c890f0a703b17df81037657662781fba2fdaaf2e7dc834544b |
| | File: interfaces/IWeth9.sol<br>SHA3: 2ede2091fd570df556758974a2e3f019d1889b2a15eddf80480199c727ee4cd9 |
| | File: proxies/ERC20SockProxy.sol<br>SHA3: 317ac08d512a086a55929636961d45e33283159a693ef53c337340635cd17a5c |
| | File: proxies/UniswapV3SockProxy.sol<br>SHA3: 50250d4d27e2a4f4fe45ec8c0ca4f9f7c6b199fd61e6cd867236282cfe5f9866 |
| | File: registry/SockFunctionRegistry.sol<br>SHA3: a087876d8a39e910261d658588f1e4f96f0f593d23fbb2d516b674239aa63612 |
| | File: sock-account/SockAccount.sol<br>SHA3: fd50947e02af963e64130e87554b97099e9d36a74b716c93a4ac379271d42f4d |
| | File: sock-account/SockAccountFactory.sol<br>SHA3: 142929144aae17f79cb4b217ed955f99780a414c5a7a41a3ed93676f711f34dc |
| | File: sock-account/SockOwnable.sol<br>SHA3: ab62483821b0162f1d0704d768a6925df5677176e77abbac042c6a2eb08a79e0 |
| | File: sock-account/SockRegistryAccessManager.sol<br>SHA3: 56253078e72e33e9eb8eb62d5587834942aaf3b05b3c143879246b195ab2f219 |
| | File: sock-account/SockRegistryImplementer.sol<br>SHA3: 95011a8affdd4518eb2383e211bbad37fbe467aee069ee1db19b27f123417bcd |
| | File: sock-infra/SockFeeManager.sol<br>SHA3: 850c98ae8985cdfcb8f38556de721e81841d54ab59ca515d683d0cf2f5b5d595 |
| | File: sock-infra/SockFeeWhitelist.sol<br>SHA3: 0da3f67e9df2df8fcec03ea6e75623456b36f3a029f3a9eb7b3e25e3b04036a4 |