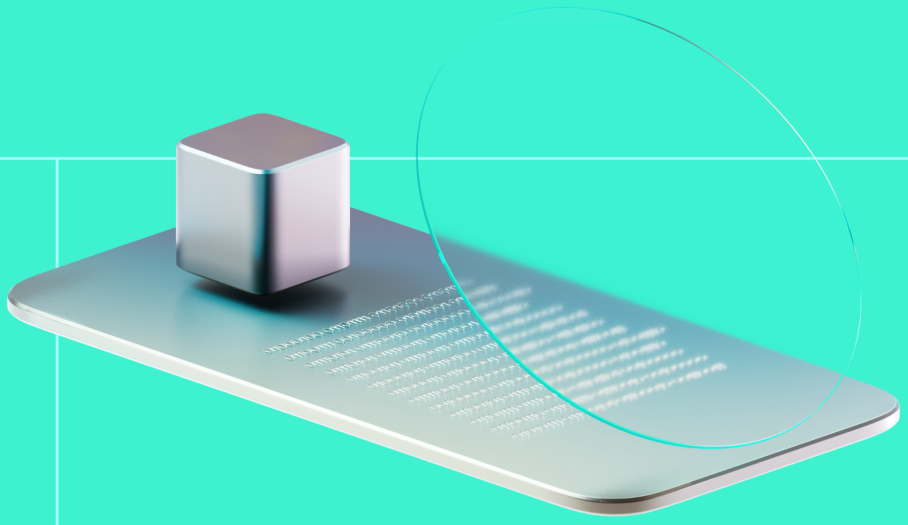
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: VeChain

Date: 20 Dec, 2023

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

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

VeChain Account Abstraction is an implementation of ERC-4337, forked from Eth-Infinitism.

Platform: EVM

Language: Solidity

Tags: ERC-4337

Timeline: 28.11.2023 - 20.12.2023

Methodology: Link

Last review scope

Repository	https://github.com/vechainfoundation/account-abstraction
Commit	bc2a9ec

View full scope

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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

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.

3	3	0	0
Total Findings	Resolved	Acknowledged	Mitigated

Findings by severity	Findings Number	Resolved	Mitigated	Acknowledged
Critical	0	0	0	0
High	0	0	0	0
Medium	1	1	0	0
Low	2	2	0	0

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for VeChain
Audited By	Niccolò Pozzolini | SC Lead Auditor at Hacken OÜ Maksym Fedorenko| SC Auditor at Hacken OÜ
Approved By	Przemysław Świątowiec | SC Audits Expert at Hacken OÜ
Website	https://www.vechain.org
Changelog	12.12.2023 – Preliminary Report 20.12.2023 – First Remediation

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Introduction

Hacken OÜ (Consultant) was contracted by VeChain (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

VeChain Account Abstraction is an implementation of ERC-4337, forked from Eth-Infinitism.

ERC-4337 is a standard for account abstraction in Ethereum, allowing users to interact with the blockchain without holding Ether for transaction fees. It introduces the concept of "bundlers" and "Smart Contract Wallets" (SCWs) to facilitate transactions, enabling more flexible and user-friendly wallet operations and smart contract interactions.

Privileged roles

The owner of *BasePaymaster* can

- Stake VTHO tokens on the EntryPoint through the *addStake()* function;
- Unlock the staked VTHO tokens through the *unlockStake()* function, in order to withdraw them;
- Withdraw the staked VTHO tokens through the *withdrawTo()* and *withdrawStake()* functions.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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

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**.

- The development environment is configured.

Security score

As a result of the audit, the code contains no security 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**.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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

Findings

Critical

No critical severity issues were found.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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

High

No high severity issues were found.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Medium

M01. Lack of Penalty for Reserving Redundant Gas

Impact	Medium
Likelihood	Medium

The *EntryPoint.handleOps()* takes care of handling each user operation of the bundle. User operations specify the amount of Gas to be reserved through the field *UserOp.callGasLimit*. The excess amount of pre-charged Gas is then refunded to the user: *refund = opInfo.prefund - actualGasCost*.

User operations are able to reserve large parts of the Gas space in the bundle and leave it unused, preventing the bundler from including other *UserOperations*.

Path: ./contracts/core/EntryPoint.sol : _postExecution()

Recommendation: To avoid such behavior, a penalty proportional to the amount of reserved and unused Gas should be applied: *unusedGas = UserOp.callGasLimit - executionGasUsed*. An example implementation can be found in EthInfinitsm's AccountAbstraction project.

Found in: 131c057

Status: Fixed (Revised commit: 644caed)

Remediation: A 10% penalty for reserved and unused gas is implemented.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Low

L01. Possible Execution Reversion Due to the Revert Reason of the Wrong Size

Impact	Medium
Likelihood	Low

The *EntryPoint* contract takes care of executing user operations through the function *_executeUserOp()*. This function calls *innerHandleOp()* which revert is handled through a try/catch, where *returndatacopy()* is used to fetch the 32 bytes revert reason.

This might lead to the unexpected reversion calling the *returndatacopy* function, if the revert reason is shorter than 32 bytes.

Path: ./contracts/core/EntryPoint.sol : _executeUserOp()

Recommendation: Before calling *returndatacopy(0, 0, 32)*, *returndatasize()* should be checked to be 32 bytes.

An example implementation can be found in EthInfinitsm's AccountAbstraction project.

Found in: 131c057

Status: Fixed (Revised commit: 74faedf)

Remediation: The return data size is now checked to be 32 bytes before being copied.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

L02. Inconsistent User Operation Hash Generation

Impact	Medium
Likelihood	Low

The *getUserOpHash* function during the parameters encoding modifies the chain id with *& 0xFF* operation. Due to this operation, the original *chain ID* is truncated to 8 bits, so different *chain IDs* might result in the same value after such an operation.

For example: *100009 & 0xFF = 169*

This leads to the inability to compute the same *UserOpHash* off-chain, which might result in inconsistencies on the application level.

Path: ./contracts/core/EntryPoint.sol : getUserOpHash()

Recommendation: Remove the bitwise AND boolean operation applied to the *block.chainId* during User Operation hash generation.

Found in: 131c057

Status: Fixed (Revised commit: 277fc77)

Remediation: The bitmask previously applied to the *chainId* was removed.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Informational

I01. Misplaced BeforeExecution() Event

BeforeExecution() event is used to track the user operations handling flow. It should be emitted right before the execution phase, after the validations. Such an event is emitted in the correct place in the *handleOps()* function, while it's misplaced in *handleAggregatedOps()*, where it is placed in the middle of the validations.

Path: ./contracts/core/EntryPoint.sol : handleAggregatedOps()

Recommendation: It is suggested to move the *BeforeExecution()* event after the validations to achieve a consistent tracking in the *handleAggregatedOps()* function.

Found in: 131c057

Status: Fixed (Revised commit: 9701c02)

Remediation: The event generation was moved.

I02. Untracked Revert Reason In _executeUserOp()

The *_executeUserOp()* function on the EntryPoint contract is internally called by *handleOps()* and *handleAggregatedOps()* to execute each user operation in the bundle. The actual execution logic is performed by calling *innerHandleOp()* inside a try/catch to parse the revert reason if any. The current code only checks if the revert reason is *INNER_OUT_OF_GAS*, ignoring any other reason.

Path: ./contracts/core/EntryPoint.sol : _executeUserOp()

Recommendation: In the case that the revert reason does not match *INNER_OUT_OF_GAS*, it is suggested to emit an event containing the *opInfo* data

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

and the revert reason. This way the unexpected issues can be tracked and debugged off-chain.

Found in: 131c057

Status: Acknowledged

I03. Possible Lack of Tokens to Cover Gas Expenses

The *BaseAccount.validateUserOp()* is ignoring the *missingAccountFunds* parameter. When not using a *PayMaster* to pay for a transaction Gas, an account must have a sufficient deposited balance on the EntryPoint; otherwise, the transaction will revert. Funds on the account contract won't be automatically taken by the *EntryPoint* to pay for the user operation, leading to its failure.

Path: ./contracts/core/EntryPoint.sol : validateUserOp()

Recommendation: Implement the logic to cover the *missingAccountFunds* by sending VTHO tokens.

Found in: 131c057

Status: Fixed (Revised commit: bc2a9ec)

Remediation: A *_payPrefund()* virtual function was added to host the missing functionality.

I04. Redundant mySimulateValidation() Function

The *simulateValidation()* method simulates a call to *account.validateUserOp()* and *paymaster.validatePaymasterUserOp()*, and provides the validation results as errors - the successful result is *ValidationResult* error, other errors are failures.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

In parallel to *simulateValidation()*, an additional *mySimulateValidation()* function has been added, which contains a subset of the logic and its usage is unclear.

Path: ./contracts/core/EntryPoint.sol : mySimulateValidation()

Recommendation: Add to the documentation the expected behavior and usage of *mySimulateValidation()* or remove the draft function.

Found in: 131c057

Status: Fixed (Revised commit: 0af0fc4)

Remediation: The draft function *mySimulateValidation()* was removed.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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

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

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Appendix 2. Scope

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

Scope details

Repository	https://github.com/vechainfoundation/account-abstraction
Commit	bc2a9ec
Requirements	Link
Technical Requirements	Link

Contracts in Scope

./contracts/core/BaseAccount.sol
./contracts/core/BasePaymaster.sol
./contracts/core/EntryPoint.sol
./contracts/core/Helpers.sol
./contracts/core/NonceManager.sol
./contracts/core/SenderCreator.sol
./contracts/core/StakeManager.sol
./contracts/interfaces/IAccount.sol
./contracts/interfaces/IAggregator.sol
./contracts/interfaces/IEntryPoint.sol
./contracts/interfaces/INonceManager.sol
./contracts/interfaces/IPaymaster.sol
./contracts/interfaces/IStakeManager.sol

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

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

./contracts/interfaces/UserOperation.sol
./contracts/utils/Exec.sol

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.