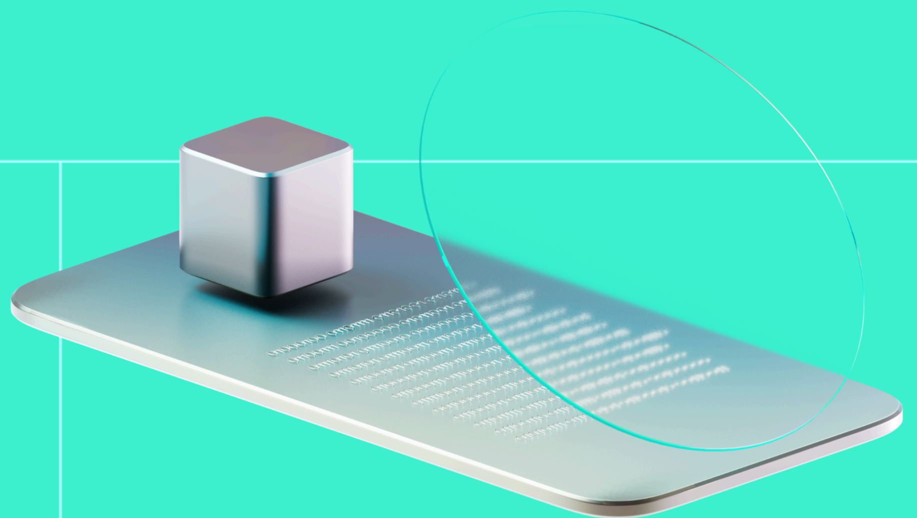
Smart Contract Code Review And Security Analysis Report



Customer: Ociswap

Date: 16 Jan, 2024

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

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

Ociswap AVLTree is a scalable implementation of a binary tree structure in Scrypto.

Platform: Radix DLT

Language: Rust, Scrypto

Tags: AVL Tree

Timeline: 17.10.2023 - 17.01.2023

Methodology: Link

Last review scope

Repositories	https://github.com/ociswap/scrypto-avltree
Commit	254a7ab

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.

0	0	1	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	0	0	0	0
Low	1	0	0	1

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 Ociswap
Approved By	Grzegorz Trawiński | SC Audits Expert at Hacken OÜ
Audited By	Jakub Heba | SC Auditor at Hacken OÜ Vladyslav Khomenko | SC Auditor at Hacken OÜ
Website	https://ociswap.com
Changelog	20.11.2023 – Preliminary Report 17.01.2024 - Remediation Check

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

Introduction.......6
System Overview.......6
Executive Summary.......6
Risks.......7
Findings.......8
- Critical.......8
- High.......8
- Medium.......8
- Low.......9
  - L01. Floating language version.......9
- Informational.......10
  - I01. Unnecessary pattern matching is redundant.......10
  - I02. Vulnerable dependencies.......11
  - I03. Unformatted Code.......12
  - I04. AlvTree optimization to fix recomputation.......12

Disclaimers.......14
Appendix 1. Severity Definitions.......15
- Risk Levels.......16
- Impact Levels.......16
- Likelihood Levels.......17
- Informational.......17

Appendix 2. Scope.......18

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

Introduction

Hacken OÜ (Consultant) was contracted by Ociswap (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

AvlTree - is a radix-specific library implementation of a balanced binary tree structure in Scrypto. It is able to overcome limitations of built-in Rust key-value maps since it is able to lazy load specific elements instead of loading the whole collection.

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 and technical requirements are provided in full.
- Code is properly outlined with comments.

Code quality

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

Test coverage

Code coverage of the project is **100%** (functional coverage).

- The code is thoroughly tested.

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.

Security score

As a result of the audit, the code contains **1** low severity issue. 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**. The system users should acknowledge all the risks summed up in the risks section of the report.

Risks

- Current implementation of the AVLTree has room for gas efficiency optimisations.

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.

■■■ High

No high severity issues were found.

■■ Medium

No medium 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.

Low

L01. Floating language version

Impact	Low
Likelihood	Low

It is preferable for a production project, especially a smart contract, to have the programming language version pinned explicitly. This results in a stable build output, and guards against unexpected toolchain differences or bugs present in older versions, which could be used to build the project.

The language version could be pinned in automation/CI scripts, as well as proclaimed in README or other kinds of developer documentation. However, in the Rust ecosystem, it can be achieved more ergonomically via a rust-toolchain.toml descriptor (see https://rust-lang.github.io/rustup/overrides.html#the-toolchain-file)

Path: *

Recommendation: It is suggested to set a concrete Rust version.

Found in: d1f4553

Status: Accepted

Remediation: The compiler version is already pinned through the Scrypto toolchain using Deterministic Builder.

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

Informational

I01. Unnecessary pattern matching is redundant

At the beginning of its call, the *insert* function verifies whether the key already exists in the store, and if so, old value is returned, new value is inserted, and execution is finished. In the next step, the *insert_node_in_empty_spot* function is called, which has pattern matching in its content verifying whether there is any child in the key direction.

```
fn insert_node_in_empty_spot(&mut self, key: &K, value: V) -> Option<(K,
Direction)> {
       let mut current = self.root.clone();
       let mut parent = None;
       while let Some(parent_key) = current.as_ref() {
           let current_node = self.get_node(parent_key).expect("Root should
exist");
           match current_node.get_child_in_key_direction(key) {
               Some(child) => {
                   parent = current;
                   current = child.cloned();
               }
               None => {
                   panic!("Key already exists this should be caught in the
beginning of insert");
               }
           }
       }
```

If *self.key* is greater, *left_child* is returned. If it is less, *right_child* is returned. However, none is returned only for the case when *self.key* already exists in the tree. Since this case was already checked at the beginning of the insert function execution, it is impossible that at that state these keys will be equal.

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

```
fn get_child_in_key_direction(&self, other_key: &K) -> Option<Option<&K>> {
   match self.key.cmp(other_key) {
       Greater => Some(self.left_child.as_ref()),
       Equal => None,
       Less => Some(self.right_child.as_ref()),
   }
}
```

Ultimately, whole pattern matching is unnecessary because the *Some()* branch will be reached every time.

Path: ./src/avl_tree.rs : insert_node_in_empty_spot()

Recommendation: It is suggested to remove whole pattern matching to reduce the cost of transaction execution and the readability of the code.

Found in: d1f4553

Status: Fixed (Revised commit: 254a7ab)

Remediation: Unnecessary code is removed.

I02. Vulnerable dependencies

Few contracts and libraries use packages with publicly known vulnerabilities, which is considered a deviation from leading security practices. Vulnerable packages may have uncertain impact on implemented functionalities.

```
Crate:     ed25519-dalek
Version:   1.0.1
Title:     Double Public Key Signing Function Oracle Attack on `ed25519-dalek`
Date:      2022-06-11
ID:        RUSTSEC-2022-0093
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0093
```

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

```
Solution:  Upgrade to >=2
Dependency tree:
ed25519-dalek 1.0.1
```

Path: ./Cargo.toml

Recommendation: It is recommended to verify that none of the vulnerable functions are used in the code, or update the package to a higher, secure version.

Found in: d1f4553

Status: Fixed (Revised commit: 254a7ab)

Remediation: The vulnerable dependency is used by the SDK and is only used in tests. The contract does not have vulnerable dependencies.

I03. Unformatted Code

The tool *cargo fmt --check* reports that code is not formatted.

Path: *

Recommendation: It is suggested to format the code using *rustfmt* or an equivalent.

Found in: d1f4553

Status: Fixed (Revised commit: 254a7ab)

Remediation: The code is now formatted by *rustfmt* - default formatter.

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

I04. AlvTree optimization to fix recomputation

During the operation of *AvlTree*, a lot of times, the same node has to be accessed multiple times. At the moment, it is also retrieved multiple times. This is slow, but fortunately, the nodes are cached into *HashMap*, which makes it less computational intensive. However, this optimization does not fully solve the problem of retrieving the same node over and over again during one operation on the tree such as insertion. This is probably done in order to satisfy the strict Rust compiler and its borrow checker. The functions *get_node* and *get_mut_node* are used in this constantly.

Repeated computation decreases efficiency and increases cost of operation/usage.

```
pub(crate) fn get_node(&mut self, key: &K) -> Option<&Node<K, ()>> {
    self.cache_if_missing(key);
    // Carefully this is not synced with the store!
    self.store_cache.get(&key)
}

fn get_mut_node(&mut self, key: &K) -> Option<&mut Node<K, ()>> {
    self.cache_if_missing(key);
    self.store_cache.get_mut(key)
}
```

Path: ./src/avl_tree.rs

Recommendation: Since this is a library data structure, we recommend working on optimizing it. For example, it is possible to utilize the same *HashMap* optimization except only fetch nodes once by operating on raw pointers to them. Keep in mind that insertions into *HashMap* can cause underlying data to move which would invalidate the pointers.

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.

Found in: d1f4553

Status: Accepted

Remediation: It was decided that the suggested optimisation would require many code changes, but result in a minor performance improvement.

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.

https://hacken.io/

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

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.

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

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.

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

Appendix 2. Scope

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

Scope details - Preliminary Audit

Repositories	https://github.com/ociswap/scrypto-avltree
Commits	d1f4553e70339dfd98a12658cf4c4449c9583f1c
Whitepaper	-
Requirements	Link
Technical Requirements	Link

Contracts in Scope

./src/avl_tree.rs

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

Scope details - Remediation

Repositories	https://github.com/ociswap/scrypto-avltree
Commits	254a7ab209bd928462509f7ca1dcf7b05add4094
Whitepaper	-
Requirements	Link
Technical Requirements	Link

Contracts in Scope

./src/avl_tree.rs

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.