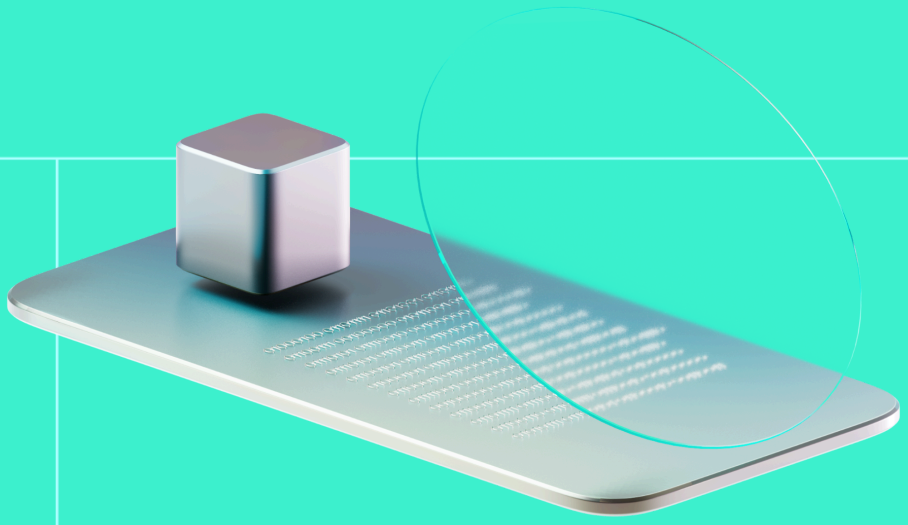
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Ociswap

Date: 9 May, 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 is a top-tier solution, combining greatest DEX methodologies into a novel, high-performance system, catalyzing decentralized finance in the Radix ecosystem.

Platform: Radix DLT

Language: Rust, Scrypto

Tags: Oracle

Timeline: 01.11.2023 - 09.05.2024

Methodology: Link

Last review scope

Repositories	Initial - https://github.com/ociswap/ociswap-blueprints Final - https://github.com/ociswap/oracle
Commit	f828830

View full scope

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: 10.0/10

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

1	2	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	0	0	0	1

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 03.05.2024 - Remediation Check

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

Ociswap Oracle - is a time weighted on-chain price feed. It is designed to store prices of Ociswap DEX token pairs.

Flex Pools and Precision Pools of token pairs can be equipped with the hook to the oracle to be able to record recent prices at per-minute rate for a reliable price source. Users can request a precise price at specific time (historic data is limited) or an average price over a time period for more temper-proof data.

Similar to Uniswap's oracle on Ethereum, Ociswap Oracle keeps track of the cumulative sum of the spot prices, weighted by time between observations. Ociswap introduced several improvements to older iteration of oracle design:

- The Oracle keeps track of multiple recent observations which allows users to get time-weighted prices instantaneously.
- Using arithmetic meant a need for keeping two records: A/B (mean price of token A in terms of token B and B/A. Instead, Ociswap uses geometric means which makes it possible to store a single record, A/B such that its reciprocal is an accurate B/A.

Privileged roles

- hook_admin - role used for executing hooks after instantiation and before every swap, editable by OWNER.
- OWNER - badge who instantiated the contract. It has different abilities depending on the component.

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**.

- Documentation is very accurate and describes the processes taking place in Oracle.

Code quality

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

- Debug macros are present.

Test coverage

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

- The code is thoroughly tested.
- Deployment and basic user interactions are covered with tests.

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

Risks

- The system is designed to call external code, which can have an impact on the fundamental pieces of the system: fees and tokens, involved in swap. If a vulnerable or compromised component were to have access, it could pose a risk of mishandling mentioned parts of the system.

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.

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: a805c12

Status: Accepted

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

Informational

I01. 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
Solution:   Upgrade to >=2
Dependency tree:
ed25519-dalek 1.0.1
[..]
```

Path: Multiple Cargo.toml files

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: a805c12

Status: Accepted

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

I02. Multiple TODO comments

It was identified that in many places in the code there are comments indicating that the contract logic is not completed at this point - marked as "TODO". While they do not pose an immediate threat, their existence in production code is considered bad security practice because it can help a would-be attacker map interesting attack vectors onto the protocol.

Sample of the “TODO” comment left:

```
impl OracleHook {
   pub fn instantiate() -> (Global<OracleHook>, Bucket) {
       let hook_badge = ResourceBuilder::new_fungible(OwnerRole::None)
           .divisibility(DIVISIBILITY_NONE)
           .mint_initial_supply(1);

       let hook_global = (Self {
           calls: vec![HookCall::AfterInstantiate, HookCall::BeforeSwap,
HookCall::AfterSwap],

           pool_address: None,
           x_address: None,
           y_address: None,

           observations: KeyValueStore::new(),

           last_timestamp: 0,
           last_observation_index: 0,
           observations_stored: 0,
           observations_limit: OBSERVATIONS_LIMIT_INITIAL,
           observations_limit_max: u16::MAX,


           acc_x_volume: dec!(0), //todo is this what we wanna keep?
           acc_y_volume: dec!(0),


           liquidity: pdec!(0),
           price_sqrt: pdec!(0),
       })
```

Occurrences throughout the codebases:

- ./oracle/oracle_hook.rs#183, 416

Recommendation: It is recommended to eliminate all "TODO" comments.

Found in: a805c12

Status: Fixed (Revised commit: f828830)

Remediation: TODO comments were removed from the codebase.

I03. Usage of debugging macros throughout the codebases

When creating code, functions that support developers in returning the values of state variables and comparing them with the expected results are very helpful. Examples of such functions/macros are *debug!()* and *info!()*, which display strings and variable values in the console.

While this is very helpful during code development, in a production version of the solution, these types of calls should be removed. They affect the readability of the code and may be negatively perceived by developers creating solutions based on the protocol code.

Sample of *debug!()* macro usage:

```
pub fn after_instantiate(
   &mut self,
   pool_address: ComponentAddress,
   price_sqrt: PreciseDecimal,
   x_address: ResourceAddress,
   y_address: ResourceAddress
```

```
) -> (ComponentAddress, PreciseDecimal, ResourceAddress, ResourceAddress) {
    debug!(
        "[ORACLE HOOK] After instantiate, observation count: {}",
        self.last_observation_index
    );


    debug!(
        "[ORACLE HOOK] After instantiate, clock_time: {} unix-minutes, i.e. {}
unix-seconds",
        Clock::current_time_rounded_to_minutes().seconds_since_unix_epoch,
        Clock::time_in_minutes()
    );


    self.pool_address = Some(pool_address);
    self.x_address = Some(x_address);
    self.y_address = Some(y_address);


    (pool_address, price_sqrt, x_address, y_address)
}
```

Path: *

Recommendation: It is suggested to eliminate the *debug!()* and *info!()* macros in contract and library codes in production code.

Found in: a805c12

Status: Fixed (Revised commit: f828830)

Remediation: All mentioned macros were removed from the common/oracle.rs file. These existing in the *oracle_hook.rs* file are used mostly for testing purposes, and will be eliminated in the future release.

I04. Redundant check in Oracle

Data records are written to oracle at most once per minute. Before creating a data record in Oracle, the variable *self.observations_stored* is checked to see if there are any records yet. Then there is a check for whether the last data record was added in the current minute. This second check covers every case since *self.last_timestamp* will be equal to 0 if there are no records yet.

```
if self.observations_stored != 0 && self.last_timestamp ==
Clock::time_in_minutes() {
    return (swap_state, input_bucket);
}
```

Path: ./oracle/src/oracle_hook.rs : before_swap()

Recommendation: It is recommended to remove the first check.

Found in: a805c12

Status: Fixed **(**Revised commit: f828830)

Remediation: Check mentioned above was removed from the Oracle code.

I05. Duplicate calculation

There is a calculation that is performed two times. One in a check and second when it is actually being used.

```
assert!(
    t_left / 60 < t_right / 60,
    "Provided intervals must be of the type [a, b], where a < b. Interval [{},
{}] does not obey this condition.",
    t_left,
    t_right
);

let t_left = t_left / 60;
let t_right = t_right / 60;
```

Path: ./oracle/src/oracle_hook.rs : observation_intervals()

Recommendation: Consider using assertion on variables after calculation.

Found in: a805c12

Status: Fixed **(**Revised commit: f828830)

Remediation: The operations have been swapped so that division only happens once.

I06. 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.

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

Found in: a805c12

Status: Fixed (Revised commit: f828830)

Remediation: Formatter is not returning any issues now.

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 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

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.

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

Repositories	https://github.com/ociswap/ociswap-blueprints
Commits	Initial - a805c12 Final - f828830
Requirements	Link
Technical Requirements	Link

Contracts in Scope

./oracle/oracle_hook.rs