HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SatoshiPay
Date: 28 July, 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 SatoshiPay
Approved By	Oleksii Zaiats | Head of Solidity SC Auditor department at Hacken OU
Type	ERC20 token;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://pendulumchain.org/
Changelog	07.07.2023 - Initial Review 28.07.2023 - 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 SatoshiPay (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

- ERC20Wrapper - this contract implements openzeppelin's *IERC20* and *IERC20Metadata* interfaces. It is able to communicate with the chain extensions directly, without the need to have a second wrapper contract implemented in ink! This purpose of this contract is to allow our wrapped tokens to satisfy *IERC20* so that other Solidity contracts can use them the same way as any other *IERC20* token.
- PriceOracleWrapper - this contract provides the function that is expected by Nabla's IPriceOracleGetter interface. It calls the 1200 chain extension which fetches a price feed from the chain. The inputs blockchain and symbol are the keys to query a particular price feed.
- IPreceOracleGetter - an interface which contains a function that returns token price in USD.

Privileged roles

- Contracts have no privileges roles.

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

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 **9** out of **10**.

- Functional requirements are not provided:
  - Project overview is detailed.
  - Use cases are described and detailed.
  - All interactions are described.
- Technical description is inadequate:
  - Run instructions are provided.
  - Technical specification is provided.
  - NatSpec is partially missing.

Code quality

The total Code Quality score is **8** out of **10**.

- Best practice violations.
- Insufficient Gas modeling.
- Solidity Style Guide violations.

Test coverage

Code coverage of the project is **0.0%** (branch coverage).

- Tests are not provided.
- Project has less than 250 lines of code, the lack of tests will not affect score, although it’s recommended to include tests.

Security score

As a result of the audit, the code contains **1** 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.

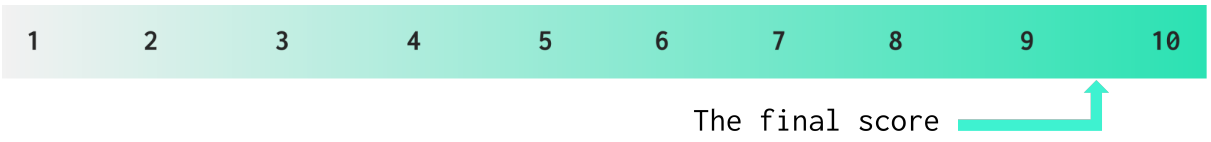

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
6 July 2023	4	0	0	0
28 July 2023	1	0	0	0

Risks

No potential security risks were found during the audit research.

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

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Not Relevant	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

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

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Not Relevant	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Not Relevant	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	
Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed	
Style Guide Violation	Style guides and best practices should be followed.	Failed	L02, I01, I02, I05, I06
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

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

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 Pragma

Impact	Medium
Likelihood	Low

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.

Path:
./price-oracle-wrapper/interfaces/IPriceOracleGetter.sol

Recommendation: it is recommended to lock the pragma version in all contracts as stated by SWC-103.

Found in: db10871f6ec81d74e0afe9e1e49b8f2e143aadc5

Status: Fixed (Revised commit: b54848f713f5e8f6b64b585dee2d9be8715f47ed).

L02. Missing Zero Address Validation

Impact	Low
Likelihood	Low

Additional checks against the *0x0* address should be included in the reported functions to avoid unexpected results.

Paths:

./price-oracle-wrapper/price-oracle-wrapper.sol : getOracleKeyAsset(), getOracleKeyBlockchain(), getOracleKeySymbol(), getAssetPrice();

./erc20-wrapper/erc20-wrapper.sol : balanceOf(), transfer(), allowance(), approve(), transferFrom();

Recommendation: it is recommended to add zero address checks.

Found in: db10871f6ec81d74e0afe9e1e49b8f2e143aadc5

Status: Reported (Revised commit: b54848f713f5e8f6b64b585dee2d9be8715f47ed). (Several functions still need to be improved)

L03. Variables That Can Be Set Immutable

Impact	Low
Likelihood	Low

Use the *immutable* keyword on the state variables to limit changes to its state and save Gas.

Paths:

./erc20-wrapper/erc20-wrapper.sol : _name, _symbol, _decimals, _variant, _code, _issuer;

Recommendation: consider using the keyword immutable for said variable.

Found in: db10871f6ec81d74e0afe9e1e49b8f2e143aadc5

Status: Fixed (Revised commit: b54848f713f5e8f6b64b585dee2d9be8715f47ed).

L04. Missing Events

Impact	Low
Likelihood	Medium

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

Paths:

./erc20-wrapper/erc20-wrapper.sol : transfer(), transferFrom(), approve();

Recommendation: consider emitting events in said functions.

Found in: db10871f6ec81d74e0afe9e1e49b8f2e143aadc5

Status: Fixed (Revised commit: b54848f713f5e8f6b64b585dee2d9be8715f47ed).

Informational

I01. Solidity Style Guide Violation: Naming mismatch

Files names should be the same as smart contracts naming.

Paths:
./price-oracle-wrapper/price-oracle-wrapper.sol;

./erc20-wrapper/erc20-wrapper.sol;

Recommendation: change the name of the files to match the names of the smart contracts.

Found in: db10871f6ec81d74e0afe9e1e49b8f2e143aadc5

Status: Reported

I02. Functions That Should Be External

Public functions that are not called from inside the contract should be declared external to save Gas.

Paths:

./erc20-wrapper/erc20-wrapper.sol : name(), symbol(), decimals(), totalSupply(), balanceOf(), transfer(), allowance(), approve(), transferFrom();

./price-oracle-wrapper/price-oracle-wrapper.sol : getAnyAssetSupply(), getAnyAssetLastUpdateTimestamp();

Recommendation: consider changing the function visibility to external.

Found in: db10871f6ec81d74e0afe9e1e49b8f2e143aadc5

Status: Reported

I03. Solidity Style Guide Violation: Order Of Layout

Inside each contract, library or interface, use the following order:

1. Type declarations
2. State variables
3. Events
4. Errors
5. Modifiers
6. Functions
    a. constructor
    b. initializer (if exists)
    c. receive function (if exists)
    d. fallback function (if exists)
    e. external
    f. public

g. internal
h. private

Path:

./price-oracle-wrapper/price-oracle-wrapper.sol

Recommendation: change order of layout to fit Official Style Guide.

Found in: db10871f6ec81d74e0afe9e1e49b8f2e143aadc5

Status: Fixed (Revised commit: b54848f713f5e8f6b64b585dee2d9be8715f47ed).

I04. Inefficient Gas Modeling

Inside loop the value of *oracleKeys.length* taken every loop iteration. Declaring a variable equal to the length of the list before the loop will reduce the gas consumption when deploying a smart contract.

Path:

./price-oracle-wrapper/price-oracle-wrapper.sol : constructor();

Recommendation: declare variable above loop to decrease .

Found in: db10871f6ec81d74e0afe9e1e49b8f2e143aadc5

Status: Reported

I05. Redundant Function Virtualization

The following functions are marked as virtual in the code, but never being overridden. Virtual functions are much more Gas expensive compared to default functions.

Paths:
./erc20-wrapper/erc20-wrapper.sol : name(), symbol(), decimals();

Recommendation: make these functions non-virtual.

Found in: db10871f6ec81d74e0afe9e1e49b8f2e143aadc5

Status: Reported

I06. Solidity Style Guide: mixedCase in State Variables Names

Local and State Variable names should be mixedCase: capitalize all the letters of the initialisms, except keep the first one lower case if it is the beginning of the name.

Path:

./price-oracle-wrapper/price-oracle-wrapper.sol : _oracleByAsset;

Recommendation: follow the official Solidity guidelines.

Found in: db10871f6ec81d74e0afe9e1e49b8f2e143aadc5

Status: Reported

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

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.

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

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/pendulum-chain/pendulum-ink-wrapper
Commit	db10871f6ec81d74e0afe9e1e49b8f2e143aadc5
Whitepaper	https://pendulum.gitbook.io/pendulum-docs/learn/litepaper
Requirements	https://github.com/pendulum-chain/pendulum-ink-wrapper/blob/master/README.md
Technical Requirements	https://docs.google.com/document/d/1kS6S7uf0VvkPbM2xT4z4ipWd7lwdGxT2GL9pu2EneNc/edit?usp=sharing
Contracts	File: ./erc20-wrapper/price-oracle-wrapper.sol SHA3: f99d95d4c813f813592f11079169d7277f078ea0aa4a9b2f68b5a1ba88246613 File: ./price-oracle-wrapper/erc20-wrapper.sol SHA3: bcd0ce6b7a6712ed13cd34b4a0b9b8306a1cba9ff2007864e5f4a6de0863e2af File: ./price-oracle-wrapper/interfaces/IPriceOracleGetter.sol SHA3: 7c79418c642529bddfb8c10f04a56d8bc6237620a21fa3c0bebcd7250c2d7bc9

Second review scope

Repository	https://github.com/pendulum-chain/pendulum-ink-wrapper
Commit	b54848f713f5e8f6b64b585dee2d9be8715f47ed
Whitepaper	https://pendulum.gitbook.io/pendulum-docs/learn/litepaper
Requirements	https://github.com/pendulum-chain/pendulum-ink-wrapper/blob/master/docs/OVERVIEW.md
Technical Requirements	https://github.com/pendulum-chain/pendulum-ink-wrapper/blob/master/docs/OVERVIEW.md
Contracts	File: ./erc20-wrapper/erc20-wrapper.sol SHA3: b2bee305d9a10ba38d023aa5a69a92d2e9743ab344c3f17a37adbf79cf789d0f File: ./price-oracle-wrapper/price-oracle-wrapper.sol SHA3: 0a95a1b7b5f28189d8f6c97dbd00ae6dfefecd671d9a72c36b0d7a8726e4e2f7 File: ./price-oracle-wrapper/interfaces/IPriceOracleGetter.sol SHA3: 43d9c97e09aabb2cffff2cb4e9bb380f2e82fa5742b8dd598988b3db4fac3930