HACKEN

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer:** ShopX
**Date:** June 27th, 2022

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 ShopX. |
| **Approved By** | Evgeniy Bezuglyi \| SC Department Head at Hacken OU |
| **Type of Contracts** | ERC-721 token; ERC-721 token factory |
| **Platform** | EVM |
| **Language** | Solidity |
| **Methods** | Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review |
| **Website** | https://shopx.co/ |
| **Timeline** | 04.04.2202 - 24.06.2022 |
| **Changelog** | 11.04.2022 - Initial Review<br>21.04.2022 - Second Review<br>16.06.2022 - Third Review<br>27.06.2022 - Fourth 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 ShopX (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.

The second review was conducted on April 19th, 2022.
The third review was conducted on June 16th, 2022.
The fourth review was conducted on June 23th, 2022.

## Scope

The scope of the project is smart contracts in the repository:

**Repository:**
https://github.com/shopxlabs/shopx-smart-contracts/tree/feature/ID-617
**Commit:**
bfe3183de0e404357055719f8fcfb0988ca82712
**Technical Documentation:** Yes
(https://github.com/shopxlabs/shopx-smart-contracts/blob/feature/ID-617/documentation/Deploy-ReserveX.md)
**JS tests:** Yes
**Contracts:**
./contracts/ShopXReserveNFT.sol
./contracts/ShopXReserveFactory.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

| Category | Check Item |
|---|---|
| Code review | ▪ Reentrancy<br>▪ Ownership Takeover<br>▪ Timestamp Dependence<br>▪ Gas Limit and Loops<br>▪ Transaction-Ordering Dependence<br>▪ Style guide violation<br>▪ EIP standards violation<br>▪ Unchecked external call<br>▪ Unchecked math<br>▪ Unsafe type inference<br>▪ Implicit visibility level<br>▪ Deployment Consistency<br>▪ Repository Consistency |
| Functional review | ▪ Business Logics Review<br>▪ Functionality Checks<br>▪ Access Control & Authorization<br>▪ Escrow manipulation<br>▪ Token Supply manipulation<br>▪ Assets integrity<br>▪ User Balances manipulation<br>▪ Data Consistency<br>▪ Kill-Switch Mechanism |

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 methodology.

## Documentation quality

The Customer provided a basic description of the project but did not provide functional requirements. The total Documentation Quality score is **5** out of **10**.

## Code quality

The total CodeQuality score is **10** out of **10**. Most of the code is commented and covered with unit tests.

## Architecture quality

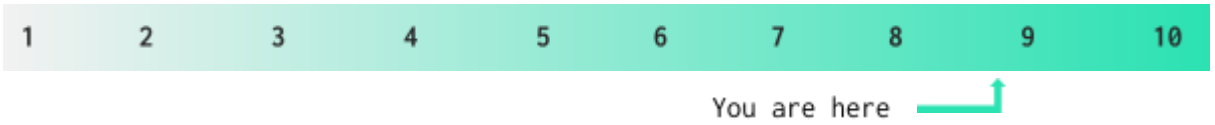
The architecture quality score is **10** out of **10**. The project has clear and clean architecture.

## Security score

As a result of the audit, security engineers found **1** medium severity issue. The security score is **9** out of **10**. All found issues are displayed in the “Issues overview” section.

## Summary

According to the assessment, the Customer's smart contract has the following score: **8.8**



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

## Severity Definitions

| Risk Level | Description |
|---|---|
| Critical | Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations. |
| High | High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions |
| Medium | Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations. |
| Low | Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution |

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

# Findings

## Critical

No critical severity issues were found.

## High

1. **Missing allowance on Burn operation.**

   The function does not validate whether approval for burning was set by a user.

   Any token may be burned by the account with the `SALE_ADMIN` role without user approval.

   **Contracts**: ShopXReserveNFT.sol

   **Function**: burn

   **Recommendation**: Burn tokens only when a user provides approval for this.

   **Status**: Fixed (0ec7932779dde2b4e8cd1d165412563ab71adf89)

## Medium

1. **Exceed the tokens limit per wallet.**

   The contract has the function `mintTo` that allows the admin to mint a new ERC-721 token for free and send it to any address. The function does not check if the number of tokens assigned to the recipient exceeds the `mintLimitPerWallet` value.

   The admin may mint an unlimited amount of tokens to any address, whereas whitelisted users may not mint more tokens than the `mintLimitPerWallet` amount.

   **Contracts**: ShopXReserveNFT.sol

   **Function**: mintTo

   **Recommendation**: Add a `require` statement to check if `mintLimitPerWallet` limit is exceeded.

   **Status**: Mitigated (This is how the client expected the `mintTo()` function to work, and this behavior was designed by the client.)

2. **The total supply amount may be wrong.**

   The ERC-721 contract has the `totalSupply` getter function which returns the `_nextTokenId` counter value minus `1`, but the `_nextTokenId` counter value is not updated after burning tokens with a `burn` function.

   The `totalSupply` value may exceed the real amount of tokens if some of the tokens are burned.

   **Contracts**: ShopXReserveNFT.sol

**Function**: burn, _burn

**Recommendation**: Update the counter value after burning tokens.

**Status**: Fixed (8e89404d2f34b48af0767885849a9901c2d6f59b, f6c9d7883c08e05369307dc962c9f4359db72917)

3. **The initialization function is callable more than once.**

   The implementation of the upgradable contract has an initialization function, but the function has no modifier, which may restrict reinitialization.

   If the initialization function may be called more than once, it is possible to reset the state variables, which may cause unexpected issues.

   **Contracts**: ShopXReserveNFT.sol

   **Function**: initialize

   **Recommendation**: Add `initializer` modifier to the `initialize` function.

   **Status**: Fixed (9a428829ba9f204084e484b98839bbf23a7b8dc5)

4. **Missing input value validation.**

   The factory function *setShopxFee* allows setting any *shpoxFee* value.

   It will be impossible to mint the new ERC721 token if the *shpoxFee* is greater than *10000,* due to the inability to transfer more funds than on the contract.

   **File:** ./contracts/ShopXReserveFactory.sol

   **Functions**: setShopxFee

   **Recommendation**: Add validation for the input value in the *setShopxFee* function.

   **Status**: New

## Low

1. **Redundant modifier.**

   The contract has the `onlyAllow` modifier, which is never used.

   **Contracts**: ShopXReserveNFT.sol

   **Recommendation**: Remove unused modifier.

   **Status**: Fixed (a102ef4ebddcf72a9f81f36344c12b14233bf9e3)

2. **Redundant events.**

   The contract has declared a couple of events that are never used.

   Two events are never used: `ETHReceive`, `PriceChange`.

**Contracts**: ShopXReserveNFT.sol

**Recommendation**: Remove unused events or emmit them when needed.

**Status**: Fixed (56a47924358aec8d09833ce72cae625bcb3320f1)

3. **Missing zero address validation.**

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

   This can lead to unwanted external calls to 0x0.

   **File**: ./contracts/ShopXReserveFactory.sol

   **Functions**: constructor, setShopxAddress

   **File**: ./contracts/ShopXReserveNFT.sol

   **Functions**: updateShopxAddress, updateBeneficiaryAddress, updateRoyaltyAddress

   **Recommendation**: Implement zero address validations.

   **Status**: Fixed (a39b36b29079d7d19df119d2b83e18ab371c2a90, cbadb61f439367e601f8a666e6612a36a393a642, c839373b65ac9c948a2b5201ff0d22b11e863b46)

4. **Floating pragma.**

   The contracts use floating pragma ^0.8.13.

   Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma helps 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.

   **Recommendation**: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

   **Status**: Fixed (fb2e2bfdd28371da6de5b58d04393069e268c331)

5. **Missing explicit visibility levels.**

   Labeling the visibility explicitly makes it easier to catch incorrect assumptions about the accessibility of the variable.

   **File**: ./contracts/ShopXReserveNFT.sol

   **Variables**: maxSupply, mintPrice, mintLimitPerWallet, royaltyValue, royaltyRecipient, shopxAddress, shopxFee, beneficiaryAddress

   **Recommendation**: Explicitly define visibility for all state variables.

   **Status**: Fixed (fdbd50059a7d28010717874a18f0a6fde3f81259)

6. **Unnecessary import statements.**

   Unnecessary import statements may make code unclear.

   **File**: ./contracts/ShopXReserveFactory.sol

**Import**: @openzeppelin/contracts/access/Ownable.sol

**File:** ./contracts/ShopXReserveNFT.sol

**Import**:
@openzeppelin/contracts/access/Ownable.sol
@openzeppelin/contracts/utils/Counters.sol

**Recommendation**: Remove unnecessary import statements.

**Status**: Fixed (518fe82ab0b52816258b9f7750d2998797dbedd2, 7b7e24f0c9729b2a816ce2a05d64fba735c9f8c6)

7. **Some functions that could be declared as external.**

   The functions with the `public` visibility that are never called inside the contract should be declared as `external` to save Gas.

   **File**: ./contracts/ShopXReserveNFT.sol

   **Functions**: mint

   **Recommendation**: Use the `external` attribute for functions never called from the contract.

   **Status**: Fixed (97f367fa05ca6e5850e807c0f39dd9b0bfc83536)

8. **Redundant usings**

   The contract declares the `String` for `uint256` values, but the library functions are never used.

   **File**: ./contracts/ShopXReserveNFT.sol

   **Recommendation**: Remove unused statement.

   **Status**: Fixed (08885cedbc0b7c6c90fecc494e845af687cb81e3)

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 by the best industry practices at the date 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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

## 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 audit cannot guarantee the explicit security of the audited smart contracts.