Berry Block - Swaptor - Audit

Introduction

A time-boxed security review of the **Swaptor** contract was done by **Bugzy Von Buggernaut**.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Overview

The Swaptor contract enables atomic swaps between ERC20 and ERC721 assets. Sellers sign a desired swap, either with a defined or undefined buyer. A buyer can then use the Seller's signature along with the relevant swap data to execute the trade on Swaptor.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack
Likelihood - the chance that a particular vulnerability gets discovered and exploited
Severity - the overall criticality of the risk

Security Assessment Summary

The following number of issues were found, categorised by their severity:

- Critical & High: 1 issue
- Medium: 1 issue
- Low: 1 issue
- Informational: 3 issues

Findings Summary

ID	Title	Severity
[H-01]	Assuming non-reverting ERC20 transfers are successful puts user funds at risk	High

ID	Title	Severity
[M-01]	Use `safeTransferFrom` instead of `transferFrom` for ERC721 transfers	Medium
[L-01]	`OwnableUpgradeable` uses single-step ownership transfer	Low
[I-01]	Upgrade `ECDSAUpgradeable.sol`	Informational
[I-02]	Chainlink's `latestRoundData` might return stale or incorrect results	Informational
[I-03]	A view function that returns the `msg.value` required to cover the `fee` would help to avoid confusion regarding dollar vs wei denomination	Informational

Detailed Findings

[H-01] Assuming non-reverting ERC20 transfers are successful puts user funds at risk

Severity

Impact: High, because users won't receive funds from the counter-party. They can get scammed.

Likelihood: Medium, because a user would need to want to trade a token that returns false on failed transfers, like ZRX

Code Location

105; 110; 167; 219

Description

All ERC20 transfer functions are implemented as so:

```
IERC20Upgradeable(wantedERC20).transferFrom(
        _msgSender(),
        seller,
        wantedERC20Amount
);
```

On the surface it looks okay, but the issue is that ERC20 as a standard has not been implemented consistently across tokens.

One such issue is that tokens like ZRX don't revert on transfer failure but instead return false.

This means that a malicious actor could conduct a swap with a token like ZRX, not pay the counter-party, yet still receive the tokens.

A malicious actor doesn't even need to hold any ZRX, they just need to create many attractive selling offers and wait for another user to get tricked.

Recommended Mitigation Steps

To avoid the above issue, it's recommended to use `safeTransferFrom` from OpenZeppelin's SafeERC20 Lib for ERC20 transfers.

As per the docs: "To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract, which allows you to call the safe operations as `token.safeTransfer(…)`, etc."

Here is the code snippet that prevents the described issue:

```
 bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level
call failed");
        if (returndata.length > 0) {
            // Return data is optional
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did
not succeed");
        }
```

Note that since OpenZeppelin v5, "The upgradeable library no longer includes upgradeable variants for libraries and interfaces.", meaning it's safe to use `SafeERC20` instead of `SafeERC20Upgradable` (The same applies for `ECDSAUpgradeable`).

For a good coverage of ERC20 inconsistencies, see here: https://github.com/d-xo/weird-erc20

Discussion

Bugzy: Fixed

[M-01] Use `safeTransferFrom` instead of `transferFrom` for ERC721 transfers

Severity

Impact: Medium, either the trade won't execute properly (possibly with the seller/buyer leaving with both the bid and the ask), or an NFT buyer permanently loses their purchase.

Likelihood: Medium, most users purchasing NFTs with smart contracts are assumed to be somewhat sophisticated, and an attacker wouldn't stand to gain considerable value from an intentional scam.

Code Location

162; 224; 276; 281

Description

The ERC721 `transferFrom()` method is used instead of `safeTransferFrom()`.

This is not recommended because:

i) Smart contracts are sometimes incapable of receiving ERC721 tokens and `onERC721Received()` is a safeguard against this.

ii) Some NFT's have logic in the `onERC721Received()` function, which is only triggered in the `safeTransferFrom()` function and not by `transferFrom()` (e.g. here)

Recommended Mitigation Steps

Call the `safeTransferFrom()` method instead of `transferFrom()`. This is also recommended by [OpenZeppelin](#)

Discussion

Bugzy: Fixed

[L-01] `OwnableUpgradeable` uses single-step ownership transfer

Severity

Impact: Low, because it requires an error on the admin side

Likelihood: Medium, the contract will no longer be upgradable and the team won't be able to change certain parameters or take profits

Code Location

16

Description

Single-step ownership transfer means that if a wrong address is passed when transferring ownership or admin rights, the role is lost forever. The ownership pattern implementation for the protocol is in `OwnableUpgradeable.sol` where a single-step transfer is implemented. This can be a problem for all methods marked with `onlyOwner` throughout the protocol.

Recommended Mitigation Steps

It is a best practice to use the two-step ownership transfer pattern, meaning ownership transfer gets to a "pending" state and the new owner can claim their new rights, otherwise the old owner still has control of the contract.

Consider using OpenZeppelin's `Ownable2StepUpgradable` contract, but be aware that [`Ownable2StepUpgradeable` requires initialising `OwnableUpgradeable`](#) with `__Ownable_init(address initialOwner)`.

Discussion

Bugzy: Fixed

[I-01] Upgrade ECDSAUpgradeable.sol

Since [OpenZeppelin v5](#), "The upgradeable library no longer includes upgradeable variants for libraries and interfaces.", and therefore it's safe to use `ECDSA.sol` (v5) on upgradable contracts.

Although highly unlikely in Swaptor's case, there are known issues with the v4 ECDSA Library (>= 4.1.0 < 4.7.3) where the single byte argument version of `ECDSA.recover` is vulnerable to replay attacks.

Discussion

Bugzy: Fixed

[I-02] Chainlink’s latestRoundData might return stale or incorrect results

If there is problem with Chainlink starting a new round and finding consensus (e.g. chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the chainlink system) , `latestRoundData` may use outdated or incorrect data .

This doesn't pose significant threat to Swaptor but can be circumvented by adding the following:

```
( roundId, rawPrice, , updateTime, answeredInRound ) =
AggregatorV3Interface(XXXXX).latestRoundData();
require(rawPrice > 0, "Chainlink price <= 0");
require(updateTime != 0, "Incomplete round");
require(answeredInRound >= roundId, "Stale price");
```

Discussion

Bugzy: Fixed

[I-03] A view function that returns the `msg.value` required to cover the `fee` would help to avoid confusion regarding dollar vs wei denomination.

Currently a `fee` parameter exists in the contract and it's easy to assume that whatever this is is the `fee` a user must pay. However, when the price oracle is set, this is not the case.

For example, assuming the fee is $5 (5*(10^8))) and the price oracle is set, when a user sends 500,000,000 wei, the function will revert.

This will cause UX friction as a user will need to work out the fee manually (i.e. query the oracle, and do the math). A `getFee` function would help circumvent this.

Discussion

Bugzy: Acknowledged