WERC721 Security Review

Pashov Audit Group

Conducted by: pashov

August 30th, 2023

Contents

1. About pashov

Krum Pashov, or **pashov**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Check his previous work here or reach out on Twitter @pashovkrum.

2. 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.

3. Introduction

A time-boxed security review of the **WERC721** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About WERC721

WERC721 is an ERC721 wrapper that allows for the following features/optimizations:

- Native call-batching
- Significant `transfer` gas costs reduction
- Meta transactions (authorized transfers)

The protocol is permissionless, non-upgradeable and free - it has no built-in fees. WERC721 is partially-compliant implementation of the ERC721 standard, as it is missing some of its methods (`balanceOf`, `safeTransferFrom`, `approve`, `getApproved`). The protocol is expected to be deployed on Ethereum Mainnet, Optimism, Arbitrum, Arbitrum Nova, Polygon and zkEVM.

Observations

The `WERC721Factory` contract uses the CloneWithImmutableArgs approach for deploying new `WERC721` instances.

The `WERC721` contract will have allowance to move user's NFTs to be wrapped before a `wrap` call.

Signature nonces used in the application are of type `bytes32` as opposed to a sequential `uint256` nonce. It's following EIP3009.

Privileged Roles & Actors

- Authorized WERC721 spender - can transfer a WERC721 token using a signed payload by the token owner
- ERC721 token holder - can call `WERC721::wrap` to wrap his NFT token
- Any user - can call `WERC721Factory::create` to deploy a WERC721 wrapper contract for any ERC721 collection contract

5. Risk Classification

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

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

***review commit hash* - 615d8633d8928e5a13af98ac098cf3631d092d49**

***fixes review commit hash* - 2422c33588f104704e554ca14e4ab4c9a4878755**

Scope

The following smart contracts were in scope of the audit:

- `WERC721`
- `WERC721Factory`

Also the following methods from the project's dependencies were included in the scope of the audit:

- LibClone::clone
- Clone::_getArgAddress
- Multicallable::multicall

7. Executive Summary

Over the course of the security review, pashov engaged with WERC721 to review WERC721. In this period of time a total of **4** issues were uncovered.

Protocol Summary

Protocol Name	WERC721
Date	August 30th, 2023

Findings Count

Severity	Amount
Low	4
Total Findings	**4**

Summary of Findings

ID	Title	Severity	Status
[L-01]	No way to get ETH out of proxy	Low	Resolved
[L-02]	Some contracts might not be able to handle WERC721 tokens	Low	Resolved
[L-03]	Bytecode might not be compatible with all EVM-based chains	Low	Resolved
[L-04]	Timestamp comparison has a logical error	Low	Resolved

8. Findings

8.1. Low Findings

[L-01] No way to get ETH out of proxy

The proxy pattern used implements a `receive() payable` method (can be seen here), but there is no way to get the received ETH out of the proxy, as the implementation contract that it `delegatecall`s to (`WERC721`) does not have a method to withdraw ETH out of it. Since the proxy is not expected to receive ETH this is not a High Impact issue, but it can still be worth it to add a `withdrawETH` method.

[L-02] Some contracts might not be able to handle `WERC721` tokens

The `ERC721` standard added the `safeTransferFrom` functionality because there can be smart contract accounts that do not expect to receive such tokens and can't handle (move) them. The same problem exists for `WERC721`, as a smart contract might not be able to call `transfer` or `unwrap`. A possible solution is the same mechanism as `ERC721`.

[L-03] Bytecode might not be compatible with all EVM-based chains

The protocol is expected to be deployed on multiple EVM-based chains (Optimism, Arbitrum, Polygon etc) but the `pragma` statement shows usage of 0.8.21 version of the Solidity compiler. This version (and every version after 0.8.19) will use the `PUSH0` opcode, which is still not supported on some EVM-based chains, for example Arbitrum. Consider using version 0.8.19 so that the same deterministic bytecode can be deployed to all chains.

[L-04] Timestamp comparison has a logical error

In `WERC721::transferFromWithAuthorization` we see the following code:

```
if (block.timestamp < validAfter) revert InvalidTransferAuthorization();
if (block.timestamp > validBefore)
    revert InvalidTransferAuthorization();
```

There is a logical error there, as even though the variable is named `validAfter`, the value of the variable would still work as a valid timestamp for transfer. Example is if the code says "validAfter = 12:00:00 PM" and then you pas 12:00:00 it should fail, as it should be only valid "after", but with the current code it will work. Change `<` to `<=` and `>` to `>=`.