


ERC-721 Permit2 Review - Cont... Try  HackMD (https://hackmd.io?utm_source=view-page&utm_medium=logo-nav).



ERC-721 Permit2 Review

May 11, 2024

Prepared for Contango Protocol

Conducted by Richie Humphrey (devtooligan)

About the Contango ERC-721 Permit2 Review

Contango builds perps by automating a looping strategy, also known as recursive borrowing and lending. This is achieved using spot and money markets.

This contract allows transfers of ERC-721 non-fungible tokens if the signature of the owner is provided. This contract is heavily inspired by Uniswap's Permit2 with only slight modifications for transferring ERC-721 tokens instead of ERC-20 tokens.

About Offbeat Security

Offbeat Security Labs, LLC is a boutique blockchain security company specializing in complex and novel DeFi projects. Our mission is to elevate the blockchain security landscape through innovative, collaborative, and unconventional solutions.

Summary & Scope

The [erc721-permit2](https://github.com/contango-xyz/erc721-permit2/blob/88a4649c3016ff1e664571edc101058b11c8eaa4/src/ERC721Permit2.sol) (<https://github.com/contango-xyz/erc721-permit2/blob/88a4649c3016ff1e664571edc101058b11c8eaa4/src/ERC721Permit2.sol>) repo was reviewed at commit 88a4649c3016ff1e664571edc101058b11c8eaa4.

The following one contract was in scope:

- src/ERC721Permit2.sol

Contango has also requested that we document any assumptions implicit in the code which we encounter during this review.

Summary of Findings

Identifier	Title	Severity	Fixed
<u>M-01</u>	SafeTransfer not used	Medium	Fixed in <u>PR1</u> (https://github.com/contango-xyz/erc721-permit2/pull/2).
<u>I-01</u>	Invalid signatures accepted	Informational	Fixed in <u>PR2</u> (https://github.com/contango-xyz/erc721-permit2/pull/2).

Code Assumptions

Assumptions noted:

1. Users are implementing their own measures or don't care if this contract accepts signatures outside of the lower half of the ECDSA curve order as described in Appendix F. of the Ethereum Yellow Paper (<https://ethereum.github.io/yellowpaper/paper.pdf>). Read more in finding I-01.
2. For nonces, each user will need no more than 256 slots per bitmap, but they can create up to 2^{248} bitmaps.
3. Forked chains will have a different chain id or else users are aware of risks of signature replay on a fork that re-uses a chain id.

Detailed Findings

Medium Findings

[M-01] SafeTransfer not used

A user can transfer an ERC-721 token to a smart contract that does not have the ability to interact with the token. This would result in the token being non-transferable and locked forever.

Both the single token and multi-token `permitTransferFrom()` functions use the `transferFrom` method on the token.

```
function transferFrom(
    address from,
    address to,
    uint256 tokenId
) public virtual {
    if (to == address(0)) {
        revert ERC721InvalidReceiver(address(0));
    }
    address previousOwner = _update(to, tokenId, _msgSender());
    if (previousOwner != from) {
        revert ERC721IncorrectOwner(from, tokenId, previousOwner);
    }
}
```

Unlike `safeTransferFrom`, the `transferFrom` function does not call `_checkOnERC721Received`. This means that `transferFrom` does not ensure that the recipient is capable of receiving ERC-721 tokens.

In contrast, `safeTransferFrom` checks if the recipient is a contract and, if so, requires it to implement the `onERC721Received` function from the `ERC721Receiver` interface. This check ensures that the receiving contract is aware of the ERC-721 protocol and has the necessary functionality to handle the incoming token safely.

```
function _checkOnERC721Received(
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) private {
    if (to.code.length > 0) {
        try
            IERC721Receiver(to).onERC721Received(
                _msgSender(),
                from,
                tokenId,
                data
            )
        returns (bytes4 retval) {
            if (retval != IERC721Receiver.onERC721Received.selector) {
                revert ERC721InvalidReceiver(to);
            }
        } catch (bytes memory reason) {
            if (reason.length == 0) {
                revert ERC721InvalidReceiver(to);
            } else {
                /// @solidity memory-safe-assembly
                assembly {
                    revert(add(32, reason), mload(reason))
                }
            }
        }
    }
}
```

Recommendation

Consider using `safeTransferFrom` instead of `transferFrom`.

Informational Findings

[I-01] Malleable signatures accepted

The SignatureVerification library is susceptible to signature malleability, however since the contract makes proper use of nonces, we did not identify an immediate security risk.

According to the Ethereum Yellow Paper, a compliant implementation should enforce that the signature's 's' value is within the lower half of the curve order to prevent potential malleability issues. Failing to perform this check could theoretically allow for multiple valid signatures for the same message and private key, which might lead to compatibility issues with other implementations or future upgrades.

A similar issue was reported by Chain Security during their [review of Uniswap's Permit2](https://chainsecurity.com/wp-content/uploads/2022/11/ChainSecurity_Uniswap_Permit2_audit.pdf) (https://chainsecurity.com/wp-content/uploads/2022/11/ChainSecurity_Uniswap_Permit2_audit.pdf).

Recommendation

To address this, use Open Zeppelin's SignatureChecker or add a similar check:

```
if (
  uint256(s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A
) {
  return (address(0), RecoverError.InvalidSignatureS, s);
}
```