Solady ERC721 Security Review

Part of OpenSense Solwaifu Initiative

Shung

July 8, 2023

Table of Contents

1 Introduction

This report is part of OpenSense's Operation Solwaifu initiative to community audit Solady contracts. As part of the initiative, I have volunteered to review the ERC721 contract.

The review took place between June 26, 2023 and July 1, 2023. Over the course of the review I have not found any significant issues.

Disclaimer: This security report is not a guarantee that the system has no other bugs. This is a best effort review of the system by a single person conducted within a limited time period. All the diffs and code samples provided are for demonstration purposes only, are not tested, and have no security guarantees.

About Me - shung: I am an independent smart contract auditor with deep knowledge of EVM, Solidity, and decentralized finance. I have a long track record of developing safe and secure smart contracts at Pangolin. I am also an NFT and crypto enjoyer. If you're in need of smart contract development or auditing services, find me on Twitter.

2 Scope

The review covered the smart contract code hosted at the following repository, version tag, commit hash, and contract names:

Repository: `https://github.com/Vectorized/solady`

Version Tag: v0.0.107

Commit Hash: 7175c21f95255dc7711ce84cc32080a41864abd6

Contract Names:

```
src/
└── tokens/
    └── ERC721.sol
```

3 Custom Storage Layout Review

Solady ERC721 implements a custom storage layout. This storage layout is secure and does not clash within itself or with any native Solidity storage slot. The storage slots are always accessed and written correctly in the contract.

The storage layout is defined in the comments.

```
The ownership data slot of `id` is given by:
```
    mstore(0x00, id)
    mstore(0x1c, _ERC721_MASTER_SLOT_SEED)
    let ownershipSlot := add(id, add(id, keccak256(0x00, 0x20)))
```
Bits Layout:
- [0..159]   `addr`
- [160..223] `extraData`

The approved address slot is given by: `add(1, ownershipSlot)`.

See: https://notes.ethereum.org/%40vbuterin/verkle_tree_eip

The balance slot of `owner` is given by:
```
    mstore(0x1c, _ERC721_MASTER_SLOT_SEED)
    mstore(0x00, owner)
    let balanceSlot := keccak256(0x0c, 0x1c)
```
Bits Layout:
- [0..31]   `balance`
- [32..225] `aux`

The `operator` approval slot of `owner` is given by:
```
    mstore(0x1c, or(_ERC721_MASTER_SLOT_SEED, operator))
    mstore(0x00, owner)
    let operatorApprovalSlot := keccak256(0x0c, 0x30)
```
```

Code 1: ERC721.sol#L79

4 Manual Memory Operations Review

Solady ERC721 almost always manually reads and writes from the memory. With the exception of `_checkOnERC721Received` function, memory is only written to the first two 32 byte chunks. In `_checkOnERC721Received`, memory is written from the safe memory pointer onwards. There is no memory read operation that reads an unknown byte. All memory reads are made to the bytes that are written in the same assembly block. All in all, memory operations in the contract have no errors.

5 Security Review Findings

5.1 No underflow check exists when decrementing user balance

Severity: Low risk

Context: ERC721.sol#L287-L288, ERC721.sol#L555-L556, ERC721.sol#L750-L751

Description

There is no underflow checks when decrementing the `from` balance during a transfer or burn. Although within the constraints of Solady ERC721 the underflow is impossible, an inheriting contract can invalidate this assumption.

Recommendation

Consider either checking for underflow or documenting this feature.

Response

Vectorized (Solady): Added a cautionary comment to remind developers to never violate the ERC721 invariant when overriding. As long as the user balance is always equal to their number of ownership slots, underflow is not possible.

5.2 Precompile `0x04` might be missing on some chains

Severity: Low risk

Context: ERC721.sol#L860

Description

In `_checkOnERC721Received` internal function, there is a call made to the identity precompile at address `0x04`. This precompile does not exist on zkSync, and might also be missing on other chains. Moreover, `_checkOnERC721Received` function lacks any checks to ensure the returned value from the precompile is valid. This silent failure can lead to user provided safe transfer `data` to be passed incorrectly to the NFT recipient.

Recommendation

Consider either using the standard opcodes for memory manipulation or documenting this feature.

Response

Vectorized (Solady): Edited README in a1ae24c to contain a section on EVM compatibility.

5.3 Internal functions are not reused

Severity: Informational

Context: ERC721.sol#616-618, ERC721.sol#606-613, ERC721.sol#572-602, ERC721.sol#796-798, ERC721.sol#813-819, ERC721.sol#771-773, ERC721.sol#787-793, ERC721.sol#664-680, ERC721.sol#702-768, ERC721.sol#687-689

Description

There are internal functions added mostly for the convenience of parent contract. These functions are not reused in relevant external functions. This can lead to blunders where a parent contract overrides an internal function with the expectation that the behaviour of the external function would change.

In the below call graph of internal transfer functions, the transfer logic is at `_transfer by` function. However, none of the internal functions below in the graph are used by `safeTransferFrom`, `safeTranferFrom` with data, or `transferFrom`.

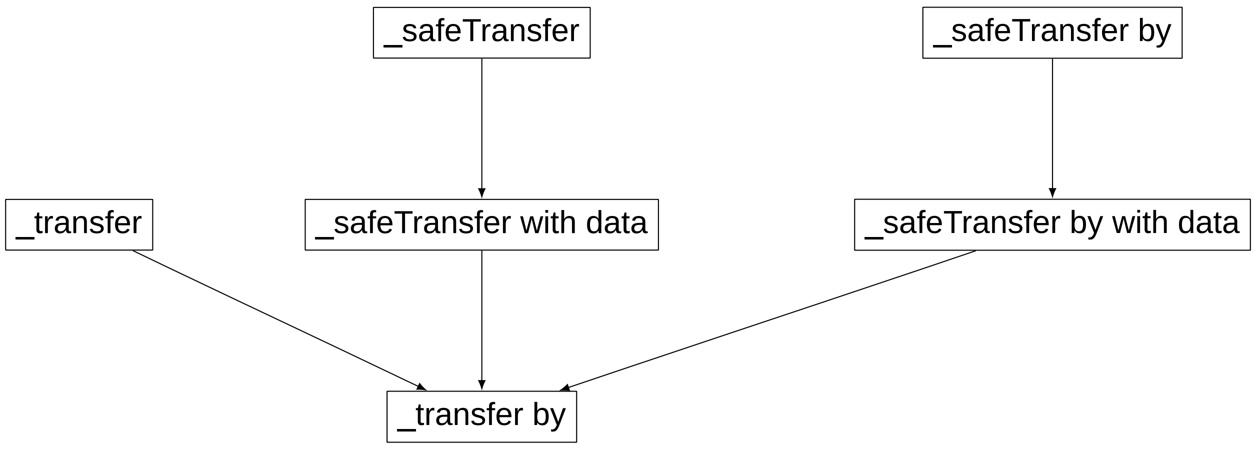

Figure 1: Call graph of internal transfer functions

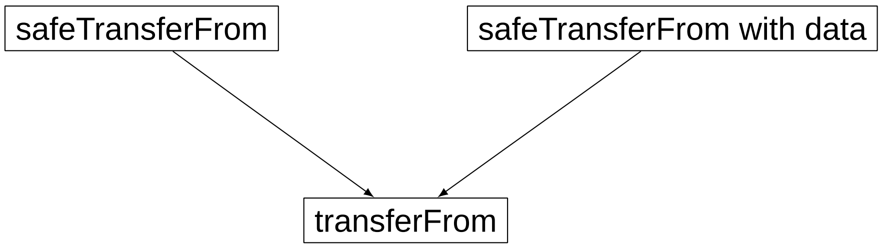

Figure 2: Call graph of external transfer functions

The same design is used for approval functions. An inheriting contract can override `_approve` which would have no effect on `approve`.

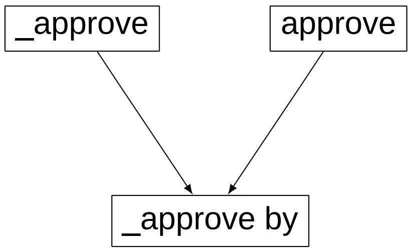

Figure 3: Call graph of approve functions

Similarly, `getApproved` and `setApprovalForAll` re-implements the logic of `_getApproved` and `_setApprovalForAll`, respectively.

Another example is `_isApprovedOrOwner` and `_ownerOf` functions, which are not used for access control in functions like `transferFrom` and `approve`. A deveoper might blunder by overriding `_isApprovedOrOwner` function with the expectation that it is reused in all relevant functions. All in all, this design can subvert developer expectations and lead to mistakes.

Recommendation

Consider thoroughly documenting this design.

Response

Vectorized (Solady): Added a cautionary comment mentioning that code duplicated and manually inlined for performance, and overriding internal functions may not affect the behavior of external functions.

5.4 Treating zero address as a special condition in `by` argument

Severity: Informational

Context: ERC721.sol#L517, ERC721.sol#L628, ERC721.sol#L664, ERC721.sol#L702, ERC721.sol#L796, ERC721.sol#L813

Description

Many internal functions take a `by` argument. Inheriting contract can pass `msg.sender`, `address(0)`, or another address as `by`. If `address(0)` is passed, it is treated as a boolean flag, and the function executes without checking for access control. For that purpose, to be more explicit a boolean can be used. Otherwise, these functions appear more prone to developer errors.

Recommendation

Consider using boolean or thoroughly documenting what to pass as `by`.

Response

Vectorized (Solady): This is intentional to reduce the number of stack operations for performance.

5.5 External functions are marked payable

Severity: Informational

Context: ERC721.sol#L190, ERC721.sol#L241, ERC721.sol#L308, ERC721.sol#L325-L329

Description

Four external functions in the contract are marked payable. Although this complies with ERC-721 standard, user mistakes can lead to locked ether in the contract.

Recommendation

Consider either removing payable modifier from those functions or documenting this feature.

Response

Addressed by an inline comment in PR 496.

5.6 No event is emitted for auxiliary and extra data changes

Severity: Informational

Context: ERC721.sol#L387-L399, ERC721.sol#L413-L425

Description

`_setAux` and `_setExtraData` internal functions update the extra data attached to an address and a token respectively. Although these functions perform important state changes, no event is emitted.

Recommendation

Consider either adding events or documenting that parent contracts using these functions should emit appropriate events.

Response

Vectorized (Solady): This is intentional for performance. A comment is added to remind users to emit their own events if needed.

5.7 Typographical errors in the comments

Severity: Informational

Context: ERC721.sol#L568, ERC721.sol#L596, ERC721.sol#L663

Description

The *managed* should be *manage*:

```
568 /// @dev Returns whether `account` is the owner of token `id`, or is
    ↪ approved to managed it.
```

Code 2: ERC721.sol#L568

The end of sentence should have a period:

```
// Check if `account` is approved to
```

Code 3: ERC721.sol#L596

The *a* should be *an*:

```
/// Emits a {ApprovalForAll} event.
```

Code 4: ERC721.sol#L663

Two forward slashes should be three forward slashes:

```
// - [0..159]   `addr`
// - [160..223] `extraData`
```

Code 5: ERC721.sol#L86-L87

Recommendation

Consider fixing the typos.

Response

Vectorized (Solady): Addressed in this PR [URL].

5.8 Local variable can be set after the conditional revert

Severity: Execution cost optimization

Context: ERC721.sol#L580-L592

Description

In `_isApprovedOrOwner` view function, the return value is initially set to `1`, and then later conditionally set to `0`. However, there is a potential revert in between if token does not exist. Defining `result` as `1` after the `iszero(owner)` check can save gas on reverting case.

```
assembly {
    result := 1
    // Clear the upper 96 bits.
    account := shr(96, shl(96, account))
    // Load the ownership data.
    mstore(0x00, id)
    mstore(0x1c, or(_ERC721_MASTER_SLOT_SEED, account))
    let ownershipSlot := add(id, add(id, keccak256(0x00, 0x20)))
    let owner := shr(96, shl(96, sload(ownershipSlot)))
    // Revert if the token does not exist.
    if iszero(owner) {
        mstore(0x00, 0xceea21b6) // `TokenDoesNotExist()`.
        revert(0x1c, 0x04)
    }
    // Check if `account` is the `owner`.
    if iszero(eq(account, owner)) {
        mstore(0x00, owner)
        // Check if `account` is approved to
        if iszero(sload(keccak256(0x0c, 0x30))) {
            result := eq(account, sload(add(1, ownershipSlot)))
        }
    }
}
```

Code 6: ERC721.sol#L579-L601

Recommendation

Consider moving the line with `result := 1` below the token existence check.

Response

Vectorized (Solady): This is intentional to aid the compiler to produce more optimized bytecode. At the beginning of the function, the compiler implicitly generates a `result := 0`. Placing `result := 1` at the top of the function places it right after the `result := 0`. The bytecode locality helps to compiler to detect that it is unnecessary and remove the `result := 0`.

5.9 Master slot might be unnecessary

Severity: Execution cost optimization

Context: ERC721.sol#L82, ERC721.sol#L95, ERC721.sol#L105

Description

All custom storage slots are derived using a master slot seed constant. The purpose of the seed is not clear. If it is to prevent slot collision, it would be sufficient to hash different lengths of data than 32 bytes. This would reduce the access cost of the storage slots.

Recommendation

Consider the below storage layout.

~~~
The ownership data slot of `id` can be given by:
```
    mstore(0x00, id)
    let ownershipSlot := add(id, add(id, keccak256(0x00, 0x1f)))
```

The approved address slot can be given by: `add(1, ownershipSlot)`.

The balance slot of `owner` can be given by:
```
    mstore(0x00, owner)
    let balanceSlot := keccak256(0x0c, 0x14)
```

The `operator` approval slot of `owner` can be given by:
```
    mstore(0x14, operator)
    mstore(0x00, owner)
    let operatorApprovalSlot := keccak256(0x0c, 0x28)
```
~~~

Response

For upgrade compatibility, we prefer not to change our storage layout unless there is a bug.

5.10 `mstore` can be avoided by passing the length to identity precompile

Severity: Execution cost optimization

Context: ERC721.sol#L858-L860

Description

`_checkOnERC721Received` function calls the identity precompile with the assumption that the precompile exists. I mentioned in a previous finding that this can be dangerous. But if Solady team wants to preserve the current behaviour, it is possible to use the precompile more efficiently.

The function first pushes `data` length to the stack, and copies it to the memory. This is done before calling the identity precompile. It instead possible to use the returndata from identity call to store all the parts of the `data` (including its length) in the same call.

```
assembly {
    // Prepare the calldata.
    let m := mload(0x40)
    let onERC721ReceivedSelector := 0x150b7a02
    mstore(m, onERC721ReceivedSelector)
    mstore(add(m, 0x20), caller()) // The `operator`, which is always `msg.sender`.
    mstore(add(m, 0x40), shr(96, shl(96, from)))
    mstore(add(m, 0x60), id)
    mstore(add(m, 0x80), 0x80)
    let n := mload(data)
    mstore(add(m, 0xa0), n)
    if n { pop(staticcall(gas(), 4, add(data, 0x20), n, add(m, 0xc0), n)) }
    // Revert if the call reverts.
    if iszero(call(gas(), to, 0, add(m, 0x1c), add(n, 0xa4), m, 0x20)) {
        if returndatasize() {
            // Bubble up the revert if the call reverts.
            returndatacopy(0x00, 0x00, returndatasize())
            revert(0x00, returndatasize())
        }
        mstore(m, 0)
    }
    // Load the returndata and compare it.
    if iszero(eq(mload(m), shl(224, onERC721ReceivedSelector))) {
        mstore(0x00, 0xd1a57ed6) // `TransferToNonERC721ReceiverImplementer()`.
        revert(0x1c, 0x04)
    }
}
```

Code 7: ERC721.sol#L849-L875

Recommendation

Consider the below diff.

```
diff --git a/src/tokens/ERC721.sol b/src/tokens/ERC721.sol
index 042c4c3..34a6ce2 100644
--- a/src/tokens/ERC721.sol
+++ b/src/tokens/ERC721.sol
@@ -856,8 +856,7 @@ abstract contract ERC721 {
```

```diff
                mstore(add(m, 0x60), id)
                mstore(add(m, 0x80), 0x80)
                let n := mload(data)
-               mstore(add(m, 0xa0), n)
-               if n { pop(staticcall(gas(), 4, add(data, 0x20), n, add(m,
 ↪  0xc0), n)) }
+               if n { pop(staticcall(gas(), 4, data, add(n, 0x20), add(m,
 ↪  0xa0), add(n, 0x20))) }
                // Revert if the call reverts.
                if iszero(call(gas(), to, 0, add(m, 0x1c), add(n, 0xa4), m,
                 ↪  0x20)) {
                    if returndatasize() {
```

Response

Vectorized (Solady): The `mstore` is still required, as we cannot assume that the memory is zeroized. An inheriting contract may perform operations that can leave regions in the free memory in a non-zero state.

6 Conclusion

I have found Solady ERC721 to be sound and secure by itself. Most of the concerns raised were regarding potential misuses. With the exception of finding 5.4, these were all addressed through documentation and inline comments in PRs 495 and 496.