

# Solady

Tokens & Utils Selection

by Ackee Blockchain

*30.05.2023*



# Contents

1. Document Revisions .....	4
2. Overview .....	5
2.1. Ackee Blockchain .....	5
2.2. Audit Methodology .....	5
2.3. Finding classification .....	7
2.4. Review team .....	9
2.5. Disclaimer .....	9
3. Executive Summary .....	10
Revision 1.0 .....	10
4. Summary of Findings .....	12
5. Report revision 1.0 .....	14
5.1. System Overview .....	14
5.2. Trust Model .....	15
H1: ERC-1155 <code>_setApprovalForAll</code> emits incorrect owner .....	17
M1: ERC-1155 safe transfer re-entrancy .....	19
W1: ERC-1155 safe transfer hooks order inconsistency .....	21
W2: EIP-712 parameters cannot be set .....	23
W3: ERC-20 mint to zero address .....	25
W4: Execution order of Yul arguments relied on .....	26
I1: MerkleProofLib duplicated code .....	28
I2: Token revert checks order inconsistency .....	29
I3: Token approvals to self allowed .....	31
I4: Misleading comments referring to <code>delegatecall</code> .....	32
I5: Increase balance comment in burn function .....	34
Appendix A: How to cite .....	35
Appendix B: Glossary of terms .....	36

Appendix C: Woke outputs ..... 37

    C.1. [H1](#) proof of concept ..... 37

    C.2. [M1](#) proof of concept ..... 37

## 1. Document Revisions

<a href="#"><u>0.1</u></a>	Draft report	30.05.2023
<a href="#"><u>1.0</u></a>	Final report	30.05.2023

## 2. Overview

This document presents our findings in reviewed contracts.

### 2.1. Ackee Blockchain

[Ackee Blockchain](#) is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses [School of Solana](#), [Summer School of Solidity](#) and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, [RockawayX](#).

#### RockawayX

RockawayX is a digital asset venture capital firm supporting founders of web3 companies since early stages. In addition to investing, RockawayX provides liquidity to Defi protocols, runs blockchain infrastructure of nodes and RPCs, develops dashboards ([observatory.zone](#)) and tools for foundations to better decentralize their blockchains ([stakebar.io](#), [smartdelegation.app](#)), funds smart contract audits ([ackeeblockchain.com](#)) and research in accelerating generation of zero knowledge proofs ([maya-zk.com](#)), and organizes hackathons and conferences ([gateway.events](#)).

### 2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and [Woke](#) is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture

is reviewed.

4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzz testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzz tests.

## 2.3. Finding classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

*Low* to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

### Severity

		<i>Likelihood</i>			
		High	Medium	Low	-
<i>Impact</i>	High	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

## Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

## Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.



## 2.4. Review team

Member's Name	Position
Michal Převrátíl	Lead Auditor
Lukáš Böhm	Auditor
Jan Kalivoda	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

## 2.5. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

## 3. Executive Summary

Solady is a library of gas-optimized Solidity code snippets. It is a collection of contracts that can be used as building blocks for other contracts. The library is intended to be used by developers who want to build their own contracts and want to save gas by using already optimized code.

### Revision 1.0

RockawayX engaged Ackee Blockchain to perform a security review of the Solady protocol with a total time donation of 15 engineering days in a period between May 15 and May 29, 2023 and the lead auditor was Michal Převrátíl.

The audit has been performed on the commit [e158762](#) and the scope was the following:

- tokens/ERC20.sol
- tokens/ERC721.sol
- tokens/ERC1155.sol
- utils/SafeTransferLib.sol
- utils/ERC1967Factory.sol
- utils/SignatureCheckerLib.sol
- utils/MerkleProofLib.sol
- utils/EIP712.sol

We began our review by interacting with contracts using [Woke](#) testing framework. We then prepared differential fuzzing tests in Python and started fuzzing the contracts. In parallel, we performed a manual review of the codebase. During the review, we paid special attention to:

- ensuring upper bits of variables shorter than 256 bits are cleared when necessary,
- looking for any memory constraint violations, especially interactions with the free memory pointer,
- ensuring tokens and utility libraries are implemented with respect to corresponding EIPs,
- looking for common issues specific to inline assembly.

Our review resulted in 11 findings, ranging from Info to High severity. The most severe one results in incorrect ownership data emitted in an event which can lead to off-chain applications malfunction (see [H1](#)).

Ackee Blockchain recommends Solady:

- pay special attention when copying existing code blocks from one file to another to avoid introducing bugs,
- reconsider if internal transfer and approval functions are necessary in ERC721 and ERC1155 contracts as they may lead to misleading or incorrect data being emitted as in [H1](#),
- fix the [M1](#) re-entrancy issue,
- deeply look into reported warnings and informational findings.

See [Revision 1.0](#) for the system overview of the codebase.

## 4. Summary of Findings

The following table summarizes the findings we identified during our review.

Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Fix*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
<a href="#">H1: ERC-1155</a> <a href="#">_setApprovalForAll emits</a> <a href="#">incorrect owner</a>	High	<a href="#">1.0</a>	Reported
<a href="#">M1: ERC-1155 safe transfer</a> <a href="#">re-entrancy</a>	Medium	<a href="#">1.0</a>	Reported
<a href="#">W1: ERC-1155 safe transfer</a> <a href="#">hooks order inconsistency</a>	Warning	<a href="#">1.0</a>	Reported
<a href="#">W2: EIP-712 parameters</a> <a href="#">cannot be set</a>	Warning	<a href="#">1.0</a>	Reported
<a href="#">W3: ERC-20 mint to zero</a> <a href="#">address</a>	Warning	<a href="#">1.0</a>	Reported

	Severity	Reported	Status
<a href="#">W4: Execution order of Yul arguments relied on</a>	Warning	<a href="#">1.0</a>	Reported
<a href="#">I1: MerkleProofLib duplicated code</a>	Info	<a href="#">1.0</a>	Reported
<a href="#">I2: Token revert checks order inconsistency</a>	Info	<a href="#">1.0</a>	Reported
<a href="#">I3: Token approvals to self allowed</a>	Info	<a href="#">1.0</a>	Reported
<a href="#">I4: Misleading comments referring to <code>delegatecall</code></a>	Info	<a href="#">1.0</a>	Reported
<a href="#">I5: Increase balance comment in burn function</a>	Info	<a href="#">1.0</a>	Reported

Table 2. Table of Findings

## 5. Report revision 1.0

### 5.1. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

#### Contracts

Contracts we find important for better understanding are described in the following section.

##### ERC20

The ERC20 abstract contract is a base contract for [ERC-20](#) token contracts implementing [ERC-2612](#) permit approvals.

##### ERC721

The ERC721 abstract contract is a base implementation of [ERC-721](#) token contracts.

##### ERC1155

The ERC1155 abstract contract is a base implementation of [ERC-1155](#) token contracts.

##### SafeTransferLib

The SafeTransferLib is a library for safe ETH and [ERC-20](#) token transfers gracefully handling missing function return values.

##### ERC1967Factory

The ERC1967Factory contract serves as a factory for deploying and managing

[ERC-1967](#) proxy contracts.

### **SignatureCheckerLib**

The SafeTransferLib library performs ECDSA and [ERC-1271](#) signature correctness checks.

### **MerkleProofLib**

The MerkleProofLib library implements functions for verifying if a given leaf or a set of leaves belongs to a Merkle tree, given its root hash and a Merkle proof.

### **EIP712**

The EIP712 abstract contract provides helper functions for building the [EIP-712](#) domain separator and preparing the data for signing and verifying signatures.

## **Actors**

This part describes actors of the system, their roles, and permissions.

### **Owner**

Depending on the final implementation of token contracts (ERC20, ERC721, ERC1155) inheriting from abstract contracts provided in Solady, the token contract owner can modify balances, including transferring tokens, minting, and burning tokens. The token contract owner can also modify approvals and allowances.

## **5.2. Trust Model**

Users of tokens (ERC20, ERC721, ERC1155) have to trust the token contract and its final implementation as the provided contracts are abstract, and the logic may be changed due to inheritance. Token users must trust the

addresses they give approvals to and set allowances for.



## H1: ERC-1155 `_setApprovalForAll` emits incorrect owner

*High severity issue*

Impact:	High	Likelihood:	Medium
Target:	tokens/ERC1155.sol	Type:	Logic error

*Listing 1. Excerpt from [ERC1155. setApprovalForAll](#)*

```

728     function _setApprovalForAll(address by, address operator, bool
        isApproved) internal virtual {
729         /// @solidity memory-safe-assembly
730         assembly {
731             // Convert to 0 or 1.
732             isApproved := iszero(iszero(isApproved))
733             // Update the `isApproved` for (`by`, `operator`).
734             mstore(0x20, _ERC1155_MASTER_SLOT_SEED)
735             mstore(0x14, by)
736             mstore(0x00, operator)
737             sstore(keccak256(0x0c, 0x34), isApproved)
738             // Emit the {ApprovalForAll} event.
739             mstore(0x00, isApproved)
740             // forgefmt: disable-next-line
741             log3(0x00, 0x20, _APPROVAL_FOR_ALL_EVENT_SIGNATURE,
        caller(), shr(96, shl(96, operator)))
742         }
743     }

```

Signature of the `ApprovalForAll` event:

```

event ApprovalForAll(address indexed _owner, address indexed _operator,
    bool _approved);

```

### Description

The contract `ERC1155` implements two variants of a `setApprovalForAll`

function. The first variant checks access controls and is public. The second variant (`_setApprovalForAll`) does not check access controls and is internal. The second variant accepts one additional argument, `by`, which is used to set approval from any owner to any operator. The function emits the `ApprovalForAll` event as required by [EIP-1155](#) but uses `caller` (`msg.sender` in Solidity) as an address of the account that gives an approval (the owner).

Typically, the second (internal) variant will be used when the owner is not equal to the function's caller. This will result in incorrect data emitted in the `ApprovalForAll` event, which can lead to incorrect behavior of off-chain services relying on this event.

### Vulnerability scenario

The contract owner calls the `_setApprovalForAll` function for a pre-defined set of owners and operators. The function emits the `ApprovalForAll` event with the owner set to the contract owner instead of the actual token's owner.

There is a proof of concept script in [Woke](#) development and testing framework in [Appendix C](#).

### Recommendation

Use `shr(96, shl(96, by))` instead of `caller()` in the `log3` instruction emitting the `ApprovalForAll` event.

[Go back to Findings Summary](#)

## M1: ERC-1155 safe transfer re-entrancy

*Medium severity issue*

Impact:	High	Likelihood:	Low
Target:	tokens/ERC1155.sol	Type:	Re-entrancy

*Listing 2. Excerpt from [ERC1155.\\_safeTransfer](#)*

```

827         if (_hasCode(to)) _checkOnERC1155Received(from, to, id, amount,
            data);
828         if (_useAfterTokenTransfer()) {
829             _afterTokenTransfer(from, to, _single(id), _single(amount),
            data);
830         }
```

*Listing 3. Excerpt from [ERC1155.\\_safeBatchTransfer](#)*

```

943         if (_hasCode(to)) _checkOnERC1155BatchReceived(from, to, ids,
            amounts, data);
944         if (_useAfterTokenTransfer()) {
945             _afterTokenTransfer(from, to, ids, amounts, data);
946         }
```

### Description

The `_safeTransfer` and `_safeBatchTransfer` functions of the `ERC1155` contract call post-transfer hooks in the reverse order compared to other transfer functions implemented in the same contract. In the reversed order, the external hook, `_checkOnERC1155Received` and `_checkOnERC1155BatchReceived`, is called before the internal hook, `_afterTokenTransfer`. This allows for a re-entrancy attack with the following preconditions:

- the contract inheriting from the `ERC1155` abstract contract uses one of the `_safeTransfer` or `_safeBatchTransfer` functions with an untrusted `to`

address,

- internal `_afterTokenTransfer` hook is used in the inheritor contract, and it performs any state change,
- the inheritor contract does not implement its own re-entrancy protection.

## Vulnerability scenario

Alice uses the `ERC1155` abstract contract to implement her custom ERC-1155 token. The token has a `mint` function that limits the amount of tokens a single address can hold. The verification logic is implemented in the `mint` function, and the information about the amount of tokens held by an address (together with other metadata) is stored in the `_afterTokenTransfer` hook.

Bob calls the `mint` function through his malicious contract that re-enters the `mint` function from the `_checkOnERC1155Received` hook. Because the `_afterTokenTransfer` hook is called after the `_checkOnERC1155Received` external hook, Bob can bypass the verification logic and mint more tokens than allowed.

There is a proof of concept script in [Woke](#) development and testing framework in [Appendix C](#).

## Recommendation

Call the `_afterTokenTransfer` hook before `_checkOnERC1155Received` and `_checkOnERC1155BatchReceived`, respectively.

[Go back to Findings Summary](#)

## W1: ERC-1155 safe transfer hooks order inconsistency

Impact:	Warning	Likelihood:	N/A
Target:	tokens/ERC1155.sol	Type:	Code quality

*Listing 4. Excerpt from [ERC1155. safeTransfer](#)*

```

827         if (_hasCode(to)) _checkOnERC1155Received(from, to, id, amount,
            data);
828         if (_useAfterTokenTransfer()) {
829             _afterTokenTransfer(from, to, _single(id), _single(amount),
            data);
830         }

```

*Listing 5. Excerpt from [ERC1155. safeBatchTransfer](#)*

```

943         if (_hasCode(to)) _checkOnERC1155BatchReceived(from, to, ids,
            amounts, data);
944         if (_useAfterTokenTransfer()) {
945             _afterTokenTransfer(from, to, ids, amounts, data);
946         }

```

### Description

The internal `_afterTokenTransfer` and external `_checkOnERC1155Received` (or `_checkOnERC1155BatchReceived`, respectively) hooks are called in a different order across all safe transfer functions in the ERC1155 contract. This inconsistency can lead to unexpected behavior in an off-chain application in a scenario where both the internal and external hooks emit an event, and the off-chain application relies on the order of the events to be consistent.

### Recommendation

Call the hook functions in the same order in all safe transfer functions in the

ERC1155 contract. It is strongly recommended to call the internal hook before the external hook to fix the [M1](#) issue.

## W2: EIP-712 parameters cannot be set

Impact:	Warning	Likelihood:	N/A
Target:	utils/EIP712.sol	Type:	Standards deviation

Listing 6. Excerpt from [EIP712. buildDomainSeparator](#)

```

151     function _buildDomainSeparator() private view returns (bytes32
        separator) {
152         bytes32 nameHash = _cachedNameHash;
153         bytes32 versionHash = _cachedVersionHash;
154         /// @solidity memory-safe-assembly
155         assembly {
156             let m := mload(0x40) // Load the free memory pointer.
157             mstore(m, _DOMAIN_TYPEHASH)
158             mstore(add(m, 0x20), nameHash)
159             mstore(add(m, 0x40), versionHash)
160             mstore(add(m, 0x60), chainid())
161             mstore(add(m, 0x80), address())
162             separator := keccak256(m, 0xa0)
163         }
164     }

```

### Description

The EIP712 abstract contract implements data preparations for [EIP-712](#) signing. However, the implementation has some limitations in contrast to the EIP:

- the address of the verifying contract cannot be set, i.e., the current implementation assumes the verifying contract will be the same as the contract producing the hash to be signed,
- `salt`, which is an optional parameter serving as a domain separator of last resort, cannot be set.

The current implementation does not allow inheriting the EIP712 contract and overriding necessary functions to make both parameters (`verifyingContract` and `salt`) configurable.

## **Recommendation**

Reconsider making both parameters configurable directly in the abstract contract.



## W3: ERC-20 mint to zero address

Impact:	Warning	Likelihood:	N/A
Target:	tokens/ERC20.sol	Type:	Data validation

### Description

The ERC20 contract allows minting tokens to the zero address. This contrasts with the OpenZeppelin implementation, where such behavior is not allowed.

### Recommendation

Consider checking that the recipient of tokens is not the zero address in the ERC20 `_mint` function.

## W4: Execution order of Yul arguments relied on

Impact:	Warning	Likelihood:	N/A
Target:	utils/SignatureCheckerLib.sol, utils/SafeTransferLib.sol	Type:	Undocumented features utilization

Listing 7. Excerpt from [SignatureCheckerLib.isValidSignatureNow](#)

```

88         isValid := and(
89             and(
90                 // Whether the returndata is the magic value
91                 '0x1626ba7e' (left-aligned).
92                 eq(mload(0x00), f),
93                 // Whether the returndata is exactly 0x20 bytes
94                 (1 word) long.
95                 eq(returndatasize(), 0x20)
96             ),
97             // Whether the staticcall does not revert.
98             // This must be placed at the end of the 'and'
99             clause,
100             // as the arguments are evaluated from right to
101             left.
102             staticcall(
103                 gas(), // Remaining gas.
104                 signer, // The 'signer' address.
105                 m, // Offset of calldata in memory.
106                 add(signatureLength, 0x64), // Length of
107                 calldata in memory.
108                 0x00, // Offset of returndata.
109                 0x20 // Length of returndata to write.
110             )
111         )

```

Listing 8. Excerpt from [SafeTransferLib.safeTransferFrom](#)

```

164         if iszero(
165             and( // The arguments of 'and' are evaluated from right
166                 to left.

```

```

166          // Set success to whether the call reverted, if not
    we check it either
167          // returned exactly 1 (can't just be non-zero
    data), or had no return data.
168          or(eq(mload(0x00), 1), iszero(returndatasize())),
169          call(gas(), token, 0, 0x1c, 0x64, 0x00, 0x20)
170      )
171  ) {

```

## Description

Solady relies on an undocumented behavior of the `solc` compiler that arguments of Yul internal functions are evaluated from the last to the first.

In particular, this was observed in the case of Yul `and`, where the second argument performs an external call and the first argument works with the external call return data.

## Recommendation

Consider rewriting the code to avoid relying on the execution order of Yul arguments, as this behavior is not documented and may change in future versions of the compiler.

## I1: MerkleProofLib duplicated code

Impact:	Info	Likelihood:	N/A
Target:	utils/MerkleProofLib.sol	Type:	Code quality

*Listing 9. Excerpt from [MerkleProofLib.emptyProof](#)*

```

262     function emptyProof() internal pure returns (bytes32[] calldata
      proof) {
263         /// @solidity memory-safe-assembly
264         assembly {
265             proof.length := 0
266         }
267     }

```

*Listing 10. Excerpt from [MerkleProofLib.emptyLeafs](#)*

```

270     function emptyLeafs() internal pure returns (bytes32[] calldata
      leafs) {
271         /// @solidity memory-safe-assembly
272         assembly {
273             leafs.length := 0
274         }
275     }

```

### Description

The MerkleProofLib library implements `emptyProof` and `emptyLeafs` helper functions, both returning an empty `calldata` array of `bytes32`. The functionality of both functions is the same, with names of functions and variables being the only difference.

### Recommendation

Consider merging the two functions into one with a more generic name, e.g. `emptyBytes32Array`.

## I2: Token revert checks order inconsistency

Impact:	Info	Likelihood:	N/A
Target:	tokens/ERC721.sol, tokens/ERC1155.sol	Type:	Code quality

*Listing 11. Excerpt from [ERC1155.safeTransferFrom](#)*

```

196         if iszero(eq(caller(), from)) {
197             mstore(0x00, caller())
198             if iszero(sload(keccak256(0x0c, 0x34))) {
199                 mstore(0x00, 0x4b6e7f18) //
200                 `NotOwnerNorApproved()`.
201                 revert(0x1c, 0x04)
202             }
203         }
204         // Revert if `to` is the zero address.
205         if iszero(to) {
206             mstore(0x00, 0xea553b34) // `TransferToZeroAddress()`.
207             revert(0x1c, 0x04)
208         }

```

*Listing 12. Excerpt from [ERC1155.safeBatchTransferFrom](#)*

```

305         if iszero(to) {
306             mstore(0x00, 0xea553b34) // `TransferToZeroAddress()`.
307             revert(0x1c, 0x04)
308         }
309         // If the caller is not `from`, do the authorization check.
310         if iszero(eq(caller(), from)) {
311             mstore(0x00, caller())
312             if iszero(sload(keccak256(0x0c, 0x34))) {
313                 mstore(0x00, 0x4b6e7f18) //
314                 `NotOwnerNorApproved()`.
315                 revert(0x1c, 0x04)
316             }
317         }

```

## **Description**

Revert checks are performed in a different order across ERC1155 and ERC721 functions. This is an inconsistency.

## **Recommendation**

Perform revert checks of the same type in the same order in the whole project unless this can save a significant amount of gas.

### I3: Token approvals to self allowed

Impact:	Info	Likelihood:	N/A
Target:	tokens/ERC721.sol, tokens/ERC1155.sol	Type:	Data validation

#### Description

The ERC721 and ERC1155 contracts allow calling `approve` and `setApprovalForAll` with `by` and `account` pointing to the same address, effectively giving approval to self. This behavior is prohibited in the OpenZeppelin implementation.

#### Recommendation

Consider adding an extra check that the `by` address is different from the `account` address in ERC721 and ERC1155 approval functions.

## I4: Misleading comments referring to `delegatecall`

Impact:	Info	Likelihood:	N/A
Target:	tokens/ERC721.sol, tokens/ERC1155.sol	Type:	Code quality

Listing 13. Excerpt from [ERC721.\\_checkOnERC721Received](#)

```

856         // Revert if the call reverts.
857         if iszero(call(gas(), to, 0, add(m, 0x1c), add(n, 0xa4), m,
0x20)) {
858             if returndatasize() {
859                 // Bubble up the revert if the delegatecall
reverts.
860                 returndatacopy(0x00, 0x00, returndatasize())
861                 revert(0x00, returndatasize())
862             }
863             mstore(m, 0)
864         }

```

Listing 14. Excerpt from [ERC1155.safeTransferFrom](#)

```

252         // Revert if the call reverts.
253         if iszero(call(gas(), to, 0, add(m, 0x1c), add(0xc4,
data.length), m, 0x20)) {
254             if returndatasize() {
255                 // Bubble up the revert if the delegatecall
reverts.
256                 returndatacopy(0x00, 0x00, returndatasize())
257                 revert(0x00, returndatasize())
258             }
259             mstore(m, 0)
260         }

```

### Description

In the ERC721 and ERC1155 contracts, multiple comments refer to



`delegatecall`, but there is no `delegatecall` instruction and the `call` instruction is used instead.

## Recommendation

Correct the comments to refer to the `call` instruction to avoid confusion.

## I5: Increase balance comment in burn function

Impact:	Info	Likelihood:	N/A
Target:	tokens/ERC1155.sol	Type:	Code quality

Listing 15. Excerpt from [ERC1155.\\_batchBurn](#)

```

684          // Increase and store the updated balance of `to`.
685          {
686              mstore(0x00, mload(add(ids, i)))
687              let fromBalanceSlot := keccak256(0x00, 0x40)
688              let fromBalance := sload(fromBalanceSlot)
689              if gt(amount, fromBalance) {
690                  mstore(0x00, 0xf4d678b8) //
`InsufficientBalance`.
691                  revert(0x1c, 0x04)
692              }
693              sstore(fromBalanceSlot, sub(fromBalance,
amount))
694          }

```

### Description

In the `_batchBurn` function of the ERC1155 contract, there is a comment describing an increase in balance, but the function decreases the balance.

### Recommendation

To avoid confusion, replace `Increase` with `Decrease` in the comment.

## Appendix A: How to cite

Please cite this document as:

[Ackee Blockchain](#), Solady: Tokens & Utils Selection, 30.05.2023.

## Appendix B: Glossary of terms

The following terms might be used throughout the document:

### **Superclass/Ancessor of C**

A contract that C inherits/derives from.

### **Subclass/Child of C**

A contract that inherits/derives from C.

### **Syntactic contract**

A Solidity contract. May have an inheritance chain, and may be deployed.

### **Deployed contract**

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

### **Init/initialization function**

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

### **External entripoint**

A `public` or `external` function.

### **Public/Publicly-accessible function/entripoint**

An `external` or `public` function that can be successfully executed by any network account.

### **Mutating function**

A non-`view` and non-`pure` function.

## Appendix C: Woke outputs

A part of the audit delivery is a test suite with unit and fuzz tests in [Woke](#) development and testing framework. The following section shows proof of concept code for the most severe issues [H1](#) and [M1](#).

### C.1. [H1](#) proof of concept

```
contract ERC1155Mock is ERC1155 {
    function setApprovalForAllUnchecked(address by, address operator, bool
approved) external {
        _setApprovalForAll(by, operator, approved);
    }
}
```

```
@default_chain.connect()
def test_erc1155_events():
    a = default_chain.accounts[0]
    b = default_chain.accounts[1]
    c = default_chain.accounts[2]
    erc1155 = ERC1155Mock.deploy(True, from_=a)

    tx = erc1155.setApprovalForAllUnchecked(a, b, True, from_=c)
    assert tx.events == [ERC1155Mock.ApprovalForAll(a.address, b.address,
True)]
    tx = erc1155.setApprovalForAllUnchecked(a, b, False, from_=c)
    assert tx.events == [ERC1155Mock.ApprovalForAll(a.address, b.address,
False)]
```

### C.2. [M1](#) proof of concept

The victim contract:

```
contract ERC1155Mock is ERC1155 {
    event BeforeTokenTransfer(address from, address to, uint256[] ids,
uint256[] amounts, bytes data);
```

```

    event AfterTokenTransfer(address from, address to, uint256[] ids,
uint256[] amounts, bytes data);

    bool immutable private _enableHooks;

    constructor(bool enableHooks_) {
        _enableHooks = enableHooks_;
    }

    function _useBeforeTokenTransfer() internal view override returns (
bool) {
        return _enableHooks;
    }

    function _useAfterTokenTransfer() internal view override returns (bool)
{
        return _enableHooks;
    }

    function _afterTokenTransfer(
        address from,
        address to,
        uint256[] memory ids,
        uint256[] memory amounts,
        bytes memory data
    ) internal override {
        emit AfterTokenTransfer(from, to, ids, amounts, data);
    }

    ...
}

```

The attacker contract:

```

contract ERC1155ReentrancyAttacker {
    function onERC1155Received(
        address,
        address,
        uint256,
        uint256,
        bytes calldata data
    )

```

```

    ) external returns(bytes4) {
        if (data.length == 0)
            ERC1155Mock(msg.sender).mint(address(this), 1024, 1,
hex"00112233");
        return this.onERC1155Received.selector;
    }

    function onERC1155BatchReceived(
        address,
        address,
        uint256[] calldata,
        uint256[] calldata,
        bytes calldata data
    ) external returns(bytes4) {
        if (data.length == 0)
            ERC1155Mock(msg.sender).mint(address(this), 1024, 1,
hex"00112233");
        return this.onERC1155BatchReceived.selector;
    }
}

```

```

@default_chain.connect()
def test_erc1155_reentrancy():
    a = default_chain.accounts[0]

    erc1155 = ERC1155Mock.deploy(True, from_=a)
    attacker = ERC1155ReentrancyAttacker.deploy(from_=a)

    erc1155.mint(a, 0, 1, b"", from_=a)

    tx = erc1155.safeTransferUnchecked(Address.ZERO, a, attacker, 0, 1,
b"", from_=a)
    assert tx.events == [
        ERC1155Mock.BeforeTokenTransfer(a.address, attacker.address, [0],
[1], bytearray(b"")),
        ERC1155Mock.TransferSingle(a.address, a.address, attacker.address,
0, 1),

        # re-entrant call to mint
        ERC1155Mock.BeforeTokenTransfer(Address.ZERO, attacker.address,
[1024], [1], bytearray(b"\x00\x11\x22\x33")),

```

```
        ERC1155Mock.TransferSingle(attacker.address, Address.ZERO,  
attacker.address, 1024, 1),  
        ERC1155Mock.AfterTokenTransfer(Address.ZERO, attacker.address,  
[1024], [1], bytearray(b"\x00\x11\x22\x33")),  
  
        ERC1155Mock.AfterTokenTransfer(a.address, attacker.address, [0], [  
1], bytearray(b"")),  
    ]
```



# Thank You

Ackee Blockchain a.s.



Prague, Czech Republic



[hello@ackeeblockchain.com](mailto:hello@ackeeblockchain.com)



<https://twitter.com/AckeeBlockchain>