SOLIDIFIED

Audit Report for Animoca Core Library - September 26, 2022

Summary

Audit Report prepared by Solidified covering the Animoca Core Library smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on September 26, 2022, and the results are presented here.

Audited Files

The source code has been supplied in a public source code repository:

https://github.com/animoca/ethereum-contracts

Commit number: `7269914fe12b4625fa13d9b9b2c9e8cfe3066746`

Update: The team provided fixes on October 3, 2022.
Commit number: `da716141a4fa82f5855af80c94ed4c13af98aab6`

Update 2: The team provided further fixes on October 9, 2022.
Commit number: `05666c7112a5637b9ec13b6883cb626982062244`

Intended Behavior

Animoca Core Library is a set of Solidity contracts development libraries.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

SOLIDIFIED

Audit Report for Animoca Core Library - September 26, 2022

Issues Found

Solidified found that the Animoca Core Library contracts contain 1 critical issue, 0 major issues, 2 minor issues, and 6 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	ERC1155Storage.sol: Function _transferToken() miscalculates the to account new balance	Critical	Resolved
2	ForwarderRegistry.sol: In cases where an account owns multiple EIP1271 compliant contracts, a signature provided in setForwarderApproval() can be replayed to approve the same forwarder to any number of these contracts	Minor	Resolved
3	TokenRecoveryBase.sol: Missing account validation in function recoverETH()	Minor	Acknowledged
4	ContractOwnershipStorage.sol: Function constructorInit() can be called multiple times	Note	Acknowledged
5	InterfaceDetectionStorage.sol: Redundant ILLEGAL_INTERFACE_ID check in function supportsInterface()	Note	Acknowledged
6	ERC20MetadataStorage.sol: Function setTokenURI() does not validate the given uri	Note	Acknowledged
7	Consider providing reentrancy documentation warnings for the library's safeTransfer() functions	Note	Resolved
8	ForwarderRegistry.sol and ERC20PermitStorage.sol: The Domain Separators in _calculateDomainSeparator() and DOMAIN_SEPARATOR() do not conform to EIP712	Note	Acknowledged

9	Miscellaneous	Note	Resolved

Critical Issues

1. ERC1155Storage.sol: Function _transferToken() miscalculates the to account new balance

Instead of setting `s.balances[id][to]` to `newToBalance`, the `_transferToken()` function increments the old balance by the new balance.

Recommendation

Replace `s.balances[id][to] += newToBalance` with `s.balances[id][to] = newToBalance`.

Status

Resolved

Major Issues

No major issues have been found.

Minor Issues

2. ForwarderRegistry.sol: In cases where an account owns multiple EIP1271 compliant contracts, a signature provided in setForwarderApproval() can be replayed to approve the same forwarder to any number of these contracts

Assuming a user is the owner of both `EIP1271_A` and `EIP1271_B` (EIP1271 compliant contracts), when they call `setForwarderApproval()` to approve a forwarder for `EIP1271_A`, the same signature can be used by anyone to approve the same forwarder to `EIP1271_B` (assuming the nonce is still zero).

Recommendation
Add `sender` to the hash computed in `_requireValidSignature()`.

Status
Resolved

3. TokenRecoveryBase.sol: Missing account validation in function recoverETH()

The function `recoverETH()` does not validate the contents of the accounts array, which could potentially lead to loss of funds.

Recommendation
Verify that `accounts[i] != address(0)` before transfering ETH to it.

Note

The same issue exists in functions `recoverERC20s()` and `recoverERC721s()`.

Status
Acknowledged. Team's response: "*The recovery functions are admin only and assume that the admin is careful about how to call them. The funds loss situation is true for many other addresses than zero addresses, such as unowned accounts or contracts not designed to receive funds. It does not seem like a zero address check brings any additional safety*".

Informational Notes

4. `ContractOwnershipStorage.sol`: Function `constructorInit()` can be called multiple times

The function `constructorInit()` can be called multiple times by the library's user, thus violating the *initializer pattern*.

Recommendation
Consider adding an `initialized` boolean flag that the function checks against when getting called.

Note
The same issue also exists in: `CheckpointsStorage.constructorInit()`, `PauseStorage.constructorInit()`, `PayoutWalletStorage.constructorInit()`, `ProxyAdminStorage.constructorInit()`, `ERC20Storage.constructorInit()`, `ERC20DetailedStorage.constructorInit()`, `ERC20MetadataStorage.constructorInit()`, `ERC721ContractMetadataStorage.constructorInit()`.

Status

Acknowledged. Team's response: "*This is the intended behavior as `constructorInit` functions are designed to be called from a constructor, as opposed to `proxyInit` which are designed to be called from an external initialization function which may be vulnerable to accidental (or not) later calls*".

5. InterfaceDetectionStorage.sol: Redundant ILLEGAL_INTERFACE_ID check in function supportsInterface()

The statement `if (interfaceId == ILLEGAL_INTERFACE_ID)` is redundant in function `supportsInterface()` since `s.supportedInterfaces[interfaceId]` will always return `false` anyways in such a condition.

Recommendation

Remove the redundant statement in order to save on gas fees.

Status

Acknowledged. Team's response: "*This is an optimization as this check is of insignificant cost compared to reading the storage*".

6. ERC20MetadataStorage.sol: Function setTokenURI() does not validate the given uri

---

The function `setTokenURI()` can be given an invalid string `uri`.

Recommendation

Require that `bytes(uri).length != 0`.

Note

The same issue also exists in: `ERC20MetadataStorage.constructorInit()`, `TokenMetadataPerTokenStorage.setTokenURI()`, `TokenMetadataPerTokenStorage.batchSetTokenURI()`, `TokenMetadataWithBaseURIStorage.setBaseMetadataURI()`.

Status

Acknowledged. Team's response: "*A zero length string is a valid input*".

7. Consider providing reentrancy documentation warnings for the library's safeTransfer() functions

---

Recommendation

Consider warning users that using the library's `safeTansfer()` (or similar) functions opens up their code to reentrancy attacks, since it allows the `to` contract to run arbitrary code via calling `onERC20Received()`.

Status

Resolved

8. ForwarderRegistry.sol and ERC20PermitStorage.sol: The Domain Separators in _calculateDomainSeparator() and DOMAIN_SEPARATOR() do not conform to EIP712

EIP712 suggests the following fields when composing a domain separator:

- `string name`: the user readable name of signing domain, i.e. the name of the DApp or the protocol.

- `string version`: the current major version of the signing domain. Signatures from different versions are not compatible.
- `uint256 chainId`: the EIP-155 chain id. The user-agent should refuse signing if it does not match the currently active chain.
- `address verifyingContract`: the address of the contract that will verify the signature. The user-agent may do contract specific phishing prevention.
- `bytes32 salt`: an disambiguating salt for the protocol. This can be used as a domain separator of last resort.

For more details, please refer to the EIP712 documentation:
https://eips.ethereum.org/EIPS/eip-712

Recommendation
Consider including a `version` and a `salt` when composing the domain separator.

Status
Acknowledged. Team's response: "*From the specification `Protocol designers only need to include the fields that make sense for their signing domain`. ForwarderRegistry not being upgradeable in any way, there is no need for disambiguation other than the `verifyingContract` field. For ERC20Permit, which can be used in an upgradeable scenario, the `version` field is already present. The `salt` field is not required in any of our use-cases*".

9. Miscellaneous

- `InterfaceDetectionStorage.sol`: Unused import `ProxyInitialization`. Resolved.
- `ERC721Storage.sol`: Typo in return variable name of `_tokenHasApproval()`. Resolved.
- `IERC20Permit.sol` and `ERC20PermitStorage.sol`: `Permit` is misspelled as `Permis`. Resolved.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Animoca Brands Limited or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH