Lambdalf ERC Contracts Security Review

Version 1.1

03.04.2024

Conducted by:

nmirchev8 , SR
deth , SR

Table of Contents

1 About Egis Security

We are a team of experienced smart contract researchers, who strive to provide the best smart contract security services possible to DeFi protocols.

Both members of Egis Security have a proven track record on public auditing platforms such as Code4rena, Sherlock & Codehawks, uncovering more than 80 High/Medium severity vulnerabilities, with multiple 2nd, 5th, and 10th place finishes.

2 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

3 Risk classification

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

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	Lambdalf Ethereum Contracts
Repository	https://github.com/lambdalf-dev/ethereum-contracts
Commit hash	d1f35ccfdfda4aaa9fb49133f0a548db3bfb779c
Documentation	In the repo
Methods	Manual review

Scope

/contracts/templates
/contracts/tokens
/contracts/utils

Issues Found

Critical risk	0
High risk	1
Medium risk	3
Low risk	2
Informational	0

5 Findings

5.1 High risk

5.1.1 `tokenByIndex` doesn't check if a token is burned, resulting in counting burned tokens

Severity: *High risk*

Context: ERC721BatchBurnable.sol#L79

Description: Inside ERC721BatchBurnable.sol the _exist function is overridden to check whether the given token id is burned by using a bitmap. Because of the way the accounting works, tokens that are owned by address(0) and are not burned, are still con- sidered to be owned by someone. Whenever a token is burned, we add the tokenId to a bitmap and we send the token to address(0). The problem arises inside tokenOfOwnerByIndex, which doesn't check if the given tokenId exists and so it would count burned tokens as real tokens owned by some- one, which could be crucial for the implementors of this contract. Having in mind that this contract is a library of one of the most famous standards and such an issue could lead to wrong decisions in accounting, or reward distributions, the severity is marked as High. 1. Alice mints 3 tokens with token id's [1, 2, 3] 2. Alice burns tokenId = 2 3. tokenOfOwnerByIndex is called for Alice with index = 2

```
function tokenOfOwnerByIndex(address tokenOwner_, uint256 index_) public view
    virtual override
    returns(uint256) {
  if (tokenOwner_ == address(0)) {
    revert IERC721_INVALID_TOKEN_OWNER();
  }
  address _currentTokenOwner_;
  uint256 _index_ = 1;
  uint256 _ownerBalance_;
  while (_index_ < _nextId) {
    if (_owners[_index_] != address(0)) {
      _currentTokenOwner_ = _owners[_index_];
    }
    if (tokenOwner_ == _currentTokenOwner_) {
      if (index_ == _ownerBalance_) {
        return _index_;
      }
      unchecked { ++_ownerBalance_; }
    }
    unchecked { ++_index_; }
  }
  revert IERC721Enumerable_OWNER_INDEX_OUT_OF_BOUNDS(index_);
}
```

- First iteration `owners[1] = Alice`, `currentTokenOwner = Alice`, ownerBalance = becomes 1
- Second iteration `owners[2] = address(0)`, `currentTokenOwner = Alice` and because we don't check if the token id exists, ownerBalance increases to 2
- Third iteration is fine, as Alice has the token and in the end the function will return `index = 3`, even though it should have returned `index = 2`

Recommendation: Check if `_index_` exist before incrementing `_ownerBalance_`:

```
while (_index_ < _nextId) {
    if (_owners[_index_] != address(0)) {
    _currentTokenOwner_ = _owners[_index_];
    }

+     if (tokenOwner_ == _currentTokenOwner_ && _exists(_index_)) {

-     if (tokenOwner_ == _currentTokenOwner_) {
      if (index_ == _ownerBalance_) {
          return _index_;
      }
    unchecked {
          ++_ownerBalance_;
      }
    }
    unchecked {
      ++_index_;
    }
}
```

Resolution: Fixed

5.2 Medium risk

5.2.1 Signatures can be replayed on different chains

Severity: *Medium risk*

Context: Whitelist.sol.sol#L78

Description: Inside `Whitelist.sol` contract there is signature-based validation, wwhich is using `address account`, `uint8 whitelistId`, `uint256 alloted_` as arguments when `adminSigner` is signing proof. The problem is that if the contracts are deployed on different EVM chains and the admin account is the same, the same signature can be can be used across all deployed chains, as the signature doesn't contain block.chainId

Recommendation: Add `block.chainid` to the digest of the proof, so each proof is different on each chain.

Resolution: Fixed

5.2.2 `ERC721Batch` - `Approve` shouldn't be callable by the approved address for that tokenId

Severity: *Medium risk*

Context: ERC721Batch.sol#93

Description: EIP721 states for approve Throws unless msg.sender is the current NFT owner, or an authorized operator of the current owner. Considering this a Medium severity issue, as it does break access control by the EIP standard and these contracts are supposed to be inherited from and used by other protocols. approve uses isApprovedOrOwner which checks if the spender is approved for that tokenId and if he is an operator, while it should only check if he is an operator.

```
function approve(address to_, uint256 tokenId_) public virtual override {
   address _tokenOwner_ = ownerOf(tokenId_);  // ok
   if (to_ == _tokenOwner_) {
     revert IERC721_INVALID_APPROVAL();
   }
   bool _isApproved_ = _isApprovedOrOwner(_tokenOwner_, msg.sender, tokenId_);
   if (!_isApproved_) {
     revert IERC721_CALLER_NOT_APPROVED(msg.sender, tokenId_);
   }
   _approvals[tokenId_] = to_;
   emit Approval(_tokenOwner_, to_, tokenId_);
  }
```

```
function _isApprovedOrOwner(address tokenOwner_, address operator_, uint256 tokenId_
    ) internal view
    virtual returns(bool isApproved) {
  return operator_ == tokenOwner_ || operator_ == getApproved(tokenId_) ||
         _operatorApprovals[tokenOwner_][operator_];
}
```

Recommendation: When `approve` is called check only if `msg.sender` is the owner, or `isApprovedForAll`

Resolution: Fixed

5.2.3 `ERC1155.sol` Doesn't have a `supportsInterface` function, which is a must by the standard

Severity: *Medium risk*

Context: ERC1155.sol

Description: EIP1155 states: Smart contracts implementing the ERC-1155 standard MUST implement the ERC-165 supportsInterface function and MUST return the constant value true if 0xd9b67a26 is passed through the interfaceID argument. - Any external function that attempts to call supportsInterface will revert, as the function doesn't exist.

Recommendation: Implement the following:

```
function supportsInterface(bytes4 interfaceId_) public view override returns(bool) {
  return interfaceId_ == type(IERC1155).interfaceId ||
         interfaceId_ == type(IERC1155MetadataURI).interfaceId interfaceId_ ==
             type(IERC165).interfaceId;
}
```

Resolution: Acknowledged, both ERC1155 and ERC721Batch don't implement ERC165 on purpose as they're meant to be inherited. The purpose is to simplify inheritance tree. Added comments in natspec to remind user to implement ERC165.

5.3 Low risk

5.3.1 When minting an NFT we don't check if the owner is a contract in `Template721::airdrop`

Severity: *Low risk*

Context: Template721.sol#L169

Description: It's possible to airdrop tokens to a contract, that doesn't implement `onERC721Received` If `airdrop` is called and one of the `accounts_` is an contract, that can't handle NFT's (i.e doesn't implement onERC721Received), the NFT will be lost.

Recommendation: When minting, check if the address is a contract and if it is, apply `_checkOnERC721Received`

Resolution: Acknowledged, `_checkOnERC721Received()` creates a reentrancy vulnerability, as well as a waste of gas.

5.3.2 `balanceOf` function of `ERC721Batch` is very gas inneficient for big collections

Severity: *Low risk*

Context: ERC721Batch.sol#L203

Description: This may be problem if protocol A itegrate protocol B (NFT) and has to check `balanceOf`. Otherwise it would be called with `staticcall`. The function reads from storage inside unbounded loop, which could result in very poor user experience, depending on the protocol implementation.

Recommendation: Unfortunately, for the current accounting logic, we cannot think of another way to check balances.

Resolution: Acknowledged, this is the particularity of this implementation. Added note in natspec.