MiloTruck

LUKSO

Security Review

January, 2024

Contents

Introduction

About MiloTruck

MiloTruck is an independent security researcher who specializes in smart contract audits. Currently, he works as a Senior Auditor at Trust Security and Security Researcher at Spearbit. He is also one of the top wardens on Code4rena.

For private audits or security consulting, please reach out to him on:

- Twitter - @milotruck

You can also request a quote on Code4rena or Cantina to engage them as an intermediary.

Disclaimer

A smart contract security review **can never prove the complete absence of vulnerabilities**. Security reviews are a time, resource and expertise bound effort to find as many vulnerabilities as possible. However, they cannot guarantee the absolute security of the protocol in any way.

Risk Classification

Severity Level

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality.
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality.
- Low - Funds are **not** at risk.

Likelihood

- High - Highly likely to occur.
- Medium - Might occur under specific conditions.
- Low - Unlikely to occur.

Executive Summary

About LUKSO

LUKSO is the digital base layer for the New Creative Economies. It provides creators and users with future-proof tools and standards to unleash their creative force in an open interoperable ecosystem.

Overview

Project Name	LUKSO (LSP4, LSP7, LSP8)
Project Type	ERC-20, ERC-721
Repository	https://github.com/lukso-network/lsp-smart-contracts
Commit Hash	acf5ea902e5b964b4e7dba08331d09b998b84052

Scope

- contracts/LSP4DigitalAssetMetadata/*
- contracts/LSP7DigitalAsset/*
- contracts/LSP8IdentifiableDigitalAsset/*

Issues Found

Severity	Count
High	0
Medium	5
Low	9
Informational	3

Findings

Summary

ID	Description	Severity
M-01	transferBatch() is declared wrongly in the LSP7 and LSP8 specification	Medium
M-02	LSP1 hooks for transfer() are declared wrongly in the LSP-7 specification	Medium
M-03	balanceOf() in LSP8CompatibleERC721.sol deviates from the ERC-721 specification	Medium
M-04	tokenURI() in LSP8CompatibleERC721.sol deviates from the ERC-721 specification	Medium
M-05	LSP1 hooks for authorizeOperator() in LSP8CompatibleERC721.sol is missing the isAuthorized boolean	Medium
L-01	tokenURI() in LSP8CompatibleERC721.sol is incompatible with VerifiableURI	Low
L-02	Missing virtual keyword on burn() in LSP8Burnable.sol	Low
L-03	Approval event is emitted twice on approve() in LSP8CompatibleERC721InitAbstract.sol	Low
L-04	Approval event is wrongly emitted during transfer in LSP7CompatibleERC20.sol	Low
L-05	tokenIds and dataKeys can have different lengths in getDataBatchForTokenIds()	Low
L-06	revokeOperator() does not protect against the double-spending allowance attack	Low
L-07	_beforeTokenTransfer hook should occur before balance checks in LSP7DigitalAssetCore.sol	Low
L-08	Missing token existence check before _beforeTokenTransfer hook in _mint()	Low
L-09	tokenOwner should not be allowed to change after _beforeTokenTransfer hook in _transfer()	Low
I-01	_existsOrError() check in isOperatorFor() is redundant in LSP8IdentifiableDigitalAssetCore.sol	Informational
I-02	Inconsistencies in the LSP-7 specification	Informational
I-03	Inconsistencies in the LSP-8 specification	Informational

Medium Severity Findings

M-01: `transferBatch()` is declared wrongly in the LSP7 and LSP8 specification

Context

- LSP-7-DigitalAsset.md
- LSP-8-IdentifiableDigitalAsset.md
- LSP7DigitalAssetCore.sol#L319-L325
- LSP8IdentifiableDigitalAssetCore.sol#L382-L388

Description

In the LSP-7 specification, the `force` parameter in `transferBatch()` is defined as `bool`:

```
function transferBatch(..., bool force, bytes[] memory data) external;
```

Note that this declaration exists in two places in the specification - under transferBatch and in the interface cheatsheet.

However, `force` is actually a `bool[]` array in the code:

```
function transferBatch(
    address[] memory from,
    address[] memory to,
    uint256[] memory amount,
    bool[] memory force,
    bytes[] memory data
) public virtual override {
```

`transferBatch()` is also wrongly defined in the LSP-8 specification.

This discrepancy between the specification and code could cause developers to wrongly implement or integrate with LSP7/LSP8 contracts.

Recommendation

Modify the LSP-7 and LSP-8 specification to include `bool[] memory force` in `transferBatch()` instead.

LUKSO: Fixed in commit fcabab4 as recommended.

M-02: LSP1 hooks for `transfer()` are declared wrongly in the LSP-7 specification

Context

- LSP-7-DigitalAsset.md
- LSP7DigitalAssetCore.sol#L623-L626

Description

In the LSP-7 specification, under "LSP1 Hooks" for transfer, it states:

> `data`: The data sent SHOULD be packed encoded and contain the `sender` (address), `receiver` (address), `amount` (uint256) and the data (bytes) respectively.

This is the data that the token sender and recipient's `universalReceiver()` function should be called with. However, the code actually includes an additional parameter at the beginning, which is the caller:

```
bytes memory lsp1Data = abi.encode(msg.sender, from, to, amount, data);

_notifyTokenSender(from, lsp1Data);
_notifyTokenReceiver(to, force, lsp1Data);
```

This discrepancy between the LSP-7 specification and `LSP7DigitalAssetCore.sol` could lead to developers handling `universalReceiver()` callbacks for LSP7 token transfers incorrectly in their contracts.

Recommendation

Modify the LSP-7 specification to include the caller in `data`:

> `data`: The data sent SHOULD be packed encoded and contain the `caller` (address), the `sender` (address), `receiver` (address), `amount` (uint256) and the data (bytes) respectively.

LUKSO: Fixed in commit e2f3feb as recommended.

M-03: `balanceOf()` in `LSP8CompatibleERC721.sol` deviates from the ERC-721 specification

Context

- EIP-721
- LSP8CompatibleERC721.sol#L113-L123

Description

According to the ERC-721 specification, `balanceOf()` must revert if called with the zero address:

```
/// @notice Count all NFTs assigned to an owner
/// @dev NFTs assigned to the zero address are considered invalid, and this
///  function throws for queries about the zero address.
/// @param _owner An address for whom to query the balance
/// @return The number of NFTs owned by `_owner`, possibly zero
function balanceOf(address _owner) external view returns (uint256);
```

This is adhered to in all popular ERC-721 libraries, such as `balanceOf()` in Solmate's `ERC721.sol`:

```
function balanceOf(address owner) public view virtual returns (uint256) {
    require(owner != address(0), "ZERO_ADDRESS");

    return _balanceOf[owner];
}
```

However, in `LSP8CompatibleERC721.sol`, `balanceOf()` does not revert if `tokenOwner` is the zero address.

Recommendation

Modify `balanceOf()` in `LSP8CompatibleERC721.sol` to revert when `tokenOwner == address(0)`:

```
  function balanceOf(
      address tokenOwner
  )
      public
      view
      virtual
      override(IERC721, LSP8IdentifiableDigitalAssetCore)
      returns (uint256)
  {
+     require(tokenOwner != address(0), "LSP8CompatibleERC721: address(0) has no balance");
      return super.balanceOf(tokenOwner);
  }
```

LUKSO: `LSP8CompatibleERC721.sol` has been deprecated in PR #845.

M-04: `tokenURI()` in `LSP8CompatibleERC721.sol` deviates from the ERC-721 specification

Context

- EIP-721
- LSP8CompatibleERC721.sol#L143-L161

Description

According to the ERC-721 specification, `tokenURI()` must revert when the function is called with an invalid token ID:

```
/// @notice A distinct Uniform Resource Identifier (URI) for a given asset.
/// @dev Throws if `_tokenId` is not a valid NFT. URIs are defined in RFC
///  3986. The URI may point to a JSON file that conforms to the "ERC721
///  Metadata JSON Schema".
function tokenURI(uint256 _tokenId) external view returns (string);
```

However, in `LSP8CompatibleERC721.sol`, `tokenURI()` does not check that the provided token ID exists.

Recommendation

Modify `tokenURI()` in `LSP8CompatibleERC721.sol` to revert when called with a non-existent token ID:

```
  function tokenURI(
-     uint256 /* tokenId */
+     uint256 tokenId,
  ) public view virtual returns (string memory) {
+     _existsOrError(bytes32(tokenId));
```

LUKSO: `LSP8CompatibleERC721.sol` has been deprecated in PR #845.

M-05: LSP1 hooks for `authorizeOperator()` in `LSP8CompatibleERC721.sol` is missing the `isAuthorized` boolean

Context

- LSP-8-IdentifiableDigitalAsset.md
- LSP8IdentifiableDigitalAssetCore.sol#L272-L277
- LSP8CompatibleERC721.sol#L366-L371

Description

In the LSP-8 specification, under "LSP1 Hooks" for authorizeOperator, it states:

> `data`: The data sent SHOULD be abi encoded and contain the `tokenOwner` (address), `tokenId` (bytes32), `isAuthorized` (boolean), and the `operatorNotificationData` (bytes) respectively.

This is the data that the operator's `universalReceiver()` function should be called with.

`authorizeOperator()` in `LSP8IdentifiableDigitalAssetCore.sol` follows the specification and includes the `isAuthorized` boolean:

```
bytes memory lsp1Data = abi.encode(
    msg.sender,
    tokenId,
    true, // authorized
    operatorNotificationData
);
```

However, this isn't the case for `authorizeOperator()` in `LSP8CompatibleERC721.sol`, which is missing the `isAuthorized` boolean:

```
bytes memory lsp1Data = abi.encode(
    msg.sender,
    tokenId,
    operatorNotificationData
);
operator.notifyUniversalReceiver(_TYPEID_LSP8_TOKENOPERATOR, lsp1Data);
```

If developers use `LSP8CompatibleERC721.sol` to implement their LSP-8 tokens, operators will receive incorrect data in their `universalReceiver()` functions when calling `approve()` or `authorizeOperator()`.

Recommendation

Modify `authorizeOperator()` in `LSP8CompatibleERC721.sol` to include the `isAuthorized` boolean:

```
  bytes memory lsp1Data = abi.encode(
      msg.sender,
      tokenId,
+     true,
      operatorNotificationData
  );
```

LUKSO: `LSP8CompatibleERC721.sol` has been deprecated in PR #845.

Low Severity Findings

L-01: `tokenURI()` in `LSP8CompatibleERC721.sol` is incompatible with VerifiableURI

Context

- LSP8CompatibleERC721.sol#L151-L161
- LSP-4-DigitalAsset-Metadata.md
- LSP-2-ERC725YJSONSchema.md

Description

In `LSP8CompatibleERC721.sol`, `tokenURI()` retrieves the LSP8 token's URI stored in LSP4Metadata as such:

```
bytes memory data = _getData(_LSP4_METADATA_KEY);

// offset = bytes4(hashSig) + bytes32(contentHash) -> 4 + 32 = 36
uint256 offset = 36;

bytes memory uriBytes = data.slice(offset, data.length - offset);
return string(uriBytes);
```

The decoding scheme above is based on AssetURL, which has `bytes4` and `bytes32` hashes appended to the front of the URL.

However, AssetURL is now deprecated in favor of VerifiableURI, which is what is stored at the LSP4Metadata key. As such, `tokenURI()` will return corrupted data as it cannot decode VerifiableURI.

Recommendation

Modify `tokenURI()` in `LSP8CompatibleERC721.sol` to decode the data stored in `_LSP4_METADATA_KEY` based on VerifiableURI's encoding scheme.

LUKSO: `LSP8CompatibleERC721.sol` has been deprecated in PR #845.

L-02: Missing virtual keyword on `burn()` in `LSP8Burnable.sol`

Context

- LSP8Burnable.sol#L24

Description

In `LSP8Burnable.sol`, `burn()` is declared without the `virtual` keyword:

```
function burn(bytes32 tokenId, bytes memory data) public {
```

As such, inherited contracts will not be able to override the `burn()` function, even though it is meant to be overridable.

Recommendation

Add the `virtual` keyword to `burn()`:

```
- function burn(bytes32 tokenId, bytes memory data) public {
+ function burn(bytes32 tokenId, bytes memory data) public virtual {
```

LUKSO: Fixed in PR #835 as recommended.

L-03: `Approval` event is emitted twice on `approve()` in `LSP8CompatibleERC721InitAbstract.sol`

Context

- LSP8CompatibleERC721InitAbstract.sol#L237-L243
- LSP8CompatibleERC721InitAbstract.sol#L366

Description

In `LSP8CompatibleERC721InitAbstract.sol`, the `Approval` event is emitted once in `approve()`:

```
function approve(
    address operator,
    uint256 tokenId
) public virtual override {
    authorizeOperator(operator, bytes32(tokenId), "");
    emit Approval(tokenOwnerOf(bytes32(tokenId)), operator, tokenId);
}
```

It is then emitted again in `authorizeOperator()` with the same arguments:

```
emit Approval(tokenOwnerOf(tokenId), operator, uint256(tokenId));
```

As such, whenever `approve()` is called, the `Approval` event will be emitted twice.

Recommendation

Do not emit the `Approval` event in `approve()`, which is consistent with `LSP8CompatibleERC721.sol`:

```
  function approve(
      address operator,
      uint256 tokenId
  ) public virtual override {
      authorizeOperator(operator, bytes32(tokenId), "");
-     emit Approval(tokenOwnerOf(bytes32(tokenId)), operator, tokenId);
  }
```

LUKSO: `LSP8CompatibleERC721InitAbstract.sol` has been deprecated in PR #845.

L-04: `Approval` event is wrongly emitted during transfer in `LSP7CompatibleERC20.sol`

Context

- LSP7CompatibleERC20.sol#L199-L214
- LSP7CompatibleERC20InitAbstract.sol#L212-L227
- LSP7DigitalAssetCore.sol#L539-L564

Description

In `LSP7CompatibleERC20.sol`, `_updateOperator` is overridden to emit ERC20's `Approval` event:

```
function _updateOperator(
    ...
) internal virtual override {
    ...
    emit IERC20.Approval(tokenOwner, operator, amount);
}
```

This is meant to emit the `Approval` event whenever `approve()` is called.

`_updateOperator()` is also called in `_spendAllowance()` in `LSP7DigitalAssetCore.sol`.

As such, whenever `LSP7DigitalAsset::transfer()` or `LSP7CompatibleERC20::transferFrom()` is called, the `Approval` event will be emitted. However, the `Approval` event is only meant to be emitted when `approve()` is called, and not when tokens are transferred, even if the spender's allowance is decreased.

An example of this would be `transferFrom()` in Solmate's `ERC20.sol`, which only emits the `Transfer` event.

Recommendation

In `LSP7CompatibleERC20.sol`, emit the `Approval` event in `approve()` instead of in `_updateOperator()`.

LUKSO: `LSP7CompatibleERC20.sol` has been deprecated in PR #845.

L-05: `tokenIds` and `dataKeys` can have different lengths in `getDataBatchForTokenIds()`

Context

- LSP8IdentifiableDigitalAssetCore.sol#L141-L157

Description

In `LSP8IdentifiableDigitalAssetCore.sol`, the `getDataBatchForTokenIds()` function does not check that the `tokenIds` and `dataKeys` arrays are of the same length:

```
function getDataBatchForTokenIds(
    bytes32[] memory tokenIds,
    bytes32[] memory dataKeys
) public view virtual override returns (bytes[] memory dataValues) {
    dataValues = new bytes[](tokenIds.length);

    for (uint256 i; i < tokenIds.length; ) {
        dataValues[i] = _getDataForTokenId(tokenIds[i], dataKeys[i]);

        // Increment the iterator in unchecked block to save gas
        unchecked {
            ++i;
        }
    }

    return dataValues;
}
```

Therefore, if `dataKeys` has a larger length than `tokenIds`, the function will return data for all token IDs in `tokenIds` instead of reverting.

Recommendation

Consider checking that the length of both arrays are the same:

```
  function getDataBatchForTokenIds(
      bytes32[] memory tokenIds,
      bytes32[] memory dataKeys
  ) public view virtual override returns (bytes[] memory dataValues) {
+     if (tokenIds.length != dataKeys.length) {
+         revert LSP8TokenIdsDataLengthMismatch();
+     }
```

LUKSO: Fixed in PR #836 as recommended.

L-06: `revokeOperator()` does not protect against the double-spending allowance attack

Context

- LSP-7-DigitalAsset.md
- LSP7DigitalAssetCore.sol#L142-L153

Description

In the LSP-7 specification, under authorizeOperator, it states:

> To increase or decrease the authorized amount of an operator, it's advised to call `revokeOperator(..)` function first, and then call `authorizeOperator(..)` with the new amount to authorize, to avoid front-running through an allowance double-spend exploit. Check more information in this document.

This is also mentioned in the NatSpec documentation for `authorizeOperator()`.

However, calling `revokeOperator()` beforehand does not mitigate the double-spending attack vector. Consider the following example:

- Bob grants Alice 100 tokens.
- Bob wants to reduce Alice's allowance to 50 tokens.
- He batches the following transactions together:
  - Calls `revokeOperator()` to set Alice's allowance to 0.
  - Calls `authorizeOperator()` to allow Alice to spend 50 tokens.
- Alice front-runs Bob's transaction and spends her 100 token allowance.
- When Bob's transaction is executed:
  - The call to `revokeOperator()` passes, but does nothing.
  - `authorizeOperator()` sets Alice's allowance to 50 tokens.
- Alice can now spend another 50 tokens, for a total of 150 tokens.

As demonstrated above, since `revokeOperator()` can be called while the operator has no allowance, a malicious operator can simply front-run the call to `revokeOperator()` to achieve the same double-spending attack.

Recommendation

Modify the LSP-7 specification and `authorizeOperator()`'s NatSpec to recommend using `decreaseAllowance()`, and remove the recommendation for `revokeOperator()`.

LUKSO: Fixed in PR #834 as recommended.

L-07: `_beforeTokenTransfer` hook should occur before balance checks in `LSP7DigitalAssetCore.sol`

Context

- LSP7DigitalAssetCore.sol#L490-L495
- LSP7DigitalAssetCore.sol#L602-L607

Description

In `_transfer()` and `_burn()`, the `_beforeTokenTransfer` hook is called after the balance of the `from` address is checked:

```
uint256 balance = _tokenOwnerBalances[from];
if (amount > balance) {
    revert LSP7AmountExceedsBalance(balance, from, amount);
}

_beforeTokenTransfer(from, address(0), amount, data);
```

As such, if the `from` address balance is decreased in the `_beforeTokenTransfer` hook and becomes lower than `amount`, it will not be caught.

In `LSP7DigitalAssetCore.sol`, this will cause both functions to revert later on with an arithmetic overflow, instead of the `LSP7AmountExceedsBalance` error.

Recommendation

In both functions, consider calling `_beforeTokenTransfer` before checking if the balance of the `from` address is sufficient. This is what Openzeppelin's `ERC20.sol` does.

LUKSO: Fixed in PR #843 as recommended.

L-08: Missing token existence check before `_beforeTokenTransfer` hook in `_mint()`

Context

- LSP8IdentifiableDigitalAssetCore.sol#L509-L514

Description

`_mint()` only checks if `tokenId` already exists after the `_beforeTokenTransfer` hook:

```
_beforeTokenTransfer(address(0), to, tokenId, data);

// Check that `tokenId` was not minted inside the `_beforeTokenTransfer` hook
if (_exists(tokenId)) {
    revert LSP8TokenIdAlreadyMinted(tokenId);
}
```

This allows `_mint()` to be called when `tokenId` exists, as long as it is burned in the `_beforeTokenTransfer` hook. For example:

- Assume the `_beforeTokenTransfer` hook contains an external call to the caller.
- An attacker does the following:
    - Call a function that calls `_mint()` with `tokenId = 1`.
    - In the `_beforeTokenTransfer` hook, call a function that burns `tokenId = 1`.
- As such, he has called `_mint()` even when the `tokenId` is already minted.

Although the exploit is dependent on how developers use the `LSP8IdentifiableDigitalAssetCore` library and it is unclear how this could cause harm to users, preventing such an exploit path from being possible will probably be safer for users.

Recommendation

In `_mint()`, add another existence check before the `_beforeTokenTransfer` hook:

```
+ // Check that `tokenId` was not minted inside the `_beforeTokenTransfer` hook
+ if (_exists(tokenId)) {
+     revert LSP8TokenIdAlreadyMinted(tokenId);
+ }

  _beforeTokenTransfer(address(0), to, tokenId, data);

  // Check that `tokenId` was not minted inside the `_beforeTokenTransfer` hook
  if (_exists(tokenId)) {
      revert LSP8TokenIdAlreadyMinted(tokenId);
  }
```

This is what Openzeppelin's `ERC721.sol` does, which mitigates against the scenario described above.

LUKSO: Fixed in PR #844 as recommended.

L-09: `tokenOwner` should not be allowed to change after `_beforeTokenTransfer` hook in `_transfer()`

Context

- LSP8IdentifiableDigitalAssetCore.sol#L635-L639

Description

In `_transfer()`, after the `_beforeTokenTransfer` hook, the owner of `tokenId` is re-fetched in case it changed during the hook:

```
_beforeTokenTransfer(from, to, tokenId, data);

// Re-fetch and update `tokenOwner` in case `tokenId`
// was transferred inside the `_beforeTokenTransfer` hook
tokenOwner = tokenOwnerOf(tokenId);
```

This makes LSP8 transfers dangerous if `_beforeTokenTransfer` ever transfers execution to the caller. For example:

- Assume the following:
  - The `_beforeTokenTransfer` hook contains an external call to the caller.
  - A contract requires users to stake their LSP8 tokens for benefits.
- An attacker does the following:
  - Call `transferFrom()` to transfer his LSP8 token to another address belonging to him.
  - In the `_beforeTokenTransfer` hook, he stakes his LSP8 token into the contract.
  - After the `_beforeTokenTransfer` hook is executed, `tokenOwner` is now the contract.
  - `_transfer()` will transfer the LSP8 token back to the attacker's second address.

Recommendation

In `_transfer()`, check that `tokenOwner` and `tokenOwnerOf(tokenId)` are the same after the `_beforeTokenTransfer` hook:

```
  _beforeTokenTransfer(from, to, tokenId, data);

- // Re-fetch and update `tokenOwner` in case `tokenId`
- // was transferred inside the `_beforeTokenTransfer` hook
- tokenOwner = tokenOwnerOf(tokenId);
+ if (tokenOwner != tokenOwnerOf(tokenId)) {
+     revert LSP8TokenOwnerChanged();
+ }
```

This is what Openzeppelin's `ERC721.sol` does, which mitigates against the scenario described above.

LUKSO: Fixed in PR #846 as recommended.

Informational Findings

I-01: `_existsOrError()` check in `isOperatorFor()` is redundant in `LSP8IdentifiableDigitalAssetCore.sol`

Context

- LSP8IdentifiableDigitalAssetCore.sol#L328-L335

Description

In `LSP8IdentifiableDigitalAssetCore.sol`, `isOperatorFor()` calls `_existsOrError()`:

```
function isOperatorFor(
    address operator,
    bytes32 tokenId
) public view virtual override returns (bool) {
    _existsOrError(tokenId);

    return _isOperatorOrOwner(operator, tokenId);
}
```

`_existsOrError()` checks that the owner of `tokenId` is not `address(0)`, and reverts if it is the case.

However, `_isOperatorOrOwner()` calls `tokenOwnerOf()`, which has the same check:

```
function tokenOwnerOf(
    bytes32 tokenId
) public view virtual override returns (address) {
    address tokenOwner = _tokenOwners[tokenId];

    if (tokenOwner == address(0)) {
        revert LSP8NonExistentTokenId(tokenId);
    }
```

Therefore, calling `_existsOrError()` in `isOperatorFor()` is redundant.

Recommendation

Consider removing the `_existsOrError()` check in `isOperatorFor()`.

LUKSO: Fixed in PR #837 as recommended.

I-02: Inconsistencies in the LSP-7 specification

Context

- LSP-7-DigitalAsset.md

Description

The following functions are specified wrongly in the LSP-7 specification.

authorizedAmountFor
- Parameters is missing the `tokenOwner` parameter

increaseAllowance
- Requirements is missing the following checks:
  - `operator` cannot be calling address.
  - `operator` cannot be the zero address.

decreaseAllowance
- There is no *Requirements* section, which should include the following checks:
  - `operator` cannot be calling address.
  - `operator` cannot be the zero address.
  - `subtractedAmount` must be less than the operator's current allowance.

transfer
- *Requirements* is missing the following check:
  - `from` and `to` cannot be the same address.

transferBatch
- *Requirements* is missing the following check:
  - `from` and `to` cannot be the same address.
- This condition in *Requirements* should include the `force` and `data` parameter as well: `from`, `to`, `amount` lists are the same length.

Interface Cheat Sheet
- `transferBatch()` - `force` parameter should be a `bool` array, `bool[] memory force` instead.
- `batchCalls()` - `data` parameter should be `memory` instead of `calldata`.

Recommendation

Modify the LSP-7 specification as mentioned above.

I-03: Inconsistencies in the LSP-8 specification

Context

- LSP-8-IdentifiableDigitalAsset.md

Description

The following functions are specified wrongly in the LSP-8 specification.

authorizeOperator
- *Requirements* is missing the following check:
  - `operator` cannot be authorized for `tokenId`.

revokeOperator
- *Parameters* is missing the `notify` parameter.
- *Requirements* is missing the following check:
  - `operator` must be authorized.

isOperatorFor
- The following condition under *Requirements* is wrong:
  - "caller must be current `tokenOwner` of `tokenId`." - `msg.sender` isn't checked in the function.

getOperatorsOf
- The following conditions under *Requirements* are wrong:
  - "caller must be current `tokenOwner` of `tokenId`." - `msg.sender` isn't checked in the function.
  - "`operator` cannot be calling address." - there is no `operator` parameter.

transfer
- The following statement should be under *Requirements*, not *Parameters*:
  `from` and `to` cannot be the same address.

transferBatch
- This condition in *Requirements* should include the `force` and `data` parameter as well:
  `from`, `to`, `amount` lists are the same length.

Interface Cheat Sheet
- `transferBatch()` - `force` parameter should be a `bool` array, `bool[] memory force` instead.
- `batchCalls()` - `data` parameter should be `memory` instead of `calldata`.
- `transferOwnership()` should not have the `override` keyword.
- `renounceOwnership()` should not have the `virtual` keyword.

Recommendation

Modify the LSP-8 specification as mentioned above.