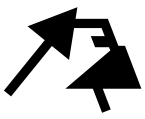
LUKSO LSPs Audit Report

Oct 20, 2022



WATCHPUG

Table of Contents

Summary

This report has been prepared for LUKSO LSPs Audit Report smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	**LUKSO LSPs Audit Report**
Codebase	**https://github.com/lukso-network/lsp-smart-contracts**
Commit	**d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89**
Language	**Solidity**

Audit Summary

Delivery Date	**Oct 20, 2022**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**23**

[WP-C1] LSP-6: `AddressPermissions:AllowedERC725YKeys:<address>` Improper access control

Critical

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L420-L469

```
420     function _verifyAllowedERC725YKeys(address from, bytes32[] memory inputKeys)
    internal view {
421         bytes memory allowedERC725YKeysEncoded =
    ERC725Y(target).getAllowedERC725YKeysFor(from);
422
423         // whitelist any ERC725Y key
424         if (
425             // if nothing in the list
426             allowedERC725YKeysEncoded.length == 0 ||
427             // if not correctly abi-encoded array
428             !LSP2Utils.isEncodedArray(allowedERC725YKeysEncoded)
429         ) return;
430
431         bytes32[] memory allowedERC725YKeys =
    abi.decode(allowedERC725YKeysEncoded, (bytes32[]));
432
433         uint256 zeroBytesCount;
434         bytes32 mask;
435
436         // loop through each allowed ERC725Y key retrieved from storage
437         for (uint256 ii = 0; ii < allowedERC725YKeys.length; ii =
    GasLib.uncheckedIncrement(ii)) {
438             // required to know which part of the input key to compare against the
    allowed key
439             zeroBytesCount = _countTrailingZeroBytes(allowedERC725YKeys[ii]);
440
441             // loop through each keys given as input
442             for (uint256 jj = 0; jj < inputKeys.length; jj =
    GasLib.uncheckedIncrement(jj)) {
443                 // skip permissions keys that have been previously checked and
    "nulled"
```

```
444                 if (inputKeys[jj] == bytes32(0)) continue;
445
446                 // use a bitmask to discard the last `n` bytes of the input key
    (where `n` = `zeroBytesCount`)
447                 // and compare only the relevant parts of each ERC725Y keys
448                 //
449                 // for an allowed key =
    0xcafecafecafecafecafecafecafecafe00000000000000000000000000000000
450                 //
451                 //                         |------compare this
    part-------|------discard this part--------|
452                 //                         v                               v
    v
453                 //               mask =
    0xffffffffffffffffffffffffffffffff00000000000000000000000000000000
454                 //        & input key =
    0xcafecafecafecafecafecafecafecafe00000000000000000000000011223344
455                 //
456                 mask = bytes32(type(uint256).max) << (8 * zeroBytesCount);
457
458                 if (allowedERC725YKeys[ii] == (inputKeys[jj] & mask)) {
459                     // if the input key matches the allowed key
460                     // make it null to mark it as allowed
461                     inputKeys[jj] = bytes32(0);
462                 }
463             }
464         }
465
466         for (uint256 ii = 0; ii < inputKeys.length; ii =
    GasLib.uncheckedIncrement(ii)) {
467             if (inputKeys[ii] != bytes32(0)) revert NotAllowedERC725YKey(from,
    inputKeys[ii]);
468         }
469     }
```

Per to the Specification of LSP-6:

> `AddressPermissions:AllowedERC725YKeys:<address>` can be used to set a range of allowed ERC725Y data keys (= **partial data keys**), by setting:

> - some part of the data keys as the exact data key bytes
> - the rest of the data key bytes as 0 bytes.

The trailing zero bytes are used as the wildcard character:

> By setting the value to `0xeafec4d89fa9619884b6000000000000000000000000000000000000000000` in the list of allowed ERC725Y data keys, one address can set any data key **starting with the first word** `SupportedStandards:...` .

However, because part of the trailing zero bytes (the beginning part) can also be part of the concrete value, this can result in an extensively unauthorized expansion of permissions.

PoC

Let's say there is a hotel called "LUKSO Hotel", fully management/control by a smart contract.

1. Alice paid for room number `1000` ;
2. Alice was granted the permissions to the keys prefixed by "0x11111111A4001000", while `11111111` is the identifier of this smart hotel application, `A400` means room control, and finally `1000` is the room number;

Alice can now set "0x11111111A4001000...D002" to `1` to open the door (`D002` controls the door) of room `1000` , and set "0x11111111A4001000...D003" to `1` to open turn on the light, etc.

However, since all the other rooms with their room number starting with `1` (eg: `1001` , `1002` , ..., `1999`) have their permissions keys start with "0x11111111A4001", Alice can actually control all these rooms: open their doors and lights.

This is improper access control.

Recommendation

Consider using `AddressPermissions:AllowedERC725YKeys:<address>` only for the percesie match (full match) and use a separate `AddressPermissions:AllowedERC725YKeysPrefixes:<address>` for permissions granted by prefixes:

| 1 byte | 31 bytes |

The first byte stores the length of the prefix that should be matched, and the max length of the prefix is 31 bytes. Otherwise, it should use `AddressPermissions:AllowedERC725YKeys:<address>` for a full match.

[WP-H2] LSP-6: `msg.value` of the RelayCall is not signed, making it vulnerable to frontrun attack

High

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/b3ff7a66439cbfed779e3ee0992d5b9047ad208e/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L109-L133

```
109  function executeRelayCall(
110      bytes memory signature,
111      uint256 nonce,
112      bytes calldata payload
113  ) public payable returns (bytes memory) {
114      bytes memory blob = abi.encodePacked(
115          block.chainid,
116          address(this), // needs to be signed for this keyManager
117          nonce,
118          payload
119      );
120
121      address signer = keccak256(blob).toEthSignedMessageHash().recover(signature);
122
123      if (!_isValidNonce(signer, nonce)) {
124          revert InvalidRelayNonce(signer, nonce, signature);
125      }
126
127      // increase nonce after successful verification
128      _nonceStore[signer][nonce >> 128]++;
129
130      _verifyPermissions(signer, payload);
131
132      return _executePayload(payload);
133  }
```

When the relayer sends the transaction to the mempool and before it gets minted, the calldata and `signature` will be made public for a short period of time. This opens an opportunity for

the attackers to frontrun the transaction with the same signature and calldata, but a different `msg.value` than the original request to the relayer.

https://github.com/ERC725Alliance/ERC725/blob/dac17fd2633af7c149d9cf456eb0d9c9b51e6021/implementations/contracts/ERC725XCore.sol#L39-L45

```
39      function execute(
40          uint256 operation,
41          address to,
42          uint256 value,
43          bytes memory data
44      ) public payable virtual override onlyOwner returns (bytes memory) {
45          require(address(this).balance >= value, "ERC725X: insufficient balance");
```

https://github.com/ERC725Alliance/ERC725/blob/dac17fd2633af7c149d9cf456eb0d9c9b51e6021/implementations/contracts/ERC725XCore.sol#L71

```
71  if (operation == OPERATION_DELEGATECALL) return _executeDelegateCall(to, value,
    data);
```

PoC

1. Alice creates a RelayCall request to transfer 1 ETH to Bob;
2. The attacker frontrun the transaction with the same calldata and signature but changed the amount from 1 ETH to 1000 ETH.

Alice will end up paying 999 ETH more than expected.

Recommendation

`msg.value` should be part of the data to be signed.

[WP-M3] LSP-6: Failed transactions should not block the queue of the channel

Medium

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/b3ff7a66439cbfed779e3ee0992d5b9047ad208e/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L109-L156

```
109  function executeRelayCall(
110      bytes memory signature,
111      uint256 nonce,
112      bytes calldata payload
113  ) public payable returns (bytes memory) {
114      bytes memory blob = abi.encodePacked(
115          block.chainid,
116          address(this), // needs to be signed for this keyManager
117          nonce,
118          payload
119      );
120
121      address signer = keccak256(blob).toEthSignedMessageHash().recover(signature);
122
123      if (!_isValidNonce(signer, nonce)) {
124          revert InvalidRelayNonce(signer, nonce, signature);
125      }
126
127      // increase nonce after successful verification
128      _nonceStore[signer][nonce >> 128]++;
129
130      _verifyPermissions(signer, payload);
131
132      return _executePayload(payload);
133  }
134
135  /**
136  * @notice execute the received payload (obtained via `execute(...)` and
     `executeRelayCall(...)`)
137  *
```

```
138  * @param payload the payload to execute
139  * @return bytes the result from calling the target with `_payload`
140  */
141 function _executePayload(bytes calldata payload) internal returns (bytes memory) {
142
143     // solhint-disable avoid-low-level-calls
144     (bool success, bytes memory returnData) = target.call{value: msg.value, gas:
    gasleft()}(
145         payload
146 );
147 bytes memory result = Address.verifyCallResult(
148     success,
149     returnData,
150     "LSP6: Unknown Error occured when calling the linked target contract"
151 );
152
153     emit Executed(msg.value, bytes4(payload));
154     return result.length != 0 ? abi.decode(result, (bytes)) : result;
155
156 }
```

In the current implementation, a user can submit multiple transactions with the same `channelId` to the relayer.

However, unlike the conventional clients (eg, `Metamask`), the scheduled transactions will be blocked by the transactions that can not be executed successfully.

For example:

Alice submits 3 transactions:

1. Buy 1 BTC with 20k USDC;
2. Buy 10 ETH with 130k USDC;
3. Transfer 1k USDC to Bob.

Due to market price change, transaction #1 can not be successfully executed. Therefore, both #2 and #3 will have to wait.

Recommendation

Consider catching the error of the low-level call and still bumping the nonce so that the subsequent transactions can continue to be executed.

[WP-M4] Returnbomb in ERC165Checker

Medium

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/eb9919fb82414657b721cfb6e70f06659366af52/contracts/Custom/ERC165Checker.sol#L117-L129

```
117     function supportsERC165Interface(address account, bytes4 interfaceId)
118         internal
119         view
120         returns (bool)
121     {
122         bytes memory encodedParams = abi.encodeWithSelector(
123             IERC165.supportsInterface.selector,
124             interfaceId
125         );
126         (bool success, bytes memory result) = account.staticcall{gas:
    30000}(encodedParams);
127         if (result.length < 32) return false;
128         return success && abi.decode(result, (uint256)) > 0;
129     }
```

A regular solidity call will automatically copy bytes to memory without consideration of gas.

This is to say, a low-level solidity call will copy any amount of bytes to local memory. When bytes are copied from returndata to memory, the memory expansion cost is paid. This means that when using a standard solidity call, the callee can ”returnbomb” the caller, imposing an arbitrary gas cost. Because this gas is paid by the caller and in the caller’s context, it can cause the caller to run out of gas and halt execution.

See: https://github.com/nomad-xyz/ExcessivelySafeCall

The same issue was fixed in OpenZeppelin’s ERC165Checker with this PR: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/3587

[WP-M5] LSP-6: Signature collisions can be exploited with phishing attack

Medium

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/b3ff7a66439cbfed779e3ee0992d5b9047ad208e/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L622-L639

```
622  function _verifyAllowedFunction(address from, bytes4 functionSelector) internal
     view {
623      bytes memory allowedFunctions = ERC725Y(target).getAllowedFunctionsFor(from);
624
625      // whitelist any function
626      if (
627          // if nothing in the list
628          allowedFunctions.length == 0 ||
629          // if not correctly abi-encoded array of bytes4[]
630          !LSP2Utils.isBytes4EncodedArray(allowedFunctions)
631      ) return;
632
633      bytes4[] memory allowedFunctionsList = abi.decode(allowedFunctions,
     (bytes4[]));
634
635      for (uint256 ii = 0; ii < allowedFunctionsList.length; ii =
     GasLib.uncheckedIncrement(ii)) {
636          if (functionSelector == allowedFunctionsList[ii]) return;
637      }
638      revert NotAllowedFunction(from, functionSelector);
639  }
```

https://github.com/lukso-network/lsp-smart-contracts/blob/eb9919fb82414657b721cfb6e70f06659366af52/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L478-L536

```
478     function _verifyCanExecute(
479         address from,
480         bytes32 permissions,
481         bytes calldata payload
482     ) internal view {
483         uint256 operationType = uint256(bytes32(payload[4:36]));
484         require(operationType < 5, "LSP6KeyManager: invalid operation type");
485 
486         require(
487             operationType != OPERATION_DELEGATECALL,
488             "LSP6KeyManager: operation DELEGATECALL is currently disallowed"
489         );
490 
491         uint256 value = uint256(bytes32(payload[68:100]));
492 
493         // prettier-ignore
494         bool isContractCreation = operationType == OPERATION_CREATE ||
    operationType == OPERATION_CREATE2;
495         bool isCallDataPresent = payload.length > 164;
496 
497         // SUPER operation only applies to contract call, not contract creation
498         bool hasSuperOperation = isContractCreation
499             ? false
500             :
    permissions.hasPermission(_extractSuperPermissionFromOperation(operationType));
501 
502         if (isCallDataPresent && !hasSuperOperation) {
503             _requirePermissions(from, permissions,
    _extractPermissionFromOperation(operationType));
504         }
505 
506         bool hasSuperTransferValue =
    permissions.hasPermission(_PERMISSION_SUPER_TRANSFERVALUE);
507 
508         if (value != 0 && !hasSuperTransferValue) {
509             _requirePermissions(from, permissions, _PERMISSION_TRANSFERVALUE);
510         }
511 
512         // Skip on contract creation (CREATE or CREATE2)
513         if (isContractCreation) return;
514 
515         // Skip if caller has SUPER permissions for operations
516         if (hasSuperOperation && isCallDataPresent && value == 0) return;
```

```
517
518         // Skip if caller has SUPER permission for value transfers
519         if (hasSuperTransferValue && !isCallDataPresent && value != 0) return;
520
521         // Skip if both SUPER permissions are present
522         if (hasSuperOperation && hasSuperTransferValue) return;
523
524         // CHECK for ALLOWED ADDRESSES
525         address to = address(bytes20(payload[48:68]));
526         _verifyAllowedAddress(from, to);
527
528         if (to.code.length != 0) {
529             // CHECK for ALLOWED STANDARDS
530             _verifyAllowedStandard(from, to);
531
532             // CHECK for ALLOWED FUNCTIONS
533             // extract bytes4 function selector from payload passed to
    ERC725X.execute(...)
534             if (payload.length >= 168) _verifyAllowedFunction(from,
    bytes4(payload[164:168]));
535         }
536     }
```

The design/implementation of `ALLOWED_FUNCTIONS` / `ALLOWED_ADDRESSES` makes it hard to control the permissions precisely.

For instance, when Alice wants to give Bob permission to call `contractA#methodA` and `contractB#methodB`, she can only set the following:

- `ALLOWED_ADDRESSES` to `[contractA, contractB]`
- `ALLOWED_FUNCTIONS` to `[methodA, methodB]`

By side effect, this also gives Bob the permission to call `contractA#methodB` and `contractB#methodA`, which is unexpected.

This will be even more tricky if we considered the case of signature collisions, which actually can happen quite often.

Attack Vector

A sophisticated attacker can create an innocent-looking smart contract with a method that collisions with a more sensitive method, say, `ERC20.transfer()` .

The victim may be tricked into approving the attacker to add such a signature to `ALLOWED_FUNCTIONS` without limiting the `ALLOWED_ADDRESSES` , and the attacker ends up being able to transfer all the ERC20 tokens.

Or the attacker can trick the victim to whitelist a harmless method on contractA, and another harmless methodB on contractB, but actually make use of methodC which has the same signature as methodB on contractA for malicious activities.

Root Cause

The root cause for this issue is that the atom of permissions should be the combination of address and signature.

Recommendation

Consider combining the two-layer permissions into one layer, called `ALLOWED_ADDRESSES_FUNCTIONS` , the items should be formatted as `<address>:<methodSig>` , for instance:

- `contractA:methodA` ;
- `ANY_ADDRESS:methodB` ;
- `contractB:ANY_SIG` .

[WP-M6] LSP-6: `_PERMISSION_CHANGEPERMISSIONS` should be able to delete a `AddressPermissions[index]`

Medium

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/eb9919fb82414657b721cfb6e70f06659366af52/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L386-L418

```
386     function _verifyCanSetPermissionsArray(
387         bytes32 key,
388         bytes memory value,
389         address from,
390         bytes32 permissions
391     ) internal view {
392         // key = AddressPermissions[] -> array length
393         if (key == _LSP6KEY_ADDRESSPERMISSIONS_ARRAY) {
394             uint256 arrayLength = uint256(bytes32(ERC725Y(target).getData(key)));
395             uint256 newLength = uint256(bytes32(value));
396 
397             if (newLength > arrayLength) {
398                 _requirePermissions(from, permissions,
    _PERMISSION_ADDPERMISSIONS);
399             } else {
400                 _requirePermissions(from, permissions,
    _PERMISSION_CHANGEPERMISSIONS);
401             }
402 
403             return;
404         }
405 
406         // key = AddressPermissions[index] -> array index
407         bytes memory valueAtIndex = ERC725Y(target).getData(key);
408 
409         if (valueAtIndex.length == 0) {
410             _requirePermissions(from, permissions, _PERMISSION_ADDPERMISSIONS);
411         } else {
412             _requirePermissions(from, permissions, _PERMISSION_CHANGEPERMISSIONS);
413         }
```

```
414
415         if (value.length != 20) {
416             revert AddressPermissionArrayIndexValueNotAnAddress(key, value);
417         }
418     }
```

Given:

- Alice has `_PERMISSION_ADDPERMISSIONS`;
- Bob has `_PERMISSION_CHANGEPERMISSIONS`;
- Current length of `AddressPermissions[]` is `2`;

1. Alice added 3 new addresses: A, B, and C; `AddressPermissions[]` length is `5`;
2. Bob delete the 3 addresses: A, B, and C by change the `AddressPermissions[]` length to `2`;
3. Alice try to add another new address D:
    - extend `AddressPermissions[]` length to `3` will success;
    - set `AddressPermissions[2]` will fail as `valueAtIndex2 !== 0`.

Recommendation

Change to:

```
386     function _verifyCanSetPermissionsArray(
387         bytes32 key,
388         bytes memory value,
389         address from,
390         bytes32 permissions
391     ) internal view {
392         // key = AddressPermissions[] -> array length
393         if (key == _LSP6KEY_ADDRESSPERMISSIONS_ARRAY) {
394             uint256 arrayLength = uint256(bytes32(ERC725Y(target).getData(key)));
395             uint256 newLength = uint256(bytes32(value));
396
397             if (newLength > arrayLength) {
398                 _requirePermissions(from, permissions,
    _PERMISSION_ADDPERMISSIONS);
399             } else {
400                 _requirePermissions(from, permissions,
    _PERMISSION_CHANGEPERMISSIONS);
```

```
401             }
402
403             return;
404         }
405
406         // key = AddressPermissions[index] -> array index
407         bytes memory valueAtIndex = ERC725Y(target).getData(key);
408
409         if (valueAtIndex.length == 0) {
410             _requirePermissions(from, permissions, _PERMISSION_ADDPERMISSIONS);
411         } else {
412             _requirePermissions(from, permissions, _PERMISSION_CHANGEPERMISSIONS);
413         }
414
415         if (value.length != 20 && value.length != 0) {
416             revert AddressPermissionArrayIndexValueNotAnAddress(key, value);
417         }
418     }
```

[WP-M7] LSP-6: `CALL` permission with empty `AllowedAddresses` and `AllowedFunctions` should not be equal to `SUPER_CALL`

Medium

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/eb9919fb82414657b721cfb6e70f06659366af52/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L571-L588

```
571     function _verifyAllowedAddress(address from, address to) internal view {
572         bytes memory allowedAddresses =
    ERC725Y(target).getAllowedAddressesFor(from);
573
574         // whitelist any address
575         if (
576             // if nothing in the list
577             allowedAddresses.length == 0 ||
578             // if not correctly abi-encoded array of address[]
579             !LSP2Utils.isEncodedArrayOfAddresses(allowedAddresses)
580         ) return;
581
582         address[] memory allowedAddressesList = abi.decode(allowedAddresses,
    (address[]));
583
584         for (uint256 ii = 0; ii < allowedAddressesList.length; ii =
    GasLib.uncheckedIncrement(ii)) {
585             if (to == allowedAddressesList[ii]) return;
586         }
587         revert NotAllowedAddress(from, to);
588     }
```

In the current implementation, one with `CALL` permission but empty bytes (= 0x) of `AllowedAddresses`, `AllowedFunctions`, `AllowedStandards` will be able to call any address on any method.

In other words, "all addresses are whitelisted", "all `bytes4` function selectors are whitelisted", "allowed to interact with any contracts, whether they implement a specific standard interface

or not".

That's equivalent to the `SUPER_CALL` permission.

This can be very error-prone, especially when the owner tries to remove/downgrade the privilege level of a certain target.

PoC

A service called Aave Protector provides automated health factor management for users.

1. Alice approved the Aave Protector by whitelisting Aave's address in `AllowedAddresses`;
2. A few months later, Aave Protector announced that they were shouting down the service; Alice then removed Aave's address from `AllowedAddresses` of the service;

Expected Results:

Aave Protector is now not allowed to call any address;

Actual Results:

Aave Protector is now able to call not only Aave but also any other addresses.

Root Cause

An unset array (empty bytes `0x`, means no restriction) can very easily be misinterpreted as an empty list (`[]`, means not allowed to call any address).

There should be a very explicit way to indicate "no restriction" or "allowed to call any address" as this represents a very high privilege.

Recommendation

Consider changing to:

1. Both unset array (empty bytes `0x`) and empty list (`[]`) means "no access" / "not allowed to call any address".
2. Using a specific value to represent "no restriction", eg, `[24217c0b4f5b311c0ef0cf3bad1b2710ec22dc2b]` (`bytes20(keccak256("AllowedAddresses:ANY") == 0x24217c0b4f5b311c0ef0cf3bad1b2710ec22dc2b`

).

[WP-M8] LSP-7: Not fully compatible with ERC20/ERC777 on transfer / mint / burn 0 values

Medium

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/eb9919fb82414657b721cfb6e70f06659366af52/contracts/LSP7DigitalAsset/LSP7DigitalAssetCore.sol#L287-L316

```
287      function _transfer(
288          address from,
289          address to,
290          uint256 amount,
291          bool force,
292          bytes memory data
293      ) internal virtual {
294          if (amount == 0) revert LSP7TransferAmountIsZero();
295
     @@ 296,315 @@
316      }
```

https://github.com/lukso-network/lsp-smart-contracts/blob/eb9919fb82414657b721cfb6e70f06659366af52/contracts/LSP7DigitalAsset/LSP7DigitalAssetCore.sol#L205-L226

```
205      function _mint(
206          address to,
207          uint256 amount,
208          bool force,
209          bytes memory data
210      ) internal virtual {
211          if (amount == 0) revert LSP7MintAmountIsZero();
212
     @@ 213,225 @@
```

```
226     }
```

https://github.com/lukso-network/lsp-smart-contracts/blob/eb9919fb82414657b721cfb6e70f06659366af52/contracts/LSP7DigitalAsset/LSP7DigitalAssetCore.sol#L240-L272

```
240     function _burn(
241         address from,
242         uint256 amount,
243         bytes memory data
244     ) internal virtual {
245         if (amount == 0) revert LSP7BurnAmountIsZero();
246
    @@ 247,271 @@
272     }
```

A recent commit 733e85d5723312f6871b6889242247ca6dc75fe3 introduced a new restriction which no longer allows transfer / mint / burn with 0 values.

However, both ERC20 and ERC777 explicitly requires that the implementation `MUST` support transfer / mint / burn 0 `amount` .

And practically, there are quite a lot of protocols that expected the tokens to be able to transfer / mint / burn 0.

https://github.com/ethereum/EIPs/blob/f589107dbf87ddc37e5c891f6fbb9b12a9b913c2/EIPS/eip-20.md?plain=1#L100

```
95  #### transfer
96
97  Transfers `_value` amount of tokens to address `_to`, and MUST fire the `Transfer`
    event.
98  The function SHOULD `throw` if the message caller's account balance does not have
    enough tokens to spend.
99
100 *Note* Transfers of 0 values MUST be treated as normal transfers and fire the
    `Transfer` event.
101
```

```
102 `` `js
103 function transfer(address _to, uint256 _value) public returns (bool success)
104 `` `
```

https://github.com/ethereum/EIPs/blob/f589107dbf87ddc37e5c891f6fbb9b12a9b913c2/EIPS/eip-20.md?plain=1#L116

```
108 #### transferFrom
109
110 Transfers `_value` amount of tokens from address `_from` to address `_to`, and MUST fire the `Transfer` event.
111
112 The `transferFrom` method is used for a withdraw workflow, allowing contracts to transfer tokens on your behalf.
113 This can be used for example to allow a contract to transfer tokens on your behalf and/or to charge fees in sub-currencies.
114 The function SHOULD `throw` unless the `_from` account has deliberately authorized the sender of the message via some mechanism.
115
116 *Note* Transfers of 0 values MUST be treated as normal transfers and fire the `Transfer` event.
117
118 `` `js
119 function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
120 `` `
```

ERC777:

https://github.com/ethereum/EIPs/blob/f589107dbf87ddc37e5c891f6fbb9b12a9b913c2/EIPS/eip-777.md?plain=1#L477

```
477 *NOTE*: Sending an amount of zero (`0`) tokens is valid and MUST be treated as a regular send.
```

https://github.com/ethereum/EIPs/blob/f589107dbf87ddc37e5c891f6fbb9b12a9b913c2/EIPS/eip-777.md?plain=1#L624

```
624   *NOTE*: Minting an amount of zero (`0`) tokens is valid and MUST be treated as a
      regular mint.
```

https://github.com/ethereum/EIPs/blob/f589107dbf87ddc37e5c891f6fbb9b12a9b913c2/EIPS/eip-777.md?plain=1#L709

```
709   *NOTE*: Burning an amount of zero (`0`) tokens is valid and MUST be treated as a
      regular burn.
```

Recommendation

Consider removing the restriction about 0 amount. Or, explicitly document that LSP-7 does not support transfer / mint / burn 0 amount with emphasis.

[WP-M9] LSP-2: `LSP2Utils.isEncodedArray()` Incomplete implementation

Medium

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP2ERC725YJSONSchema/LSP2Utils.sol#L198-L222

```
198     /**
199      * Verifying if `data` is an encoded array
200      * @param data The value that is to be verified
201      */
202     function isEncodedArray(bytes memory data) internal pure returns (bool) {
203         uint256 nbOfBytes = data.length;
204
205         // 1) there must be at least 32 bytes to store the offset
206         if (nbOfBytes < 32) return false;
207
208         // 2) there must be at least the same number of bytes specified by
209         // the offset value (otherwise, the offset points to nowhere)
210         uint256 offset = uint256(bytes32(data));
211         if (nbOfBytes < offset) return false;
212
213         // 3) there must be at least 32 x length bytes after offset
214         uint256 arrayLength = data.toUint256(offset);
215
216         //   32 bytes word (= offset)
217         // + 32 bytes word (= array length)
218         // + remaining bytes that make each element of the array
219         if (nbOfBytes < (offset + 32 + (arrayLength * 32))) return false;
220
221         return true;
222     }
```

https://github.com/GNSPS/solidity-bytes-utils/blob/v0.8.0/contracts/BytesLib.sol#L375

```
374     function toUint256(bytes memory _bytes, uint256 _start) internal pure returns
    (uint256) {
375         require(_bytes.length >= _start + 32, "toUint256_outOfBounds");
376         uint256 tempUint;
377
378         assembly {
379             tempUint := mload(add(add(_bytes, 0x20), _start))
380         }
381
382         return tempUint;
383     }
```

When:

- `isEncodedArray(abi.encode(uint256(32)))`

Result:

- Expected: returns `false`
- Actual: revert with message ”toUint256_outOfBounds”

Impact

When setting permissions array (eg, `_verifyPermissions() -> _verifyCanSetPermissions() -> isEncodedArrayOfAddresses()`), if the value of the array is not a valid ABI encoded array, it supposed to throw a custom error: `InvalidABIEncodedArray()` .

Due to the wrong implementation, it may revert with the error: ”toUint256_outOfBounds”.

While the impact in the current implementation is relatively negligible, but given that `LSP2Utils` is a broadly used library, on the other out-of-scope contracts or new contracts, this issue may result in other unexpected behavior which can cause a more severe impact.

Recommendation

Consider changing to:


```
198     /**
199      * Verifying if `data` is an encoded array
200      * @param data The value that is to be verified
201      */
202     function isEncodedArray(bytes memory data) internal pure returns (bool) {
203         uint256 nbOfBytes = data.length;
204 
205         // 1) there must be at least 32 bytes to store the offset
206         if (nbOfBytes < 32) return false;
207 
208         // 2) there must be at least 32 x length bytes after offset
209         uint256 offset = uint256(bytes32(data));
210         if (nbOfBytes < offset + 32) return false;
211         uint256 arrayLength = data.toUint256(offset);
212 
213         //   32 bytes word (= offset)
214         // + 32 bytes word (= array length)
215         // + remaining bytes that make each element of the array
216         if (nbOfBytes < (offset + 32 + (arrayLength * 32))) return false;
217 
218         return true;
219     }
```

[WP-L11] LSP-7: `LSP7CompatibleERC20` is not fully compatible with the conventional implementation of ERC20 on the emission of `Approval` events

Low

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/eb9919fb82414657b721cfb6e70f06659366af52/contracts/LSP7DigitalAsset/LSP7DigitalAssetCore.sol#L173-L194

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP7DigitalAsset/extensions/LSP7CompatibleERC20.sol#L50-L57

```
50      function transferFrom(
51          address from,
52          address to,
53          uint256 amount
54      ) public virtual returns (bool) {
55          transfer(from, to, amount, true, "");
56          return true;
57      }
```

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP7DigitalAsset/extensions/LSP7CompatibleERC20.sol#L101-L108

```
101     function _burn(
102         address from,
103         uint256 amount,
104         bytes memory data
105     ) internal virtual override {
106         super._burn(from, amount, data);
107         emit Transfer(from, address(0), amount);
108     }
```

```
173     function _updateOperator(
174         address tokenOwner,
175         address operator,
176         uint256 amount
177     ) internal virtual {
178         if (operator == address(0)) {
179             revert LSP7CannotUseAddressZeroAsOperator();
180         }
181
182         // tokenOwner is always their own operator, no update required
183         if (operator == tokenOwner) {
184             return;
185         }
186
187         _operatorAuthorizedAmount[tokenOwner][operator] = amount;
188
189         if (amount != 0) {
190             emit AuthorizedOperator(operator, tokenOwner, amount);
191         } else {
192             emit RevokedOperator(operator, tokenOwner);
193         }
194     }
```

The conventional ERC20 implementation (OpenZeppelin's) will emit an `Approval` event whenever the allowance amount changes, including:

- `approve()`
- `transferFrom()`
- `burnFrom()`

However, in `LSP7CompatibleERC20`, only the `approve()` method will emit `Approval` events.

This makes the services that rely on ERC20's `Approval` events malfunction when working with `LSP7CompatibleERC20` tokens.

Recommendation

Consider overwriting `LSP7DigitalAssetCore`'s `_updateOperator` to emit `Approval` events. And also, calling `_updateOperator` in `transferFrom` and `_burn` when called by an operator.

[WP-L12] LSP-6: `getNonce(address,uint256)` should be changed to `getNonce(address,uint128)`

Low

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L75-L78

```
75      function getNonce(address from, uint256 channelId) public view returns
    (uint256) {
76          uint128 nonceId = uint128(_nonceStore[from][channelId]);
77          return (uint256(channelId) << 128) | nonceId;
78      }
```

When `channelId > type(uint128).max`, `getNonce(from, longChannelId)` may return a wrong nonce, the signature generated based on that nonce can not be used: `executeRelayCall() -> _isValidNonce(signer, nonce)` will return `false`.

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L166-L171

```
166     function _isValidNonce(address from, uint256 idx) internal view returns (bool)
    {
167         // idx % (1 << 128) = nonce
168         // (idx >> 128) = channel
169         // equivalent to: return (nonce == _nonceStore[_from][channel]
170         return (idx % (1 << 128)) == (_nonceStore[from][idx >> 128]);
171     }
```

Recommendation

Consider changing to:


```
75      function getNonce(address from, uint128 channelId) public view returns
(uint256) {
76          uint128 nonceId = uint128(_nonceStore[from][channelId]);
77          return (uint256(channelId) << 128) | nonceId;
78      }
```

[WP-L13] LSP-6: `LSP6KeyManagerCore._executePayload()` should return `result` directly

Low

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L135-L156

```
135       /**
136        * @notice execute the received payload (obtained via `execute(...)` and `executeRelayCall(...)`)
137        *
138        * @param payload the payload to execute
139        * @return bytes the result from calling the target with `_payload`
140        */
141       function _executePayload(bytes calldata payload) internal returns (bytes memory) {
142
143           // solhint-disable avoid-low-level-calls
144           (bool success, bytes memory returnData) = target.call{value: msg.value, gas: gasleft()}(
145               payload
146           );
147           bytes memory result = Address.verifyCallResult(
148               success,
149               returnData,
150               "LSP6: Unknown Error occured when calling the linked target contract"
151           );
152
153           emit Executed(msg.value, bytes4(payload));
154           return result.length != 0 ? abi.decode(result, (bytes)) : result;
155
156       }
```

The LSP-6 specs says that:

`execute() -> _executePayload()` :

> *Returns:* `bytes` , the returned data as ***abi-encoded bytes***

The current implementation decodes the raw results before returning.

This also introduces unnecessary information loss:

Retruning the raw `result` allows the caller to determin whether that's a function call with no returns or it's the return of `execute()` by checking if `returnBytes.length == 0` .

[WP-I14] LSP-6: `executeRelayCall()`'s parameter `nonce` can be renamed to `channelNonce` for better readability

Informational

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/b3ff7a66439cbfed779e3ee0992d5b9047ad208e/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L109-L133

```
109  function executeRelayCall(
110      bytes memory signature,
111      uint256 nonce,
112      bytes calldata payload
113  ) public payable returns (bytes memory) {
114      bytes memory blob = abi.encodePacked(
115          block.chainid,
116          address(this), // needs to be signed for this keyManager
117          nonce,
118          payload
119      );
120
121      address signer = keccak256(blob).toEthSignedMessageHash().recover(signature);
122
123      if (!_isValidNonce(signer, nonce)) {
124          revert InvalidRelayNonce(signer, nonce, signature);
125      }
126
127      // increase nonce after successful verification
128      _nonceStore[signer][nonce >> 128]++;
129
130      _verifyPermissions(signer, payload);
131
132      return _executePayload(payload);
133  }
```

https://github.com/lukso-network/lsp-smart-contracts/blob/b3ff7a66439cbfed779e3ee0992d5b9047ad208e/contracts/LSP6KeyManager/

LSP6KeyManagerCore.sol#L166-L171

```
166  function _isValidNonce(address from, uint256 idx) internal view returns (bool) {
167      // idx % (1 << 128) = nonce
168      // (idx >> 128) = channel
169      // equivalent to: return (nonce == _nonceStore[_from][channel]
170      return (idx % (1 << 128)) == (_nonceStore[from][idx >> 128]);
171  }
```

”idx” is a 256bits (unsigned) integer, where:
- the 128 leftmost bits = channelId.
- the 128 rightmost bits = nonce within the channel

[WP-I15] Consider moving the mock contracts to a separate folder to differentiate them from the production-ready library contracts under the `Helpers` folder

Informational

Issue Description

There are quite a few contracts under the `Helpers` folder, some of which seems like mock contracts that should only be used for tests.

Eg, `UPWithInstantAcceptOwnership` looks like a test code for one of `universalReceiver()` 's use cases:

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/Helpers/UPWithInstantAcceptOwnership.sol

```
1  // SPDX-License-Identifier: Apache-2.0
2  pragma solidity ^0.8.0;
3
4  // modules
5  import {LSP0ERC725AccountCore} from
   "../LSP0ERC725Account/LSP0ERC725AccountCore.sol";
6  import {OwnableUnset} from
   "@erc725/smart-contracts/contracts/custom/OwnableUnset.sol";
7  import {LSP14Ownable2Step} from "../LSP14Ownable2Step/LSP14Ownable2Step.sol";
8
9  // constants
10 import "../LSP14Ownable2Step/LSP14Constants.sol";
11
12 contract UPWithInstantAcceptOwnership is LSP0ERC725AccountCore {
13     /**
14      * @notice Sets the owner of the contract
15      * @param newOwner the owner of the contract
16      */
17     constructor(address newOwner) payable {
18         OwnableUnset._setOwner(newOwner);
19     }
```


```
20
21      function universalReceiver(bytes32 typeId, bytes calldata receivedData)
22          public
23          payable
24          virtual
25          override
26          returns (bytes memory returnedValue)
27      {
28          if (typeId == _TYPEID_LSP14_OwnershipTransferStarted) {
29              LSP14Ownable2Step(msg.sender).acceptOwnership();
30          }
31          super.universalReceiver(typeId, receivedData);
32      }
33  }
```

But we only learned that when we saw the test code related to `LSP14Ownable2Step._transferOwnership()` :

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP14Ownable2Step/LSP14Ownable2Step.sol#L100-L112

```
100     function _transferOwnership(address newOwner) internal virtual {
101         if (newOwner == address(this)) revert CannotTransferOwnershipToSelf();
102         _pendingOwner = newOwner;
103
104         address currentOwner = owner();
105         _notifyUniversalReceiver(newOwner, _TYPEID_LSP14_OwnershipTransferStarted,
    "");
106         require(
107             currentOwner == owner(),
108             "LSP14: newOwner MUST accept ownership in a separate transaction"
109         );
110
111         emit OwnershipTransferStarted(currentOwner, newOwner);
112     }
```

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/tests/LSP14Ownable2Step/LSP14Ownable2Step.

behaviour.ts#L137-L155

```
137     describe("when `acceptOwnership(...)` is called in the same tx as
    `transferOwnership(...)`", () => {
138       let upWithCustomURD: UPWithInstantAcceptOwnership;
139       before(async () => {
140         context = await buildContext();
141         upWithCustomURD = await new UPWithInstantAcceptOwnership__factory(
142           context.accounts[0]
143         ).deploy(context.accounts[0].address);
144       });
145
146       it("should revert (e.g: if `universalReceiver(...)` function of `newOwner`
    calls directly `acceptOwnership(...)')", async () => {
147         const ownershipTransfer = context.contract
148           .connect(context.deployParams.owner)
149           .transferOwnership(upWithCustomURD.address);
150
151         await expect(ownershipTransfer).to.be.revertedWith(
152           "LSP14: newOwner MUST accept ownership in a separate transaction"
153         );
154       });
155     });
```

Recommendation

It's important to clearly identify the test/mock codes from the production-ready codes.

More particularly, consider creating a new folder called test/mock and moving them there.

Also, consider adding a few lines of comment on the top of the test/mock contracts to warn the readers about the danger to use these codes directly on production.

[WP-I16] LSP-0: Consider implementing the `ERC721TokenReceiver` interface to accept ERC721 safe transfers

Informational

Issue Description

```
/// @dev Note: the ERC-165 identifier for this interface is 0x150b7a02.
interface ERC721TokenReceiver {
    /// @notice Handle the receipt of an NFT
    /// @dev The ERC721 smart contract calls this function on the recipient
    ///  after a `transfer`. This function MAY throw to revert and reject the
    ///  transfer. Return of other than the magic value MUST result in the
    ///  transaction being reverted.
    ///  Note: the contract address is always the message sender.
    /// @param _operator The address which called `safeTransferFrom` function
    /// @param _from The address which previously owned the token
    /// @param _tokenId The NFT identifier which is being transferred
    /// @param _data Additional data with no specified format
    /// @return
`bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
    ///  unless throwing
    function onERC721Received(address _operator, address _from, uint256 _tokenId,
bytes _data) external returns(bytes4);
}
```

Ref: https://eips.ethereum.org/EIPS/eip-721#:~:text=A%20wallet/broker/auction%20application%20MUST%20implement%20the%20wallet%20interface%20if%20it%20will%20accept%20safe%20transfers.

Given that a considerable amount of protocols are using `safeTransferFrom` to transfer NFTs (erc721)

As a smart contract based wallet, `LSP-0-ERC725Account` should be able to support receiving various assets as much as possible.

However, in the current implementation, because `onERC721Received()` is not implemented on `LSP-0-ERC725Account`, the `safeTransferFrom` can not be done.

See:

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/a1948250ab8c441f6d327a65754cb20d2b1b4554/contracts/token/ERC721/ERC721.sol#L429-L451

```
429  function _checkOnERC721Received(
430      address from,
431      address to,
432      uint256 tokenId,
433      bytes memory data
434  ) private returns (bool) {
435      if (to.isContract()) {
436          try IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId,
     data) returns (bytes4 retval) {
437              return retval == IERC721Receiver.onERC721Received.selector;
438          } catch (bytes memory reason) {
439              if (reason.length == 0) {
440                  revert("ERC721: transfer to non ERC721Receiver implementer");
441              } else {
442                  /// @solidity memory-safe-assembly
443                  assembly {
444                      revert(add(32, reason), mload(reason))
445                  }
446              }
447          }
448      } else {
449          return true;
450      }
451  }
```

For reference, Gnosis Safe implemented the support for most of the common token callbacks:

https://github.com/safe-global/safe-contracts/blob/main/contracts/handler/DefaultCallbackHandler.sol

```
11  contract DefaultCallbackHandler is ERC1155TokenReceiver, ERC777TokensRecipient,
    ERC721TokenReceiver, IERC165 {
12      string public constant NAME = "Default Callback Handler";
13      string public constant VERSION = "1.0.0";
14
15      function onERC1155Received(
```

```
16          address,
17          address,
18          uint256,
19          uint256,
20          bytes calldata
21      ) external pure override returns (bytes4) {
22          return 0xf23a6e61;
23      }
24
25      function onERC1155BatchReceived(
26          address,
27          address,
28          uint256[] calldata,
29          uint256[] calldata,
30          bytes calldata
31      ) external pure override returns (bytes4) {
32          return 0xbc197c81;
33      }
34
35      function onERC721Received(
36          address,
37          address,
38          uint256,
39          bytes calldata
40      ) external pure override returns (bytes4) {
41          return 0x150b7a02;
42      }
43
44      function tokensReceived(
45          address,
46          address,
47          address,
48          uint256,
49          bytes calldata,
50          bytes calldata
51      ) external pure override {
52          // We implement this for completeness, doesn't really have any value
53      }
54
55      function supportsInterface(bytes4 interfaceId) external view virtual override
    returns (bool) {
56          return
57              interfaceId == type(ERC1155TokenReceiver).interfaceId ||
```

```
58              interfaceId == type(ERC721TokenReceiver).interfaceId ||
59              interfaceId == type(IERC165).interfaceId;
60      }
61  }
```

[WP-I17] LSP-8: `LSP8CompatibleERC721` is not compatible with `ERC721TokenReceiver`

Informational

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP8IdentifiableDigitalAsset/extensions/LSP8CompatibleERC721.sol#L176-L183

```
176  function safeTransferFrom(
177      address from,
178      address to,
179      uint256 tokenId,
180      bytes memory data
181  ) public virtual {
182      return _transfer(from, to, bytes32(tokenId), false, data);
183  }
```

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP8IdentifiableDigitalAsset/LSP8IdentifiableDigitalAssetCore.sol#L335-L369

```
335  function _transfer(
336      address from,
337      address to,
338      bytes32 tokenId,
339      bool force,
340      bytes memory data
341  ) internal virtual {
342      if (from == to) {
343          revert LSP8CannotSendToSelf();
344      }
345
346      address tokenOwner = tokenOwnerOf(tokenId);
347      if (tokenOwner != from) {
348          revert LSP8NotTokenOwner(tokenOwner, tokenId, from);
349      }
```

```
350
351     if (to == address(0)) {
352         revert LSP8CannotSendToAddressZero();
353     }
354
355     address operator = msg.sender;
356
357     _beforeTokenTransfer(from, to, tokenId);
358
359     _clearOperators(from, tokenId);
360
361     _ownedTokens[from].remove(tokenId);
362     _ownedTokens[to].add(tokenId);
363     _tokenOwners[tokenId] = to;
364
365     emit Transfer(operator, from, to, tokenId, force, data);
366
367     _notifyTokenSender(from, to, tokenId, data);
368     _notifyTokenReceiver(from, to, tokenId, force, data);
369 }
```

https://github.com/lukso-network/lsp-smart-contracts/blob/d6e1185ba5b7b8ac7cc8bad2f8c832abec2c4a89/contracts/LSP8IdentifiableDigitalAsset/LSP8IdentifiableDigitalAssetCore.sol#L420-L437

```
420 function _notifyTokenReceiver(
421     address from,
422     address to,
423     bytes32 tokenId,
424     bool force,
425     bytes memory data
426 ) internal virtual {
427     if (ERC165Checker.supportsERC165Interface(to, _INTERFACEID_LSP1)) {
428         bytes memory packedData = abi.encodePacked(from, to, tokenId, data);
429         ILSP1UniversalReceiver(to).universalReceiver(_TYPEID_LSP8_TOKENSRECIPIENT,
    packedData);
430     } else if (!force) {
431         if (to.code.length != 0) {
432             revert LSP8NotifyTokenReceiverContractMissingLSP1Interface(to);
433         } else {
434             revert LSP8NotifyTokenReceiverIsEOA(to);
```

```
435             }
436         }
437     }
```

[WP-I18] LSP-7: `LSP7CompatibleERC20` Compatibility to ERC20 regading the unexpected hooks

Informational

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/eb9919fb82414657b721cfb6e70f06659366af52/contracts/LSP7DigitalAsset/extensions/LSP7CompatibleERC20.sol

The current implementation of `LSP7CompatibleERC20` comes with unexpected hooks which can be used for reentrancy during `_transfer()` .

Tokens with such features are infamous for the exploits casued by the hooks, most noteably:

- Uniswap imBTC pool hack (04/18/2020)
- Cream Finance hack (08/30/2021)
- Ola and Voltage Lending hack (3/31/2022)
- Agave and Hundred Finance hack (03/15/2022)

Recommendation

1. Consider renaming to `LSP7ERC777Compatible` to indicate that it's actually a token with transfer hooks.
2. Consider removing the hooks in `LSP7ERC20Compatible` to avoid the hooks related security issues (mostly caused by the reentrancy introduced with the hooks).

[WP-D19] `authorizeOperator()` CAN NOT avoid front-running and Allowance Double-Spend Exploit

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/b3ff7a66439cbfed779e3ee0992d5b9047ad208e/contracts/LSP7DigitalAsset/LSP7DigitalAssetCore.sol#L71-L86

```
71      /**
72       * @inheritdoc ILSP7DigitalAsset
73       *
74       * @dev To avoid front-running and Allowance Double-Spend Exploit when
75       * increasing or decreasing the authorized amount of an operator,
76       * it is advised to:
77       *     1. call {revokeOperator} first, and
78       *     2. then re-call {authorizeOperator} with the new amount
79       *
80       * for more information, see:
81       *
    https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/
82       *
83       */
84      function authorizeOperator(address operator, uint256 amount) public virtual {
85          _updateOperator(msg.sender, operator, amount);
86      }
```

The current implementation and suggested steps to avoid the Allowance Double-Spend Exploit won't be practical.

Take the attack senerio from the original issue document for example:

1. Alice allows Bob to transfer N of Alice's tokens (N>0) by calling the approve method on a Token smart contract, passing the Bob's address and N as the method arguments
2. After some time, Alice decides to change from N to M (M>0) the number of Alice's tokens Bob is allowed to transfer, so she calls the approve method again, this time passing the Bob's address and M as the method arguments
3. Bob notices the Alice's second transaction before it was mined and quickly sends another

transaction that calls the transferFrom method to transfer N Alice's tokens somewhere
4. If the Bob's transaction will be executed before the Alice's transaction, then Bob will successfully transfer N Alice's tokens and will gain an ability to transfer another M tokens
5. Before Alice noticed that something went wrong, Bob calls the transferFrom method again, this time to transfer M Alice's tokens.

Even if Alice:

1. call `revokeOperator` first, and
2. then re-call `authorizeOperator` with the new amount

Unless Alice waits for the first `revokeOperator` transaction to be minted and observes whether Bob has frontrun and spent all the pre-existing allowance, then decides whether to send the second transaction to approve a new amount, which is quite impractical, because the end user won't be this prudent.

Bob can still frontrun the `revokeOperator()` call and spend all the `N`, and backrun `authorizeOperator()` to spend all the `M`.

Recommendation

We can see 3 possible solutions for this issue:

A: Consdier adding 2 new methods:

- `increaseAllowance(address operator, uint256 addedValue)`
- `decreaseAllowance(address operator, uint256 maxSubtractedValue)`

B: Consider adopting the Atomic "Compare And Set" Approve Method suggested on https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit#

C: Consider removing the comments at L74-81 given that the impact of this issue itself is negligible.

[WP-D20] LIPs / LSP-2: Typos and inconsistency

Issue Description

https://github.com/lukso-network/LIPs/blob/1b99cce54963f7d8406a38c0d01bd119dec94324/LSPs/LSP-2-ERC725YJSONSchema.md?plain=1#L115

```
107  | `valueType` | Description |
108  |---|---|
109  | `boolean`     | a value as either **true** or **false** |
110  | `string`      | an UTF8 encoded string  |
111  | `address`     | a 20 bytes long address |
112  | `uintN`       | an **unsigned** integer (= only positive number) of size `N`  |
113  | `bytesN`      | a bytes value of **fixed-size** `N`, from `bytes1` up to
     `bytes32` |
114  | `bytes`       | a bytes value of **dynamic-size** |
115  | `uintN[]`     | an array of **signed** integers |
116  | `string[]`    | an array of UTF8 encoded strings |
117  | `address[]`   | an array of addresses |
118  | `bytes[]`     | an array of dynamic size bytes  |
119  | `bytesN[]`    | an array of fixed size bytes  |
```

”an array of **signed** integers” -> ”an array of **unsigned** integers”:

```
1  diff --git a/LSPs/LSP-2-ERC725YJSONSchema.md b/LSPs/LSP-2-ERC725YJSONSchema.md
2  index a067a19..aa15032 100644
3  --- a/LSPs/LSP-2-ERC725YJSONSchema.md
4  +++ b/LSPs/LSP-2-ERC725YJSONSchema.md
5  @@ -112,7 +112,7 @@ The `valueType` can also be useful for typecasting. It enables
   contracts or inte
6    | `uintN`       | an **unsigned** integer (= only positive number) of size `N`  |
7    | `bytesN`      | a bytes value of **fixed-size** `N`, from `bytes1` up to
     `bytes32` |
8    | `bytes`       | a bytes value of **dynamic-size** |
9  -| `uintN[]`     | an array of **signed** integers |
10 +| `uintN[]`     | an array of **unsigned** integers |
11   | `string[]`    | an array of UTF8 encoded strings |
12   | `address[]`   | an array of addresses |
13   | `bytes[]`     | an array of dynamic size bytes  |
```

Whether a Number is positive or negative is depending on the `valueType` instead of the `keyType`

https://github.com/lukso-network/LIPs/blob/1b99cce54963f7d8406a38c0d01bd119dec94324/LSPs/LSP-2-ERC725YJSONSchema.md?plain=1#L138

```
131  Valid `valueContent` are:
132
133  | `valueContent`    | Description  |
134  |---|---|
135  | `Boolean`         | a boolean |
136  | `String`          | an UTF8 encoded string |
137  | `Address`         | an address |
138  | `Number`          | a Number (positive or negative, depending on the `keyType`)
     |
139  | `BytesN`          | a bytes value of **fixed-size** `N`, from `bytes1` up to
     `bytes32`  |
140  | `Bytes`           | a bytes value of **dynamic-size** |
141  | `Keccak256`       | a 32 bytes long hash digest, obtained from the keccak256
     hashing algorithm |
142  | `BitArray`        | an array of single `1` or `0` bits |
143  | `URL`             | an URL encoded as an UTF8 string |
144  | [`AssetURL`](#asseturl)   | The content contains the hash function, hash and
     link to the asset file  |
145  | [`JSONURL`](#jsonurl)     |  hash function, hash and link to the JSON file |
146  | `Markdown`        | a structured Markdown mostly encoded as UTF8 string  |
147  | `0x1345ABCD...`   | a **literal** value, when the returned value is expected to
     equal some specific bytes |
```

”a Number (positive or negative, depending on the **keyType**)” -> ”a Number (positive or negative, depending on the valueType)”

https://github.com/lukso-network/LIPs/blob/e26a72363ccd6e6a00443a0eecd1da9131202b17/LSPs/LSP-2-ERC725YJSONSchema.md#L54-L63

```
54  The table below describes each entries with their available options.
55
56  | Title                                  | Description                    |
57  |:---------------------------------------|:-------------------------------|
58  |[`name`](#name)                         | the name of the data key       |
59  |[`key`](#key)                           | the **unique identifier** of the data
    key |
60  |[`keyType`](#keyType)                   | *How* the data key must be treated <hr>
    [`Singleton`](#Singleton) <br> [`Array`](#Array) <br> [`Mapping`](#mapping) <br>
    [`MappingWithGrouping`](#mappingwithgrouping) |
61  |[`valueType`](#valueType)               | *How* a value MUST be decoded <hr>
    `boolean` <br> `string` <br> `address` <br> `uintN` <br> `intN` <br> `bytesN` <br>
    `bytes` <br> `uintN[]` <br> `intN[]` <br> `string[]` <br> `address[]` <br>
    `bytes[]` |
62  |[`valueContent`](#valueContent)| *How* a value SHOULD be interpreted <hr>
    `Boolean` <br> `String` <br> `Address` <br> `Number` <br> `BytesN` <br> `Bytes`
    <br> `Keccak256` <br> [`BitArray`](#BitArray) <br> `URL` <br>
    [`AssetURL`](#AssetURL) <br> [`JSONURL`](#JSONURL) <br> `Markdown` <br> `Literal`
    (*e.g.:* `0x1345ABCD...`) |
63
```

The `valueType` in the table above does not align with the `valueType` listed below:

https://github.com/lukso-network/LIPs/blob/e26a72363ccd6e6a00443a0eecd1da9131202b17/LSPs/LSP-2-ERC725YJSONSchema.md#L105-L121

```
105 The `valueType` can also be useful for typecasting. It enables contracts or
    interfaces to know how to manipulate the data and the limitations behind its type.
    To illustrate, an interface could know that it cannot set the value to `300` if
    its `valueType` is `uint8` (max `uint8` allowed = `255`).
106
107 | `valueType` | Description |
108 |---|---|
109 | `boolean`     | a value as either **true** or **false** |
110 | `string`      | an UTF8 encoded string  |
111 | `address`     | a 20 bytes long address |
112 | `uintN`       | an **unsigned** integer (= only positive number) of size `N`  |
113 | `bytesN`      | a bytes value of **fixed-size** `N`, from `bytes1` up to
    `bytes32` |
114 | `bytes`       | a bytes value of **dynamic-size** |
115 | `uintN[]`     | an array of **signed** integers |
```

```
116 | `string[]`    | an array of UTF8 encoded strings |
117 | `address[]`   | an array of addresses |
118 | `bytes[]`     | an array of dynamic size bytes  |
119 | `bytesN[]`    | an array of fixed size bytes  |
120
121 The `valueType` can also be a **tuple of types**, meaning the value stored under
    the ERC725Y data key is a mixture of multiple value types concatenated together.
    In such case, the types MUST be defined between parentheses. For instance:
    `(bytes4,bytes8)` or `(bytes8,address)`
```

[WP-I21] LSP-1: `UniversalReceiverDelegateVaultSetter#universalReceiverDelegate()` is a public method with no access control, anyone can call this to set arbitrary data to the target ERC725Y addresses

Informational

Issue Description

Based on the context, it seems like `UniversalReceiverDelegateVaultSetter.sol` is a very realistic-looking demo code.

However, given the fact that:

1. it works fine;
2. it looks real.

Some reckless devs might just take the code and deploy on production.

https://github.com/lukso-network/lsp-smart-contracts/blob/b3ff7a66439cbfed779e3ee0992d5b9047ad208e/contracts/Helpers/UniversalReceivers/UniversalReceiverDelegateVaultSetter.sol#L21-L32

```
21  function universalReceiverDelegate(
22      address vaultadd,
23      bytes32 key,
24      bytes memory value
25  ) external {
26      bytes32[] memory keys = new bytes32[](1);
27      bytes[] memory values = new bytes[](1);
28
29      keys[0] = key;
30      values[0] = value;
31      IERC725Y(vaultadd).setData(keys, values);
32  }
```

Recommendation

See the [Recommendation] on [WP-I15].

[WP-I22] LSP-6: `AddressPermissions:AllowedStandards:<address>` is error-prone for unexpected method calls outside the allowed standard

Informational

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/b3ff7a66439cbfed779e3ee0992d5b9047ad208e/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L590-L613

```
590      /**
591       * @dev if `from` is restricted to interact with contracts that implement a specific interface,
592       * verify that `to` implements one of these interface.
593       * @param from the caller address
594       * @param to the address of the contract to interact with
595       */
596      function _verifyAllowedStandard(address from, address to) internal view {
597          bytes memory allowedStandards = ERC725Y(target).getAllowedStandardsFor(from);
598
599          // whitelist any standard interface (ERC165)
600          if (
601              // if nothing in the list
602              allowedStandards.length == 0 ||
603              // if not correctly abi-encoded array of bytes4[]
604              !LSP2Utils.isBytes4EncodedArray(allowedStandards)
605          ) return;
606
607          bytes4[] memory allowedStandardsList = abi.decode(allowedStandards, (bytes4[]));
608
609          for (uint256 ii = 0; ii < allowedStandardsList.length; ii = GasLib.uncheckedIncrement(ii)) {
610              if (to.supportsERC165Interface(allowedStandardsList[ii])) return;
611          }
612          revert NotAllowedStandard(from, to);
613      }
```

PoC

Alice gives Bob permission to manage her NFTs by adding `ERC721InterfaceId` to `AddressPermissions:AllowedStandards:Bob`.

Alice expected Bob only to be able to call the NFT functions (eg, `trasnfer`) for her JPEGs.

But actually, Bob can now call ANY address at ANY method, as long as the address supportsInterface(ERC721).

For example, Bob can now call Uniswap v3's NonfungiblePositionManager to `mint`, `burn`, `increaseLiquidity`, `decreaseLiquidity` beacuse Uniswap v3's NonfungiblePositionManager.supportsInterface(ERC721InterfaceId) == true

See:

- https://docs.uniswap.org/protocol/reference/periphery/NonfungiblePositionManager
- 0xC36442b4a4522E871399CD717aBDD847Ab11FE88

[WP-I23] LSP-8: Consider using `bool[]` for `force` parameter in `transferBatch()`

Informational

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/b3ff7a66439cbfed779e3ee0992d5b9047ad208e/contracts/LSP8IdentifiableDigitalAsset/LSP8IdentifiableDigitalAssetCore.sol#L200-L216

```
200  function transferBatch(
201      address[] memory from,
202      address[] memory to,
203      bytes32[] memory tokenId,
204      bool force,
205      bytes[] memory data
206  ) public virtual {
207      if (
208          from.length != to.length || from.length != tokenId.length || from.length
     != data.length
209      ) {
210          revert LSP8InvalidTransferBatch();
211      }
212
213      for (uint256 i = 0; i < from.length; i = GasLib.uncheckedIncrement(i)) {
214          transfer(from[i], to[i], tokenId[i], force, data[i]);
215      }
216  }
```

The current implementation requires all the transfers within the batch to use the same `force`, while all the other parameters can be specified for each transfer.

Recommendation

Consider using `bool[]` for `force` parameter as well for consistency and better usability.

```
200  function transferBatch(
201      address[] memory from,
202      address[] memory to,
203      bytes32[] memory tokenId,
204      bool[] memory force,
205      bytes[] memory data
206  ) public virtual {
207      if (
208          from.length != to.length || from.length != tokenId.length || from.length
     != data.length || from.length != force.length
209      ) {
210          revert LSP8InvalidTransferBatch();
211      }
212
213      for (uint256 i = 0; i < from.length; i = GasLib.uncheckedIncrement(i)) {
214          transfer(from[i], to[i], tokenId[i], force[i], data[i]);
215      }
216  }
```

[WP-G24] LSP-6: Recalculate the same `mask` in the for loop is unnecessary

Gas

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/b3ff7a66439cbfed779e3ee0992d5b9047ad208e/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L420-L469

```
420  function _verifyAllowedERC725YKeys(address from, bytes32[] memory inputKeys)
     internal view {
421      bytes memory allowedERC725YKeysEncoded =
     ERC725Y(target).getAllowedERC725YKeysFor(from);
422
423      // whitelist any ERC725Y key
424      if (
425          // if nothing in the list
426          allowedERC725YKeysEncoded.length == 0 ||
427          // if not correctly abi-encoded array
428          !LSP2Utils.isEncodedArray(allowedERC725YKeysEncoded)
429      ) return;
430
431      bytes32[] memory allowedERC725YKeys = abi.decode(allowedERC725YKeysEncoded,
     (bytes32[]));
432
433      uint256 zeroBytesCount;
434      bytes32 mask;
435
436      // loop through each allowed ERC725Y key retrieved from storage
437      for (uint256 ii = 0; ii < allowedERC725YKeys.length; ii =
     GasLib.uncheckedIncrement(ii)) {
438          // required to know which part of the input key to compare against the
     allowed key
439          zeroBytesCount = _countTrailingZeroBytes(allowedERC725YKeys[ii]);
440
441          // loop through each keys given as input
442          for (uint256 jj = 0; jj < inputKeys.length; jj =
     GasLib.uncheckedIncrement(jj)) {
443              // skip permissions keys that have been previously checked and
     "nulled"
```

```
444             if (inputKeys[jj] == bytes32(0)) continue;
445
446             // use a bitmask to discard the last `n` bytes of the input key (where
    `n` = `zeroBytesCount`)
447             // and compare only the relevant parts of each ERC725Y keys
448             //
449             // for an allowed key =
    0xcafecafecafecafecafecafecafecafe00000000000000000000000000000000
450             //
451             //                         |------compare this
    part-------|------discard this part--------|
452             //                         v                               v
    v
453             //               mask =
    0xffffffffffffffffffffffffffffffff00000000000000000000000000000000
454             //        & input key =
    0xcafecafecafecafecafecafecafecafe00000000000000000000000011223344
455             //
456             mask = bytes32(type(uint256).max) << (8 * zeroBytesCount);
457
458             if (allowedERC725YKeys[ii] == (inputKeys[jj] & mask)) {
459                 // if the input key matches the allowed key
460                 // make it null to mark it as allowed
461                 inputKeys[jj] = bytes32(0);
462             }
463         }
464     }
465
466     for (uint256 ii = 0; ii < inputKeys.length; ii =
    GasLib.uncheckedIncrement(ii)) {
467         if (inputKeys[ii] != bytes32(0)) revert NotAllowedERC725YKey(from,
    inputKeys[ii]);
468     }
469 }
```

Recommendation

```
420 function _verifyAllowedERC725YKeys(address from, bytes32[] memory inputKeys)
    internal view {
421     bytes memory allowedERC725YKeysEncoded =
    ERC725Y(target).getAllowedERC725YKeysFor(from);
```

```
422
423     // whitelist any ERC725Y key
424     if (
425         // if nothing in the list
426         allowedERC725YKeysEncoded.length == 0 ||
427         // if not correctly abi-encoded array
428         !LSP2Utils.isEncodedArray(allowedERC725YKeysEncoded)
429     ) return;
430
431     bytes32[] memory allowedERC725YKeys = abi.decode(allowedERC725YKeysEncoded,
    (bytes32[]));
432
433     uint256 zeroBytesCount;
434     bytes32 mask;
435
436     // loop through each allowed ERC725Y key retrieved from storage
437     for (uint256 ii = 0; ii < allowedERC725YKeys.length; ii =
    GasLib.uncheckedIncrement(ii)) {
438         // required to know which part of the input key to compare against the
    allowed key
439         zeroBytesCount = _countTrailingZeroBytes(allowedERC725YKeys[ii]);
440         mask = bytes32(type(uint256).max) << (8 * zeroBytesCount);
441         // loop through each keys given as input
442         for (uint256 jj = 0; jj < inputKeys.length; jj =
    GasLib.uncheckedIncrement(jj)) {
443             // skip permissions keys that have been previously checked and
    "nulled"
444             if (inputKeys[jj] == bytes32(0)) continue;
445
446             // use a bitmask to discard the last `n` bytes of the input key (where
    `n` = `zeroBytesCount`)
447             // and compare only the relevant parts of each ERC725Y keys
448             //
449             // for an allowed key =
    0xcafecafecafecafecafecafecafecafecafecafe000000000000000000000000000000
450             //
451             //                         |------compare this
    part-------|------discard this part--------|
452             //                         v                                 v
    v
453             //               mask =
    0xffffffffffffffffffffffffffffffffffffffff000000000000000000000000000000
454             //        & input key =
    0xcafecafecafecafecafecafecafecafecafecafe000000000000000000000011223344
```


```
455             //
456
457             if (allowedERC725YKeys[ii] == (inputKeys[jj] & mask)) {
458                 // if the input key matches the allowed key
459                 // make it null to mark it as allowed
460                 inputKeys[jj] = bytes32(0);
461             }
462         }
463     }
464     for (uint256 ii = 0; ii < inputKeys.length; ii =
    GasLib.uncheckedIncrement(ii)) {
465         if (inputKeys[ii] != bytes32(0)) revert NotAllowedERC725YKey(from,
    inputKeys[ii]);
466     }
467 }
```

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.