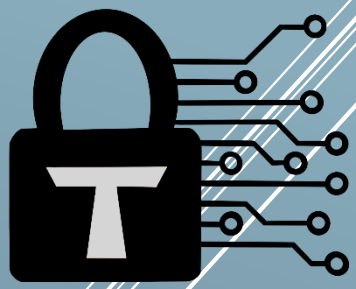
Trust Security

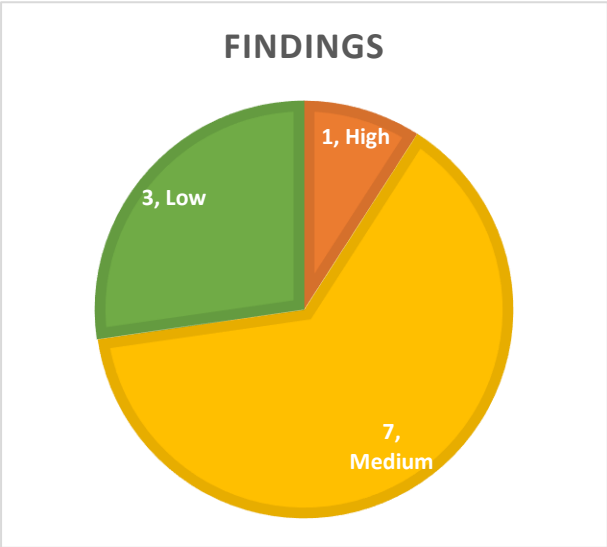


Smart Contract Audit

LUKSO LSP audit

13/04/2023

Executive summary



Category	Core Protocols
Audited file count	52
Lines of Code	2663
Auditor	Trust
Time period	25/03-13/04

Findings

Severity	Total	Fixed	Acknowledged
High	1	1	-
Medium	7	5	2
Low	3	2	1

Centralization score

Signature

Digitally signed by Trust
Date: 2023-05-21 14:37:45

Document properties

Versioning

Version	Date	Description
0.1	13/04/2023	Client report
0.2	21/05/2023	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

lsp-smart-contracts repo:

- contracts/LSP0ERC725Account/*
- contracts/LSP1UniversalReceiver/*
- contracts/LSP2ERC725YJSONSchema/*
- contracts/LSP5ReceivedAssets/*
- contracts/LSP6KeyManager/*
- contracts/LSP10ReceivedVaults/*
- contracts/LSP14Ownable2Step/*
- contracts/LSP20CallVerification/*

ERC725 repo:

- contracts/*
- contracts/interfaces/*
- contracts/custom/OwnableUnset.sol

Repository details

- **Repository URL:** https://github.com/ERC725Alliance/ERC725/
- **Commit tag:** v4.2.0
- **Repository URL:** https://github.com/lukso-network/lsp-smart-contracts
- **Commit hash:** v0.9.0

Mitigation review – Pull Requests

- **H-1:** https://github.com/lukso-network/lsp-smart-contracts/pull/576
- **M-1:** https://github.com/lukso-network/lsp-smart-contracts/pull/573
- **M-2:** https://github.com/lukso-network/LIPs/pull/206
- **M-4:** https://github.com/lukso-network/LIPs/pull/207
- **M-5:** https://github.com/lukso-network/lsp-smart-contracts/pull/562

- **M-6:** https://github.com/lukso-network/lsp-smart-contracts/pull/563
- **M-7:** https://github.com/lukso-network/lsp-smart-contracts/pull/577
- **L-1:** https://github.com/lukso-network/LIPs/pull/201
- **L-2:** https://github.com/lukso-network/lsp-smart-contracts/pull/582

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Good**	Project succeeded in simplifying code flows, reducing attack risks.
Documentation	**Excellent**	Project is very well documented.
Best practices	**Excellent**	Project consistently adheres to industry standards.
Centralization risks	**Excellent**	Project does not introduce any unnecessary centralization risks. Users can opt-in to a desired amount of centralization.

Findings

High severity findings

TRST-H-1 Reentrancy protection can likely be bypassed

- **Category:** Reentrancy issues
- **Source:** LSP6KeyManagerCore.sol
- **Status:** Fixed

Description

The KeyManager offers reentrancy protection for interactions with the associated account. Through the LSP20 callbacks or through the *execute()* calls, it will call *_nonReentrantBefore()* before execution, and *_nonReentrantAfter()* post-execution. The latter will always reset the flag signaling entry.

```
function _nonReentrantAfter() internal virtual {
    // By storing the original value once again, a refund is
triggered (see
    // https://eips.ethereum.org/EIPS/eip-2200)
    _reentrancyStatus = false;
}
```

An attacker can abuse it to reenter provided that there exists some third-party contract with REENTRANCY_PERMISSION that performs some interaction with the contract. The attacker would trigger the third-party code path, which will clear the reentrancy status, and enable attacker to reenter. This could potentially be chained several times. Breaking the reentrancy assumption would make code that assumes such flows to be impossible to now be vulnerable.

Recommended mitigation

In *_nonReentrantAfter()*, the flag should be returned to the original value before reentry, rather than always setting it to false.

Team response

Applied a fix different than recommendation.

Mitigiation review

All code paths will now leave the **_reentrancyStatus** on when the current call is not the initial call to the KeyManager.

Medium severity findings

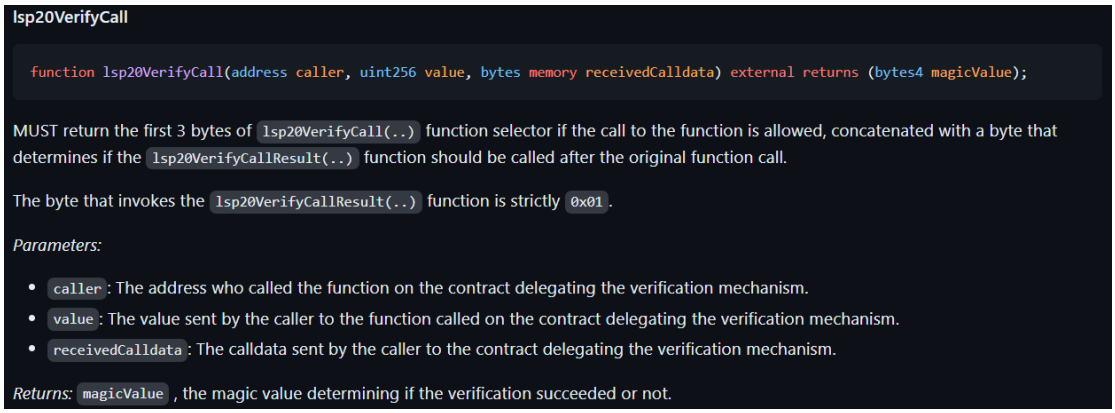
TRST-M-1 LSP20 verification library deviates from spec and will accept fail values

- **Category:** Specification errors
- **Source:** LSP20CallVerification.sol

- **Status:** Fixed

Description

The functions *lsp20VerifyCall()* and *lsp20VerifyCallResult()* are called to validate the owner accepts some account interaction. The specification states they must return a specific 4 byte magic value.



However, the implementation will accept any byte array that starts with the required magic value.

```
function _verifyCall(address logicVerifier) internal virtual returns
(bool verifyAfter) {
    (bool success, bytes memory returnedData) = logicVerifier.call(
        abi.encodeWithSelector(ILSP20.lsp20VerifyCall.selector,
msg.sender, msg.value, msg.data)
    );
    if (!success) _revert(false, returnedData);
    if (returnedData.length < 32) revert
LSP20InvalidMagicValue(false, returnedData);
    bytes32 magicValue = abi.decode(returnedData, (bytes32));
    if (bytes3(magicValue) !=
bytes3(ILSP20.lsp20VerifyCall.selector))
        revert LSP20InvalidMagicValue(false, returnedData);
    return bytes1(magicValue[3]) == 0x01 ? true : false;
}
```

Therefore, implementations of the above functions which intend to signal failure status may be accepted by the verification wrapper above.

Recommended mitigation

Verify that the return data length is 32 bytes (the 4 bytes are extended by the compiler), and that all other bytes are zero.

Team response

Fixed (applied recommendation).

Mitigation review

The code correctly validates only the first four bytes are not zero. Code will still accept data length which is greater than 32 bytes, which has been confirmed as a design decision.

TRST-M-2 Deviation from spec will result in dislocation of receiver delegate

- **Category:** Specification errors
- **Source:** LSP0ERC725AccountCore.sol
- **Status:** Fixed

Description

The LSP0 *universalReceiver()* function looks up the receiver delegate by crafting a mapping key type.

```
bytes32 lsp1typeIdDelegateKey = LSP2Utils.generateMappingKey(
    _LSP1_UNIVERSAL_RECEIVER_DELEGATE_PREFIX,
    bytes20(typeId)
);
```

Mapping keys are constructed of a 10-byte prefix, 2 zero bytes and a 20-byte suffix. However, followers of the specification will use an incorrect suffix.

The docs do not discuss the trimming of **bytes32** into a **bytes20** type. The mismatch may cause various harmful scenarios when interacting with the delegate not using the reference implementation.

Recommended mitigation

Document the trimming action in the LSP0 specification.

Team response

Fix applied (documented in specs).

Mitigation review

Docs clearly state the described behavior.

TRST-M-3 LSP0 allows delegate calls when LSP20 post-execution check is required

- **Category:** Logical flaws
- **Source:** LSP0ERC725AccountCore.sol
- **Status:** Acknowledged

Description

LSP20 call verification protects actions on an LSP0 account. The pre-execution check may mark the action as requiring a post-execution check. However, in the case of **delegate call**

execution, the post-execution check can never be guaranteed to execute. The called contract may call the SELFDESTRUCT opcode and destroy the calling contract. Call flow returns immediately from the call to *execute()*, without executing the necessary check.

Recommended mitigation

When the first call verification marks a secondary call as necessary, disallow **delegate call** execution. The fix should be applied to both *execute()* variants.

Team response

The use of **delegatecall** with **selfdestruct** will result in the destruction of the contract, in this case, the post-execution check will not run. However, **delegatecall** could be used in other scenarios requiring a post-execution check. We decided not to remove the post-execution check because of only one edge case.

TRST-M-4 Incorrect decoding of universalReceiver() call

- **Category:** Specification errors
- **Source:** LSP1Utils.sol
- **Status:** Acknowledged

Description

Some contracts use the utility below to interact with the registered universal receiver delegate.

```
function callUniversalReceiverWithCallerInfos(
    address universalReceiverDelegate,
    bytes32 typeId,
    bytes calldata receivedData,
    address msgSender,
    uint256 msgValue
) internal returns (bytes memory) {
    bytes memory callData = abi.encodePacked(
        abi.encodeWithSelector(
            ILSP1UniversalReceiver.universalReceiver.selector,
            typeId,
            receivedData
        ),
        msgSender,
        msgValue
    );
    (bool success, bytes memory result) =
universalReceiverDelegate.call(callData);
    Address.verifyCallResult(success, result, "Call to
universalReceiver failed");
    return result.length != 0 ? abi.decode(result, (bytes)) : result;
}
```

The *universalReceiver()* function has to return a **bytes** type.

```
function universalReceiver(
    bytes32 typeId,
    bytes memory /* data */
) public payable virtual returns (bytes memory result) {
```

However, the wrapper above will only decode the low-level call into a **bytes** type when **result** in non-zero. In fact, when **result** is zero the delegate has returned invalid output (no **returndata**). The callee will confuse the response with a valid zero-bytes return value and will presumably finish execution.

Recommended mitigation

When the **result** length is zero, revert the transaction.

Team response

It was decided by design to allow empty non-conforming bytes.

TRST-M-5 renounceOwnership will accidentally be one-step in a certain time period

- **Category:** Logical flaws
- **Source:** LSP14Ownable2Step.sol
- **Status:** Fixed

Description

In *_renounceOwnership()*, if renouncing did not start yet, **confirmationPeriodStart** and **confirmationPeriodEnd** will be 100 and 200 respectively. This means that if the current block is between those values, logic will funnel to the second stage handling. This is dangerous as we know Lukso will start from a new genesis block and other blockchains may use the same LSP code when they boot.

Recommended mitigation

If **_renounceOwnershipStartedAt** is zero, perform the first step regardless.

Team response

Fixed (applied recommendation).

Mitigation review

Fixed.

TRST-M-6 KeyManager ERC165 does not support LSP20

- **Category:** Specification errors
- **Source:** LSP6KeyManagerCore.sol
- **Status:** Fixed

Description

LSP6KeyManager supports LSP20 call verification. However, in *supportInterface()* it does not return the LSP20 **interfaceId**.

```
function supportsInterface(bytes4 interfaceId) public view virtual
override returns (bool) {
    return
```

```
        interfaceId == _INTERFACEID_LSP6 ||
        interfaceId == _INTERFACEID_ERC1271 ||
        super.supportsInterface(interfaceId);
}
```

As a result, clients which correctly check for support of LSP20 methods will not operate with the KeyManager implementation.

Recommended mitigation

Insert another supported **interfaceId** under *supportsInterface()*.

Team response

Fixed (applied recommendation)

Mitigation review

Fixed. An additional non-conforming ERC165 interface was created for verification clients, is a design decision.

TRST-M-7 Incorrect permission check due to confusion between empty call and the zero selector

- **Category:** Logical flaws
- **Source:** LSP6ExecuteModule.sol
- **Status:** Fixed

Description

In case the caller of LSP6 does not have **_PERMISSION_SUPER_CALL**, *_verifyAllowedCall()* is called to check they have specific permissions for the call. It will use the function *_extractExecuteParameters() to* get the call selector.

```
// CHECK if there is at least a 4 bytes function selector
bytes4 selector = executeCalldata.length >= 168
    ? bytes4(executeCalldata[164:168])
    : bytes4(0);
return (operationType, to, value, selector);
```

Due to the behavior above, the 0x00000000 selector would be confused with an empty call. Later in *_extractCallType()*, the call will not be marked as requiring **_ALLOWEDCALLS_WRITE** permission, if it's passing value.

```
if (operationType == OPERATION_0_CALL) {
    if (
        // CHECK if we are doing an empty call
        (selector == bytes4(0) && value == 0) ||
        // we do not require callType CALL
        // if we are just transferring value without `data`
        selector != bytes4(0)
    ) {
        requiredCallTypes = _ALLOWEDCALLS_WRITE;
    }
}
```

As a result, a user that has transfer permission but not write permission for 0xFFFFFFFF (function wildcard), will be permitted to pass calldata to the fallback function. The fallback as implemented in LSP0ERC725AccountCore would look up the extension for 0x00000000 selector and call it.

Recommended mitigation

Consider adding an **isEmptyCall** parameter to *_extractCallType()*. If it is not true, **_ALLOWEDCALLS_WRITE** should be turned on even for selector 0x00000000.

Team response

Fixed (applied recommendation).

Mitigation review

Fixed.

Low severity findings

TRST-L-1 LSP0 ownership functions deviate from specification and reject native tokens

- **Category:** Specification errors
- **Source:** LSP0ERC725AccountCore.sol
- **Status:** Fixed

Description

The LSP specifications define the following functions for LSP0:

```
function transferOwnership(address newPendingOwner) external payable;
function renounceOwnership() external payable;
```

However, their implementations are not payable.

```
function transferOwnership(address newOwner)
    public
    virtual
    override(LSP14Ownable2Step, OwnableUnset)
{
```

```
function renounceOwnership() public virtual
override(LSP14Ownable2Step, OwnableUnset) {
    address _owner = owner();
```

This may break interoperation between conforming and non-confirming contracts.

Recommended mitigation

Remove the **payable** keyword in the specification for the above functions, or make the implementations payable.

Team response

Fixed in Specs

TRST-L-2 Transfers of vaults from an invalid source are not treated correctly by receiver delegate

- **Category:** Logical flaws
- **Source:** LSP1UniversalReceiverDelegateUP.sol
- **Status:** Fixed

Description

In the *universalReceiver()* function, if the notifying contract does not support LSP9, yet the **typeID** corresponds to an LSP9 transfer, the function will return instead of reverting.

```
if (
    mapPrefix == _LSP10_VAULTS_MAP_KEY_PREFIX &&
    notifier.code.length > 0 &&
    !notifier.supportsERC165InterfaceUnchecked(_INTERFACEID_LSP9)
) {
    return "LSP1: not an LSP9Vault ownership transfer";
}
```

Recommended mitigation

Revert when dealing with transfers that cannot be valid.

Team response

Refactored with a different logic.

Mitigation review

The code above has been removed. This validation indeed does not seem to be necessary.

TRST-L-3 Relayer can choose amount of gas for delivery of message

- **Category:** Logical flaws
- **Source:** LSP6KeyManagerCore.sol
- **Status:** Acknowledged

Description

LSP6 supports relaying of calls using a supplied signature. The encoded message is defined as:

```
bytes memory encodedMessage = abi.encodePacked(
    LSP6_VERSION,
    block.chainid,
    nonce,
    msgValue,
    payload
);
```

The message doesn't include a gas parameter, which means the relayer can specify any gas amount. If the provided gas is insufficient, the entire transaction will revert. However, if the

called contract exhibits different behavior depending on the supplied gas, a relayer (attacker) has control over that behavior.

Recommended mitigation

Signed message should include the gas amount passed. Care should be taken to verify there is enough gas in the current state for the gas amount not to be truncated due to the 63/64 rule.

Team response

We decided not to implement this check as this would require the user to sign the gas limit provided to the call. This would affect the user experience with more complications. We decided to keep it as it is for now and implement the fix in the future if the problem is raised.

Additional recommendations

The uncheckedIncrement() code is not efficient

In various places a loop construct is used as below:

```
for (uint256 i; i < data.length; i = GasLib.uncheckedIncrement(i)) {
```

The resulting bytecode is actually unoptimized compared to an inline unchecked block. This creates a jump and several additional opcodes as overhead.

Team response

Fixed.

Improve code documentation

1. In LSP0ERC725AccountInitAbstract, the *_initialize()* call doesn't initialize the parents as they don't have non-zero initial state. It is worth explicitly noting this as usually derived Proxy classes have to initialize parent contracts.
2. Copy-paste error in documentation of *_getPermissionToSetAllowedERC725YDataKeys()*

   ```
   @dev retrieve the permission required to set some AllowedCalls
   for a controller.
   ```

Team response

Fixed Natspec.

Specification improvements

1. LSP20 specification should specify explicitly that the **callResult** passed to *lsp20VerifyCallResult()* would be an empty **bytes**, if the called function does not return values.
2. LSP0 specification states about the universal receiver:
   "*If an address is stored under the data key attached below and and this address is a contract that supports the LSP1UniversalReceiver interface id, forwards the call to the universalReceiver(bytes32,bytes) function on the address retreived. If there is no address stored under this data key, execution continues normally*."
   It would be good to clarify that execution continues normally also in the case that the address *does not* support LSP1UniversalReceiver interface id (rather than revert, for example).

Team response

Documented in specs.

Better verification

1. In LSP0ERC725AccountCore's *universalReceiver()*, the delegate value can be over 20 bytes.

```
bytes memory lsp1DelegateValue =
_getData(_LSP1_UNIVERSAL_RECEIVER_DELEGATE_KEY);
bytes memory resultDefaultDelegate;
if (lsp1DelegateValue.length >= 20) {
```

   It would be better to avoid errors and only call the delegate if length is precisely 20 bytes.

2. In LSP6ExecuteModule's *_verifyCanExecute()*, the 12 upper bytes of the address **uint256** are never parsed.

```
// MUST be one of the ERC725X operation types.
uint256 operationType = uint256(bytes32(payload[4:36]));
address to = address(bytes20(payload[48:68]));
```

   Good practice would be to verify they are all zeros. Such issues have the potential to lead to high-impact bugs as some address verifications could be bypassed.

Team response

Point #1 acknowledged as a design decision. Point #2 has been fixed.

Avoid reverts which break specs

In LSP0ERC725AccountCore's *isValidSignature()*, when owner is an EOA, *ECDSA.recover()* is used to fetch the signer. However, that function will revert if the signature is invalid. This function should only return **MAGICVALUE** or **FAILVALUE**, so it would be best to refactor and return **FAILVALUE** when the signature is invalid.

Team response

Fixed.

Improve logic in _verifyAllowedERC725YDataKeys()

When looping over the keys and validating them, if the number of keys found in the loop equals the input keys array length, it exits early. However, since some keys in the array are

pre-validated in *_verifyCanSetData()*, this condition cannot be reached although all keys are already validated. The fix would be to supply an additional parameter which is the number of keys pre-validated, and add it to the number of keys found for the check.

Team response

Implemented for gas optimization.

Set up account owner migration from EOA to LSP6

Once LSP6 becomes an account owner, all validations go through it. It seems easy to misconfigure the permissions and make an account uncallable when moving from an EOA owner to LSP6 owner. It is recommended to set up a validation at the smart contract level that the EOA can still access some key functionality.

Team response

The way it is set is a design decision. We do not want to implement the recommendation as otherwise it would make it too custom. The LSP11 standard can be used to solve this problem, if the account owner lose access to its permissions to recover access.

Limit transfer size per allowed call

LSP6 permissions either allow or disallow transfers for a specific function. From a practical perspective, it would seem necessary to limit spending to a certain amount, similar to how ERC20 allowances work.

Team response

This is a design decision. We want to keep it as it is for now.

Whitelisting functionality for universal receiver delegate

The default receiver delegate can be easily spammed by faking the receiving of assets on the account. This would make enumerating on the received assets unfeasible. It may be beneficial to have a global whitelist of credible assets, and accounts can opt-in to only track assets in the whitelist.

Team response

Considered to be implemented in future LSP Standards