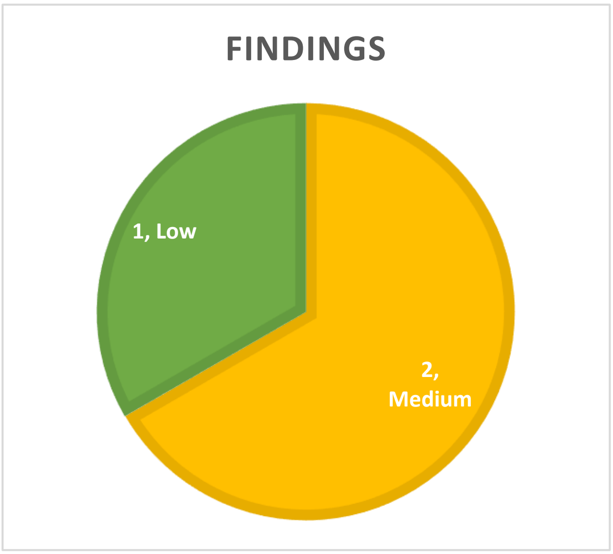
# Trust Security

Smart Contract Audit

LUKSO Genesis contract

09/02/2023

## Executive summary

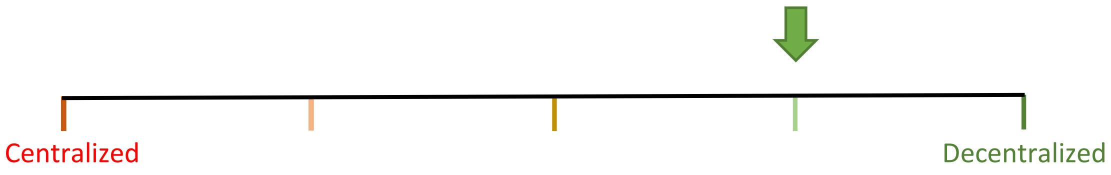


| | |
|---|---|
| Category | Deposit |
| Audited file count | 1 |
| Lines of Code | 183 |
| Auditor | Trust |
| Time period | 07-09/02/23 |

Findings

| Severity | Total | Fixed | Acknowledged | Disputed |
|---|---|---|---|---|
| High | 0 | - | - | - |
| Medium | 2 | 1 | 1 | - |
| Low | 2 | 1 | 1 | - |

Centralization score



Signature

Digitally signed by Trust
Date: 2023-04-19 11:30:08

# Document properties

## Versioning

| Version | Date | Description |
|---|---|---|
| 0.1 | 09/02/2023 | Client report |
| 0.2 | 19/04/2023 | Mitigation review |

## Contact

Or Cyngiser, AKA Trust

boss@trustindistrust.com

# 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

- contracts/LUKSOGenesisValidatorsDepositContract.sol

## Repository details

- **Repository URL:** https://github.com/lukso-network/network-genesis-deposit-contract
- **Commit hash:** c8a1b77aee09b12fe28e766eb2aef5d57cd14224
- **Mitigation commit hash:** 1c58be71d7d8171ec2a0403c3474edd0938706b1

## 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 many different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

## Qualitative analysis

| Metric | Rating | Comments |
|---|---|---|
| Code complexity | **Excellent** | Project has been kept similar to the ETH2 contract and did not introduce unneeded complexity. |
| Documentation | **Excellent** | Project is very well documented, with special attention to auditing prep. |
| Best practices | **Good** | Project implements industry standards efficiently. |
| Centralization risks | **Good** | Project mostly kept to high degree of decentralization. |

# Findings

## Medium severity findings

### TRST-M-1 Voting on LYX total supply is not defined properly

- **Category:** Undefined processing
- **Source:** LUKSOGenesisValidatorsDepositContract.sol
- **Status:** Acknowledged

#### Description

When users deposit to the Genesis contract, they may choose the LYX total supply on LUKSO (in $M). However, the contract and documentation do not define how the result is chosen. This may lead to large disagreements in the community regarding the allocated supply. Consider the following example:

| Vote % | Desired supply |
|---|---|
| 40% | 10M |
| 30% | 60M |
| 30% | 65M |

In the example above, voting results can be interpreted in three ways. The most voted count is 10M. The average count is 41.5M and the median is 60M. Since the voting decision was not documented, it is left to the maintainers of LUKSO to settle on the final supply which is bound to raise antagonism from at least some of the community.

#### Recommended mitigation

Clearly define the supply determination algorithm before any voting takes place.

#### Team response

Vote is just indicative as mentioned on the launchpad and the repository's ReadMe.

### TRST-M-2 *freezeContract* can be called at any time

- **Category:** Time-sensitive operations
- **Source:** LUKSOGenesisValidatorsDepositContract.sol
- **Status:** Fixed

#### Description

The Genesis deposit contract accepts new deposits as long as it is not frozen. A privileged address may call *freezeContract()* at any time to permanently halt deposits. The issue is that this introduces various undesired effects. Firstly, it is putting a lot of power in the owner's hands, as they may at any point censor a holder that has not yet deposited their LYXe by freezing the contract. Owner may also simply retain majority rule of LYX by freezing shortly after depositing their own LYXe. Finally, it is likely that when freezing naturally happens, there will be users who would be surprised the deposit window has passed.

**Recommended mitigation**

Contract should be deployed with a freeze timestamp, which is not controllable and well communicated to the LUKSO community.

**Team response**

Fixed. We implemented a delayed freeze with a value of 46,253 blocks (~1 week).

**Mitigation review**

Freeze logic determined to be safe.

## Low severity findings

### TRST-L-1 Contract incorrectly implements ERC165

- **Category:** Specs mismatch
- **Source:** LUKSOGenesisValidatorsDepositContract.sol
- **Status:** Fixed

**Description**

The deposit contract implements ERC165 as below:

```
function supportsInterface(bytes4 interfaceId) external pure override
returns (bool) {
    return
        interfaceId == type(IERC165).interfaceId ||
        interfaceId == type(IDepositContract).interfaceId;
}
```

It advertises the **IDepositContract** interface, which includes the ETH2 *deposit()* function:

```
function deposit(
    bytes calldata pubkey,
    bytes calldata withdrawal_credentials,
    bytes calldata signature,
    bytes32 deposit_data_root
) external payable;
```

However, the code does not implement this function (deposits operate via *tokensReceived*() callback). Therefore, code does not adhere to the specification. This could create some needless confusion around the deposit mechanism.

**Recommended mitigation**

Remove the **IDepositContract** interface from *supportsInterface()*.

**Team response**

Fixed. We removed the **IDepositContract** from *supportsInterface().*

**Mitigation review**

Fixed.

## TRST-L-2 Contract incorrectly implements ERC165

- **Category:** Front-running issues
- **Source:** LUKSOGenesisValidatorsDepositContract.sol
- **Status:** Acknowledged

**Description**

In refactoring done between the initial audit and the mitigation review, the following code was introduced to the deposit flow.

```
// Compute the SHA256 hash of the pubkey
bytes32 pubKeyHash = sha256(pubkey);
// Prevent depositing twice for the same pubkey
require(
    !_registeredPubKeyHash[pubKeyHash],
    "LUKSOGenesisValidatorsDepositContract: Deposit already
processed"
);
// Mark the pubkey as registered
_registeredPubKeyHash[pubKeyHash] = true;
```

Essentially, each validator public key can only be used once. Therefore, a deposit TX can be observed in the mempool and front-ran by an attacker. They would lose the deposit amount as they don't have the matching private key, however they would block the TX for the user, who will be unable to unlock the deposit due to the attacker-supplied withdrawal credentials. This attack could be repeated, harming the UX.

**Team response**

Acknowledged. A fix will not be applied as the attack is unlikely. Attacker will lose any deposit made.

## Additional recommendations

### Improve state hygiene

The *freezeContract()* privileged function is implemented as:

```
function freezeContract() external {
    require(msg.sender == owner,
"LUKSOGenesisValidatorsDepositContract: Caller not owner");
    isContractFrozen = true;
}
```

It is never checked that the contract is not frozen before setting the variable to true. While there is no harmful impact, it's best to maintain a predictable state-machine and only allow a single call to *freezeContract()*.

### Logging of state changes

It is recommended to emit an event on important state changes in a contract. Consider adding an event called *ContractFrozen*(), to improve transparency and interoperability with the contract.

### Documentation issues

It is observed that the **depositData** parameter for *tokensReceived()* has recently been appended the additional byte for voting. However, the function docs are rather misleading.

```
* - `depositData` MUST contain:
*    • pubkey - the first 48 bytes
*    • withdrawal_credentials - the following 32 bytes
*    • signature - the following 96 bytes
*    • deposit_data_root - last 32 bytes
*/
```

```
// 208 = 48 bytes pubkey + 32 bytes withdrawal_credentials + 96 bytes
signature + 32 bytes deposit_data_root
require(
```

The README file is also not synchronized with the change. It is very recommended to change those to make sure any interaction with the code is done properly.

### Improved testing

There is a test that is not synchronized with the change in **depositData** structure.

```
it("should revert when data's length is smaller than 208", async ()
=> {
  const data = ethers.utils.hexlify(ethers.utils.randomBytes(207));
  await expect(
    context.LYXeContract.connect(validators[0]).send(
      context.depositContract.address,
      DEPOSIT_AMOUNT,
      data
    )
  ).to.be.revertedWith(
    "LUKSOGenesisValidatorsDepositContract: depositData not encoded
properly"
  );
});
```

We advise that the test creates sample data of length **208**, which is the highest invalid data size.

## Dangerous behavior of internal function

A utility function has been added to the ETH2 deposit code called *convertBytesToBytes32()*.

```
function _convertBytesToBytes32(bytes calldata inBytes)
    internal
    pure
    returns (bytes32 outBytes32)
{
    bytes memory memoryInBytes = inBytes;
    assembly {
        outBytes32 := mload(add(memoryInBytes, 32))
    }
}
```

Because of the usage of assembly, this function will return 0x0000… when receiving an empty bytes array, which is incorrect. Fortunately, it is only used with a 32-byte input, making it safe. However, it is not advised to leave such code in a code base, as it is possible that it would be used in unsafe ways in the future and the compiler will not emit any warnings.