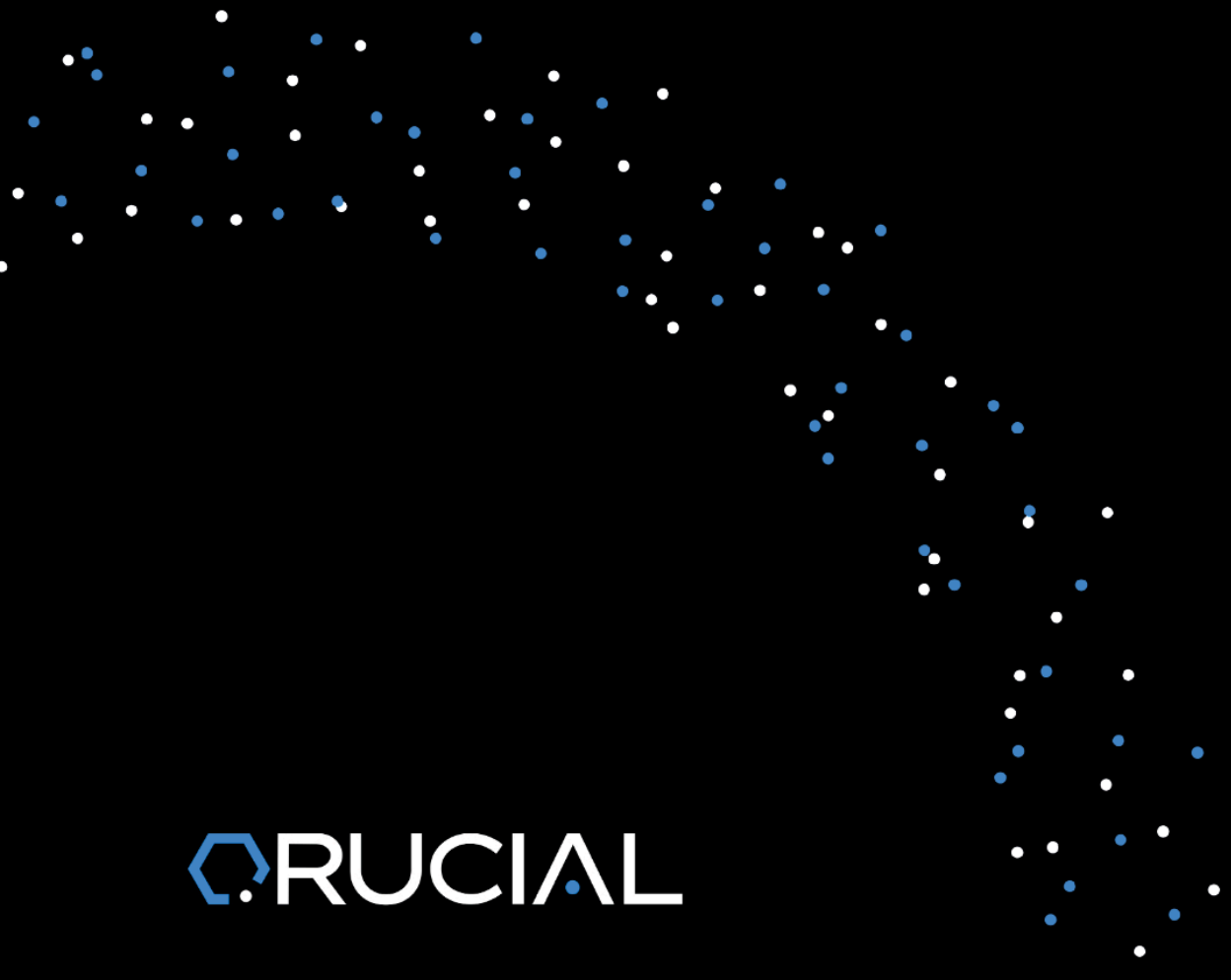
SMART CONTRACT SECURITY REPORT.

Personally audited and created for:
NextEarth Launchpad.



CRUCIAL

16th of April 2022

TABLE OF CONTENTS

Introduction

This report has been prepared for NextEarth Launchpad. An extensive analysis has been performed by manual review, static analysis and symbolic execution.

In our Audits, we focus on the following areas:

- Analysing that the smart contract's logic meet the intentions of the client.
- Examining the smart contracts against conventional and unconventional attack vectors
- Verifying the codebase meets the current Industry standard and best practices
- Cross-referencing the Project against the implementation, contract, and structure of similar Industry-leading Projects
- Elaborate line by line manual review of the entire Codebase.

The findings of the security evaluation are considered low.
We advise you to address these findings as soon as possible to assure a foremost level of security for your project and community.

- Increase good coding practices for a better structure in the source code
- Increase the number of unit test to cover all possible angles of use cases
- Add more comments per function for readability.

Overview

Project Summary

Project name	NextEarth LaunchPad
Platform	Polygon
Language	Solidity
Codebase	Contracts_LP2.zip
Hash	sha256: d4b527618be7c37eecde77497d3122bb2185b93ee9e1cb88e6e4b2b764e6d54c

Audit Summary

The NextEarth team has requested a security smart contract audit from QRUCIAL on their LaunchPad project. It consists of the main launchpad, presale, vault smart contracts including the necessary interfaces and examples.

Based on the provided specifications, QRUCIAL assessed the smart contracts from security perspective and identified informational, low and medium level risks. During the audit, these were reported to the NextEarth LaunchPad team, fixes were applied.

We conclude that the project was coded with high quality standards and even if vulnerabilities were found, these are limited in scope.

Vulnerability Summary

Level	Total	Pending	Rejected	Accepted	Partially fixed	Fixed
Critical	0	-	-	-	-	-
High	0	-	-	-	-	-
Medium	2	2	-	-	-	-
Low	5	5	-	-	-	-
Informational	3	3	-	-	-	-

Scope

Audited Code:	NextEarth LaunchPad
Blockchain Explorer Link:	Not yet deployed.
Compiler:	0.8.0 and 0.8.1
Number of files:	1
Scope (list of files)	Contracts_LP2.zip

Risk Classifications

Critical:
Vulnerabilities that can lead to a loss of funds, impairment, or control over the system or its function.
We recommend that findings of this classification are fixed immediately.

High:
Findings of this classification can impact the flow of logic and can cause direct disruption in the system and the project's organization.
We recommend that issues of this classification are fixed as soon as possible.

Medium:
Vulnerabilities of this class have impact on the flow of logic, but does not cause any disturbance that would halt the system or organizational continuity.
We recommend that findings of this class are fixed nonetheless.

Low:
Bugs, or vulnerability that have minimal impact and do not pose a significant threat to the project or its users.
We recommend that issues of this class are fixed nonetheless because they increase the attack surface when your project is targeted by malicious actors.

Informational:
Findings of this class have a negligible risk factor but refer to best practices in syntax, style or general security.

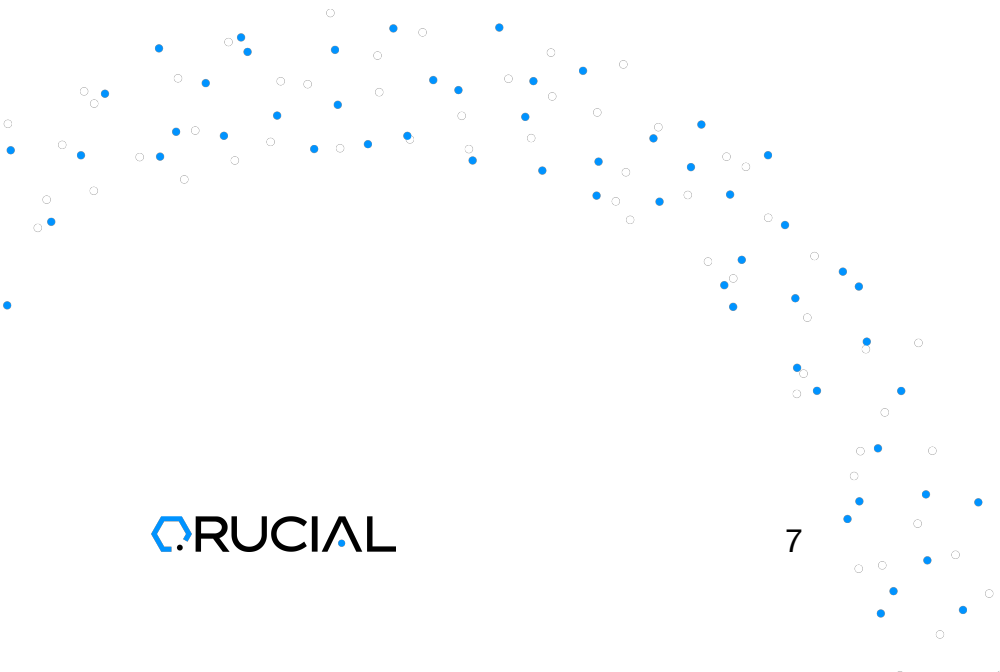
MEDIUM: Reentrancy with limited impact in SecureStorage.sol

Description:
The non-reentrant modifier is not implemented in the SecureStorage that is used for the tokens participating in the Launch Pad project.

Impact:
The impact is limited to the project owner, but the logic bug exists and is possible to exploit.

Recommendations:
Implement the non-reentrant modifier as it is implemented in other parts of the project. Note that a more reentrant calls exist in the project, but QRUCIAL found them non-exploitable in a meaningful way.

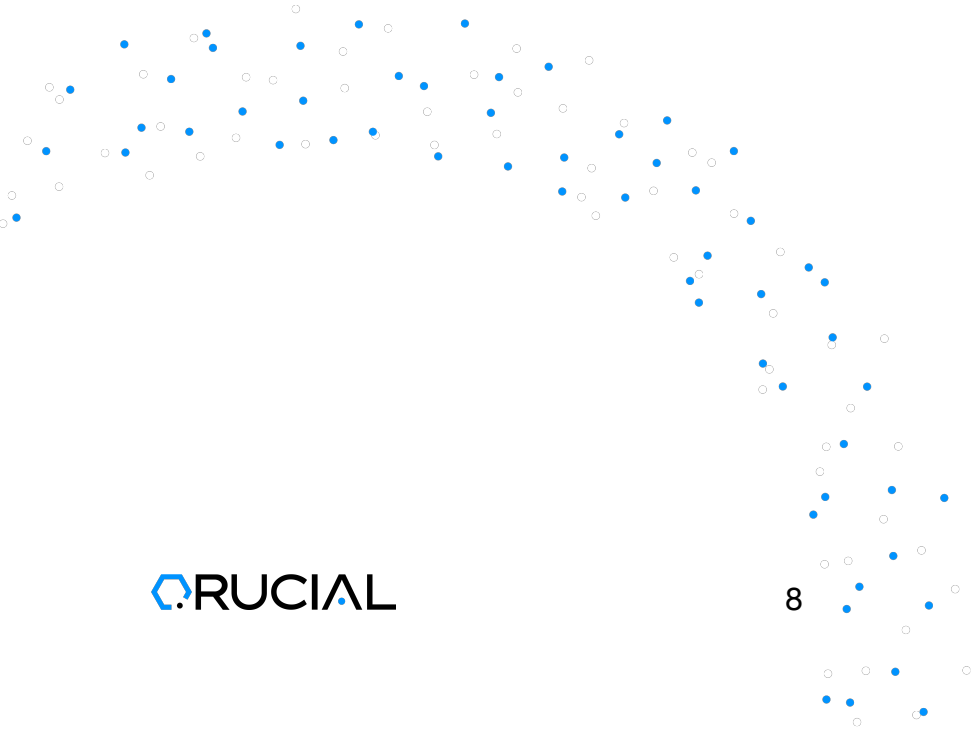
References:
https://docs.soliditylang.org/en/v0.8.13/security-considerations.html#re-entrancy



Technical Details:

```
function transfer(
IERC20 token,
address beneficiary,
uint256 amount
) external override onlyOwner {
_transfer(token, beneficiary, amount);
}

function approve(
IERC20 token,
address spender,
uint256 amount
) external override onlyOwner {
require(address(token) != address(0x0), "LPC: Token is zero");
require(spender != address(0x0), "LPC: Spender is zero");
SafeERC20.safeApprove(token, spender, amount);
}
```



MEDIUM: Lack of decentralization by single point of failure

Description:
During the audit it was found that the project have single point of failures in the system: the signer backend controls most of the major impact functions. This is a common issue for most projects in 2022, but a solution can be developed.

We consider this issue medium level because the business model does not focus on being fully decentralized, but partially and the risk of having single point of failures can breach the whole project.

LPC.sol implements renounceOwnership() which will be usable from the LaunchPad backed. In case the backend is breached, this call can destroy the whole project, including access to funds. In itself this is not a direct vulnerability, but a risk that needs to be addressed with care.

Impact:
The smart contract logic is only as secure as the backend. Impact applies in case of backend breach: losing control over the smart contract and funds.
In case the signer key or account is breached, the project might be taken down as a whole. It can happen through multiple scenarios, examples are the following:

- Stealing the device physically that stores the private keys
- Exploitation of the system that stores the private keys
- By human error, losing the device that stores private keys
- System error, eg. ssd/disk failure and lack of usable backup
- Insider threat
- Incident of the device owner and having no possibility to restore the private keys

Recommendations:
- Make sure the backend is hardened and penetration tested.
- Implement a logic that requires multiple signatures to take actions. An example can be the use of threshold ESCDA or a solidity based logic where multiple signatures from different sources are required to execute major impact functions.
- If not required by business, the function is better to be removed. Note the possibility of `transferOwnership()` too.

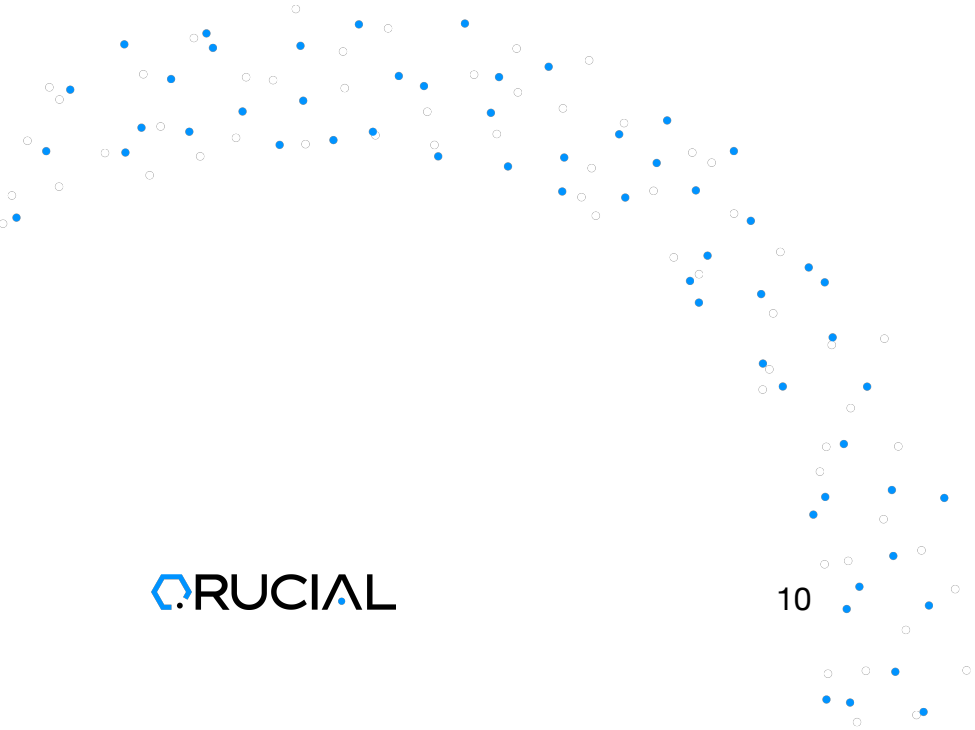
References:

Threshold ECDSA: https://eprint.iacr.org/2019/114.pdf

https://github.com/Qrucial/Voronoi

https://docs.openzeppelin.com/contracts/3.x/access-control

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol



Technical Details:

The system is built on the backend signer logic. Example used in the project:

```
function _checkSignature(bytes memory signature, bytes memory message)
private
view
{(address _sender, string memory _salt) = abi.decode(
message,
(address, string));
require(_sender == msg.sender, "LPC: Invalid Sender");
address _signer = lpc.signers(address(this));
address recovered = keccak256(abi.encodePacked(_sender, _salt))
.toEthSignedMessageHash()
.recover(signature);
require(recovered == _signer, "LPC: Invalid signature");}
```

```
/**
* @dev Leaves the contract without owner. It will not be possible to call
* `onlyOwner` functions anymore. Can only be called by the current owner.
*
* NOTE: Renouncing ownership will leave the contract without an owner,
* thereby removing any functionality that is only available to the owner.
*/
function renounceOwnership() public virtual onlyOwner {
_transferOwnership(address(0));
}
```

LOW: USDC price fixed at deploy time (multiple contracts)

Description:
The USDC price and hence all the related token prices are fixed at the time of the deployment, through the constructor.

Impact:
The impact is limited as the business requirements set a short period of time until these values are actually used. The business accepted this risk.

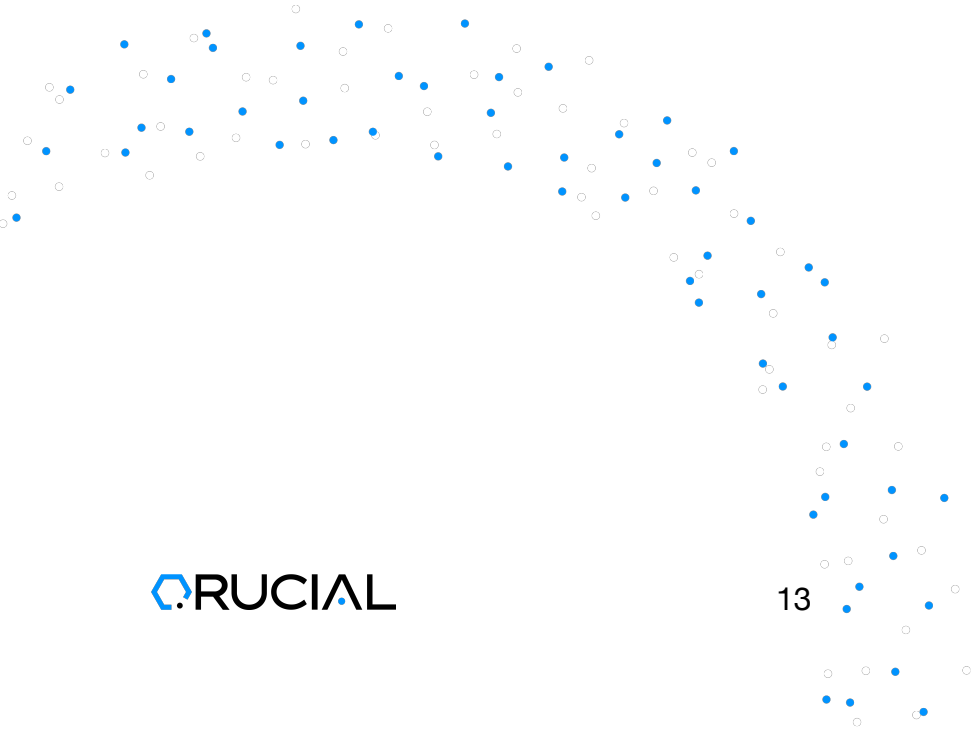
Recommendations:
Implementing an oracle could be an option, but this also opens up an attack surface from the oracle's side. As the business accepted this risk, there is nothing else to do.

References:
https://fravoll.github.io/solidity-patterns/oracle.html

Technical Details:

The vulnerability applies for Presale.sol, TieredPresale.sol and the examples provided.

```
constructor(
uint256 cap,
IERC20 _token,
IERC20 usdc,
IERC20 nxtt,
uint256 tokenUSDC,
uint256 nxttUSDC,
ILPC lpc
) Presale(_token, usdc, nxtt, nxttUSDC, lpc) {
// Commented out for coverage reasons
// require(tokenUSDC > 0, "TokenUSDC is zero");
_rate = tokenUSDC;
_cap = cap;
}
```



LOW: Use of multiple solidity versions

Description:
The project is using 0.8.0 and 0.8.1 versions of the compiler in different contracts.

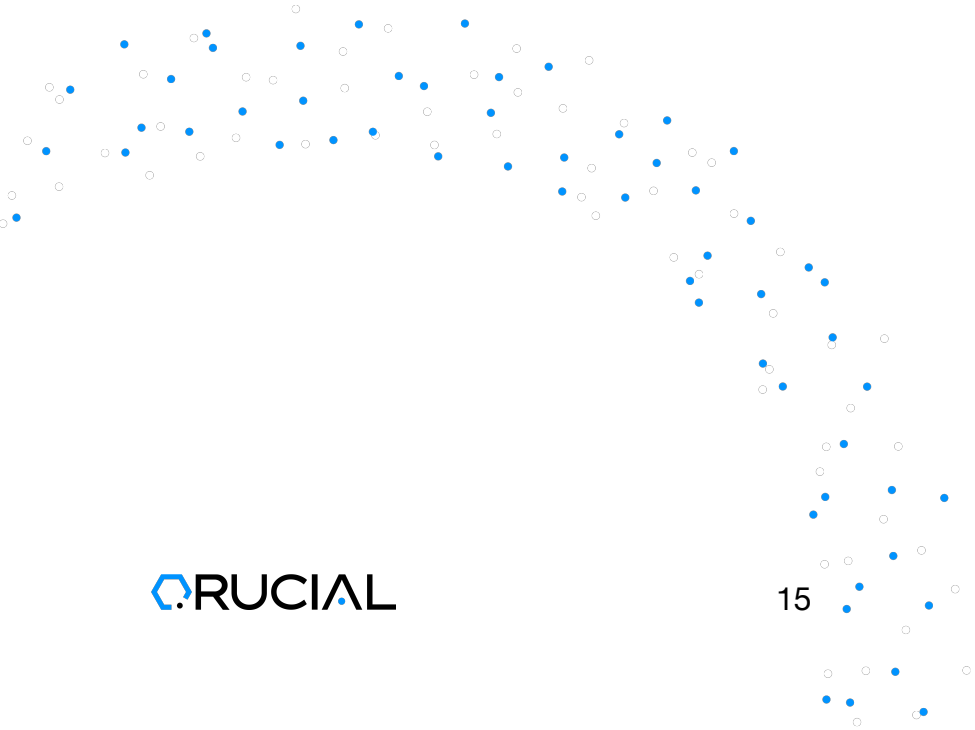
Impact:
The compilers behave slightly differently.

Recommendations:
Use a more recent compiler version, eg. 0.8.12.

References:
https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Technical Details:

Version used: ^0.8.0, ^0.8.1



LOW: The used solidity compiler version has known security issues

Description:
The solidity compiler version which was sent to QRUCIAL uses 0.8.0 and 0.8.1. These have three issues that are considered low risk. Based on the solc security documentation these are not exploitable, but better to be addressed.

Impact:
A few low level risks can be avoided by using the latest compiler version.

Recommendations:
Use a more recent version (eg. 0.8.12) of the compiler.

References:
https://blog.soliditylang.org/category/security-alerts/

Technical Details:

pragma solidity ^0.8.0;

pragma solidity ^0.8.1;

LOW: SPDX License is missing

Description:
SPDX license identifier not provided in source file. SPDX is an open standard for communicating software bill of material information, including provenance, license, security, and other related information.

Impact:
Licensing of the files are not clear.

Recommendations:
Add the SPDX license to all the source files, so it becomes clear what legal use is intended.

References:
https://spdx.org

Technical Details:

The following code part is not used in the project:

```
SPDX-License-Identifier: <SPDX-License>
```

LOW: Lack of zero check in ETHstorage.sol

Description:
The address passed to sendETH() is not checked and can be zero.

Impact:
Limited to the owner of the contract only.

Recommendations:
Implement a zero check against `address(0x0)`

References:
NA - Not Applicable.

Technical Details:

storage/ETHStorage.sol#20

```
function sendETH(address payable beneficiary) external override onlyOwner {
(bool sent, ) = beneficiary.call{value: address(this).balance}("");
require(sent, "LPC: Failed to send ETH");
}
```

INFORMATIONAL: Unexpected logic possibility in LP participants' contract

Description:
Approved.sol is used to limit the possibility to participate in the LP project only to those that are accepted by the business. Still, risks need to be addressed:

1. In case of a proxy smart contract, the approved project can easily go rouge by modifying the underlying logic.
2. In case of a modified example presale or vault, backdoors or unexpected logic can be hidden that can go rouge.

Impact:
The impact is negligible as long as the approved projects are the same as in the examples provided or well audited.

Recommendations:
Make sure the approval process is strong on the business side. Any project that modifies or does not use the LP example contracts are highly recommended to be audited separately.

References:
NA

Technical Details:

NA

INFORMATIONAL: Unexpected logic possibility in LP participants' contract

Description:
The LP project implements its own logic on the protection against replay attacks. Even if deployed on multiple chains, the signed messages include the contract addresses and are checked against, so the standard replay attacks won't work. It is noted that the signed messages do not expire.

Impact:
There is no vulnerability related impact

Recommendations:
Note the following options for future development. You can also:
- Make nonce start at a higher count than 0
- Add chainID, name of the public chain in smart contracts
- Add address(this) in keccak256()

References:
https://medium.com/cypher-core/replay-attack-vulnerability-in-ethereum-smart-contracts-introduced-by-transferproxy-124bf3694e25

Technical Details:

```
function _checkSignature(bytes memory signature, bytes memory message)
private
view
{
(address _sender, string memory _salt) = abi.decode(
message,
(address, string)
);
require(_sender == msg.sender, "LPC: Invalid Sender");
address _signer = lpc.signers(address(this));
address recovered = keccak256(abi.encodePacked(_sender, _salt))
.toEthSignedMessageHash()
.recover(signature);
require(recovered == _signer, "LPC: Invalid signature");
}
```

INFORMATIONAL: Typo in SecureStorage.sol

Description:
Typo. The word “arbitary“ should be replaced to “arbitrary”.

Impact:
Those who look at the code might think nobody else read it.

Recommendations:
Fix the typo.

References:
NA

Technical Details:

```
SecureStorage.sol, line 14. "arbitary" -> arbitrary
```

DISCLAIMER

This report is fixed to the scope and subject to terms and conditions of the service agreement provided to the customer. This report must not be referred, transmitted or disclosed to a third party without QRUCIAL's prior written consent.

This report is not an endorsement disapproval of a team, a product, a service, a company, or an individual. This report should not be considered as financial advice and does not indicate any financial or economic value in an asset, an asset class a product or service. This report is not to be seen as an indication of the legal compliance regarding of a project an asset, an asset class or a business model.

This report does not provide the guarantee, that a project is without bugs, errors vulnerabilities or code that is harmful to machines, software, or data. This report is also no indication of the validity of any business model or technology. Each individual organization is responsible to do their own due diligence or security assessment. This report is not to be seen as a guarantee of the functionality of a technology or its security. The use of access or information in this audit is used on the risk of the reader or user of this document.

This report holds no guarantee that the given information meets requirements of any kind, is compatible with applications, any software or systems. It is also not guaranteed that this audit is free of errors or harmful code or will cause interruptions of any software or systems. We do not give any guarantee of accuracy, reliability, or correctness of the information given in this audit. All third-party material provided to the client may be subject to the terms and conditions of third parties. All third-party material provided is provided without the guarantee of correctness. No third-party has the right to use the trademark QRUCIAL, its products or services as a reference or endorsement of its own products or services without prior written consent.

THANK YOU.

Our team gave their full commitment to craft this audit with precision and focus on details with the goal to support you improving your work.

With this audit we want to provide you a guidance to make your project more
secure and for presenting your
community a product they can trust in.

We make security our priority, so you don‘t have to.

RUCIAL

----Polkadot{.js} account validation address----

5EHagRYLNsCJUx5bA5Y8MZWLqPVzqQbFMC76V68Wzv7GHHXD

----Polkadot{.js} account validation address----

QRUCIAL

QRUCIAL OÜ c/o Gate to Baltics
Narva mnt 7-652, 10117 | Tallinn, Estonia