RED4SEC

Security Audit Report

09/22/2022

Legacy Network

RED4SEC

Content

Introduction

Legacy Network is an international company based in Liechtenstein/Switzerland. Their mission is to improve people's lives around personal development. They firmly believe, that personal development is the key to success in all life areas, as well as the solution for many global problems.

Legacy Network's central product is a gamified education app, which uses a play to earn system to reward its users with crypto. The app is powered by artificial intelligence, which analyzes serveral aspects of a user's personality and reflects the collected information back to him.

As solicited by **SayNode** and as part of the vulnerability review and management process, Red4Sec has been requested to perform a security code audit in order to evaluate the security of the **Legacy Network** project.

The report includes specifics retrieved from the audit for all the existing vulnerabilities of **Legacy Network**. The performed analysis shows that the smart contracts do not contain any critical issues.

Disclaimer

This document only represents the results of the code audit conducted by Red4Sec Cybersecurity and should not be used in any way to make investment decisions or as investment advice on a project.

Likewise, the report should not be considered neither "endorsement" nor "disapproval" of the guarantee of the correct business model of the analyzed project, nor as guarantee on the operation or viability of the implemented financial product.

Red4Sec makes full effort and applies every resource available for each audit, however it does not warrant the function, nor the safety of the project and it cannot be deemed a sufficient assessment of the code's utility and safety, bug-free status, or any other declarations of the project. Additionally, Red4Sec makes no security assessments or judgments about the underlying business strategy, or the individuals involved in the project.

Blockchain technology and cryptographic assets come with its own new risks and challenges, where the ecosystem, platform, its programming language, and other software related to said technology can have vulnerabilities that could lead to exploits. As a result, the audit cannot guarantee the explicit security of the audited projects.

The audit reports can be used to improve the code quality of smart contracts, to help limit the vectors of attack and to lower the high level of risks associated with utilizing new and continually changing technologies such as cryptographic tokens and blockchain, but they are unable to detect any future security concerns with the related technologies.

Scope

Red4Sec Cybersecurity has made a thorough audit of the **Legacy Network** security level against attacks, identifying possible errors in the design, configuration or programming; therefore, guaranteeing the availability, integrity and confidentiality of the project and the possible assets treated and stored.

The scope of this evaluation includes the following items provided by **SayNode**:

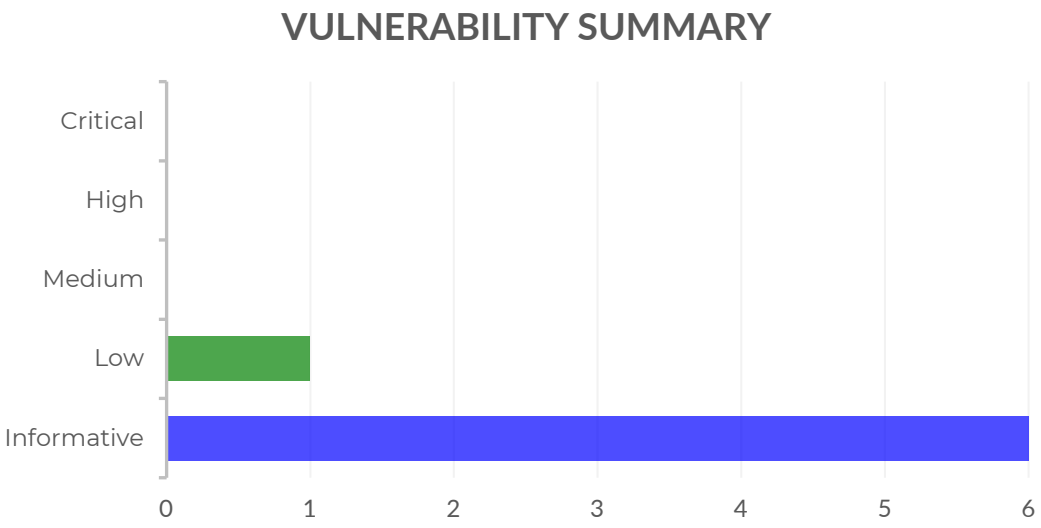
- **ERC20**
  - **Address.sol**
    2779b53a9ec322a15d52831b0c801f6250a01215845eb79197a4e3de11d9ed83
  - **Context.sol**
    6a6682301c136828a3ec2670f83f82d8f7f0a09edf2065faa6a73abe896e93ce
  - **ERC20.sol**
    34cdf94b2a0f67b06f64d692016d418d2534b89c51b877ccf6e58c85140c30fc
  - **IERC20.sol**
    9f17dd192ebc334ee7c60dd206339051ed5a841e1bd999904cea39dece5065dd
  - **IERC20Metadata.sol**
    d58377f3fda62337e0b78edbb728ed77537c6bd306cff66840a060cf876956a9
  - **LGCN.sol**
    63890ec3371c8f6fb9a716956a50d37e4f00f41024c3efd7a92df3f6b0b6d023
  - **Migrations.sol**
    4fd6092bdfa8b42f19d535c5ac69c4323b0b894717c699e58d5552eeabd04cd4
  - **SafeMath.sol**
    a45050900928cea9c094456e44c52f2ab68c99e7e5e7c434fea1d370f9479368
- **VET**
  - **Context.sol**
    36810bd9b4b9e9abd722d2401cd960c02efb17f52d3393f1e3e24ab43437c4f4
  - **ERC20.sol**
    daa91e0631eb5b2656eb6ac552426683c255000c85a17e3662cb9dc7175e79f5
  - **IERC20.sol**
    9428a6535fa667c2ad3a4634cff43916bb5506a1f0bfb9858a02dfbca0f961b4
  - **LGCN.sol**
    7aa51b2dac74443a10afbb60761842541820dcc5abe3edd3234fddf3a7d6cf2e
  - **Migrations.sol**
    7be193e29495a3dbcc6cd661e8e7f5fd999bd55f1046fe5ad983167ecbb67e69
  - **SafeMath.sol**
    e2b604005a53e31aaa2d59856db22c9db4baed244fac11a77e35227892115287
  - **authority.sol**
    0746a6f0cea0934acdf67a3fdd42955c4f89dcab61ec705f2ab8e04fb6948573
  - **builtin.sol**
    2e00d87062a8b695746a65a4fbb4ed9641f72f80ac7f259640f3ba0fca12183a
  - **energy.sol**
    68fd9b6debc4ea372e2683ac55e54a6ddc811d2b0ef56c8651e796c1d2a2d54e
  - **executor.sol**
    73be5469a12bc4206e57764c3f4fda9e8b4cf896e5634beab6ea87eca81d0b0d
  - **extension.sol**
    f158702bc211c6750d37acc787145ca0f6c613c0cb12b9528218787f6c8da36a
  - **params.sol**
    c65efcb19b3f1e65306b7dcc79bd30ed2b7d9307751a615cb79cff54415ba07d
  - **prototype.sol**
    b220dcf178595eb54f3a9073a26b4a14b08126c9dfb0cb9c6a612cc9b95711c9

Executive Summary

The security audit against **Legacy Network** has been conducted between the following dates: **04/07/2022** and **08/07/2022**.

Once the analysis of the technical aspects of the environment has been completed, the performed analysis shows that the audited source code contains non-critical issues that should be mitigated as soon as possible.

During the analysis, a total of **7 vulnerabilities** were detected, these vulnerabilities have been classified by the following level of risks, defined in Annex A.



Conclusions

To this date, **22/09/2022**, the general conclusion resulting from the conducted audit and the subsequent review of the fixes, is that the **Legacy Network project is secure**. Nevertheless, Red4Sec has found a few potential improvements, these do not pose any risk by themselves, and we have classified such issues as informative only, but they will help to continue to improve the security and quality of its developments.

The general conclusions of the performed audit are:

- The issues detected do not impact the security of the project, but they do affect its usability. An example of this would be the issue **"Wrong VIP-180 implementation"**, since errors could occur in the VeChain when the `VIP-180` standard is not well implemented. This issue has been subsequently fixed by the Legacy Network team.
- The **Legacy Network** project has been **developed using outdated versions**, and in some cases, versions marked as obsolete, both from the compiler and from third-party libraries. This is an absolutely discouraged practice and should be solved prior to the deployment.
- The **absence of the Unit Test** has been detected, during the security review, this is a highly recommended practice that has become mandatory in projects destined to manage large amounts of capital.
- A **few low impact issues** were detected and classified only as informative, but they will continue to help **Legacy Network** improve the security and quality of its developments.
- All the proposed recommendations that were considered necessary by Red4Sec in order to improve the security of the project were properly applied by the **Legacy Network** team.

Vulnerabilities

In this section, you can find a detailed analysis of the vulnerabilities encountered upon the security audit.

List of vulnerabilities

Below, we have gathered a complete list of the vulnerabilities detected by Red4Sec, presented and summarized in a way that can be used for risk management and mitigation.

Table of vulnerabilities			
ID	**Vulnerability**	**Risk**	**State**
LGC-01	Wrong VIP-180 implementation	**Low**	**Fixed**
LGC-02	Logic discrepancies between chains	**Informative**	**Partially Fixed**
LGC-03	Outdated third-party libraries	**Informative**	**Assumed**
LGC-04	Incorrect pragma range	**Informative**	**Fixed**
LGC-05	Outdated compiler	**Informative**	**Assumed**
LGC-06	GAS optimization	**Informative**	**Partially Fixed**
LGC-07	Solidity literals	**Informative**	**Fixed**

Vulnerability details

In this section, we provide the details of each of the detected vulnerabilities indicating the following aspects:

- Category
- Active
- Risk
- Description
- Recommendations

Wrong VIP-180 implementation

Identifier	Category	Risk	State
LGC-01	Business Logic Errors	**Low**	**Fixed**

The methods `transfer`, `approve` and `transferFrom` in **VIP-180** do not return a *boolean* value, which does happen in the **ERC-20** standard, this implies that the **VIP-180** implementation is erroneous and could, in certain situations, produce failures when calling the `LGCN` The methods `transfer`, `approve` and `transferFrom` in **VIP-180** do not return a *boolean* value, which does happen in the **ERC-20** standard, this implies that the **VIP-180** implementation is erroneous and could, in certain situations, produce failures when calling the `LGCN` token by third-party contracts in the VeChain.

According to the **VIP-180** standard, instead of returning *false* when one of these methods fail, there should be an error thrown.

> `The function MUST throw if the transaction failed.`

Reference:

- VIP-180.md#transfer
- VIP-180.md#approve
- VIP-180.md#transferfrom

Recommendations

- Correctly implement the **VIP-180** standard.
- The use of **VIP-180** is safe but it can cause errors in the VeChain as the **VIP-180** standard is not properly implemented.

Source Code References

- LGCN_VET_Contracts/contracts/ERC20.sol: 117, 147, 170

Fixes Review

- The issue has been addressed in the latest review.

Logic discrepancies between chains

Identifier	Category	Risk	State
LGC-02	Business Logic	**Informative**	**Partially Fixed**

The **Legacy Network** project implements the `LGCN` token in different chains, both based on EVM, so the logic between both chains is expected to be similar to avoid problems for users and developers that use said contracts.

However, some discrepancies have been found that could affect the functionality and usability of the cross-chain contracts.

The name of the token differs between chains, this could cause usability or mistrust problems when using one or the other chain, the user could understand that both tokens are not related and that they do not operate with the one of a certain chain.

```
// VET
string private constant _name = "LEGACY NETWORK";
string private constant _symbol = "LGCN";
uint8 private _decimals;


// ERC20
contract LGCN is ERC20 {
constructor() ERC20("LEGACY TOKEN", "LGCN") {
    _mint(msg.sender, 720000000 * 10**decimals());
}
```

Source references

- LGCN_ERC_Contracts/contracts/LGCN.sol: 8
  - `"LEGACY TOKEN"`
- LGCN_VET_Contracts/contracts/ERC20.sol: 42
  - `"LEGACY NETWORK"`

In the **VET** contract it is not ensured that the address that contains the initial tokens exists, however in the ERC20 token it is done, since it makes `_mint` over `msg.sender`.

Source references

- LGCN_ERC_Contracts/contracts/LGCN.sol: 9
- LGCN_VET_Contracts/contracts/LGCN.sol: 14

The `transferFrom` method in **ERC20** does not discount the `allowance` if it has been set to `type(uint).max`, however the logic corresponding to the same method in the **VeChain** does discount it. Producing a discrepancy of behaviors and of `allowance`.

Source references

- LGCN_ERC_Contracts/contracts/ERC20.sol: 372
- LGCN_VET_Contracts/contracts/ERC20.sol: 172

A few error messages between the different implementations of the token are disparate, which could make it difficult for dApps to automatically interpret these errors, since they would have to make different implementations for both networks.

Below, you can find certain examples of this issue:

- LGCN_ERC_Contracts/contracts/ERC20.sol: 272
  - `ERC20: transfer amount exceeds balance`
- LGCN_VET_Contracts/contracts/ERC20.sol: 264
  - `ERC20: amount exceeds balance`
- LGCN_ERC_Contracts/contracts/ERC20.sol: 322
  - `ERC20: burn amount exceeds balance`
- LGCN_VET_Contracts/contracts/ERC20.sol: 307
  - `ERC20: amount exceeds balance`

Recommendations

- Unify the logic between different networks.

Fixes Review

- The issue has been partially addressed in the last review, as some differences persist such as the `transferFrom` logic and different error messages.

Outdated third-party libraries

Identifier	Category	Risk	State
LGC-03	Outdated Software	**Informative**	**Assumed**

The smart contracts analyzed inherit functionalities from OpenZeppelin contracts that have been labeled obsolete and/or outdated; this does not imply a vulnerability by itself, because their logic does not present them, but it does imply that an update is not carried out by third party packages or libraries.

Currently the latest version of OpenZeppelin contracts is `4.7.0` but the **Legacy Network** project has implemented version `4.6.0` for `ERC20` contract and `3.3.0` for `VET` contract, which was released 2 years ago and is currently considered obsolete.

It would be convenient to include it as a reference instead of including the sources, in this way we will keep the development environment updated. By using the original sources, in case the project resolves any vulnerability or bug in the code, you would obtain this update automatically. Consequently, avoiding inheriting known vulnerabilities.

Recommendations

- We are aware that the implementation of a package manager may not be compatible for the `VeChain` contract, since there are changes needed in the `ERC20` in order to [make it compatible with VIP180, however it is recommended to at least implement it for the `ERC20` contract.

Incorrect pragma range

Identifier	Category	Risk	State
LGC-04	Configuration	**Informative**	**Fixed**

The **pragma** range estimated in the **VeChain** contracts (`>=0.6.0 <0.8.0`) is imprecise, since there are incompatibilities between versions **0.6.0** and **0.6.12** that prevent the correct compilation of the contracts.

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;
```

It has been verified that it is necessary to use at least the **0.7.0** compiler version to be able to compile the contract, since the visibility has not been established in the `constructor`, which is necessary for older versions of the compiler, the following versions are included from the **0.6.0** until the **0.6.12**.

```
contracts/LGCN.sol:254:5:
SyntaxError: No visibility
specified. Did you intend to add
"public"?
constructor() {
^ (Relevant source part starts here
and spans across multiple lines).
```

As can be verified in the Solidity documentation, from the 0.7.0 compiler version there is a breaking change related to this issue:

> `Visibility (public / external) is not needed for constructors anymore: To prevent a contract from being created, it can be marked abstract. This makes the visibility concept for constructors obsolete.`

Recommendations

- Update the pragma range estimated in `VeChain` contracts to at least `>=0.7.0 <0.8.0`.

References

- https://docs.soliditylang.org/en/v0.7.0/070-breaking-changes.html#functions-and-events

Source Code References

- LGCN_ERC_Contracts/contracts/ERC20.sol: 55
- LGCN_ERC_Contracts/contracts/LGCN.sol: 12

Fixes Review

- The issue has been addressed in the latest review.

Outdated compiler

Identifier	Category	Risk	State
LGC-05	Outdated Software	**Informative**	**Assumed**

Solc frequently launches new versions of the compiler. Using an outdated version of the compiler can be problematic, especially if there are errors that have been made public or known vulnerabilities that affect this version.

We have detected that the audited contract uses the following version of Solidity pragma:

- **ERC20:**
  - `^0.8.0`
- **VET:**
  - `>=0.6.0 <0.8.0;`

```
//VET
pragma solidity >=0.6.0 <0.8.0;

//ERC20
pragma solidity ^0.8.0;
```

Recommendations

- Solidity branch **0.8 has important bug fixes** related to the immutable state and treatment of arrays, so it is recommended to use the most up to date version of the pragma.
- Regarding the **0.6.X** version of **Solc** is affected by different known bugs that have already been fixed in later versions. It is always a good policy to use the most up to date version of the *pragma*.

References

- https://github.com/ethereum/solidity/blob/develop/Changelog.md

GAS optimization

Identifier	Category	Risk	State
LGC-06	Codebase Quality	**Informative**	**Partially Fixed**

Software optimization is the process of modifying a software system to make an aspect of it work more efficiently or use less resources. This premise must be applied to smart contracts as well, so that they execute faster or in order to save GAS.

On Ethereum blockchain, GAS is an execution fee which is used to compensate miners for the computational resources required to power smart contracts. If the network usage is increasing, so will the value of GAS optimization.

These are some of the requirements that must be met to reduce GAS consumption:

- Short-circuiting.
- Remove redundant or dead code.
- Delete unnecessary libraries.
- Explicit function visibility.
- Use of proper data types.
- Use hard-coded CONSTANT instead of state variables.
- Avoid expensive operations in a loop.
- Pay special attention to arithmetical operations and comparisons.

Dead code

In programming, a part of the source code that is never used is known as dead code. The execution of this type of code consumes more GAS during deployment in something that is not necessary.

The following methods can be removed:

- `_msgData`:
  - LGCN_ERC_Contracts/contracts/Context.sol: 21
  - LGCN_VET_Contracts/contracts/Context.sol: 19
- `_burn`:
  - LGCN_ERC_Contracts/contracts/ERC20.sol: 316
  - LGCN_VET_Contracts/contracts/ERC20.sol: 300
- `_setupDecimals`:
  - LGCN_VET_Contracts/contracts/ERC20.sol: 345

Storage optimization

The use of the `immutable` (*https://docs.soliditylang.org/en/v0.8.0/contracts.html#immutable*) keyword is recommended to obtain less expensive executions, by having the same behaviour as a constant. However, by defining its value in the constructor we have a significant save of GAS.

This behavior has been observed in:

- `_name` and `_symbol`:
  - LGCN_ERC_Contracts/contracts/ERC20.sol: 42-43
- `_decimals` as long as the `_setupDecimals` method is removed, which is not used:
  - LGCN_VET_Contracts/contracts/ERC20.sol: 44

Use custom errors instead of require

Solidity **0.8.4** introduced custom errors, a more efficient way to notify users of an operation failure.

It has been verified that the different contracts are managing the error messages through the `require` command, this instruction is more expensive than the use of custom errors because it requires placing the error message on the stack, whether or not it is reverted the transaction, in addition to this, is more susceptible to using long error messages that will produce a higher cost of gas, whatever the result of the condition.

Custom errors are more gas efficient than revert strings in terms of deployment and runtime cost when the revert condition is met. In order to save gas it is recommended use custom errors instead of revert strings.

The new custom errors are defined through the `error` statement, and they can also receive arguments. As indicated in the following example:

```
error Unauthorized();
error InsufficientPrice(uint256 available, uint256 required);
```

Afterwards, it is enough to just call them when needed using `if` conditionals:

```
if (msg.sender != owner) revert Unauthorized();
if (amount > balance[msg.sender]) {
            revert InsufficientPrice({
                available: balance[msg.sender],
                required: amount
           });
}
```

This behavior has been observed in:

- LGCN_ERC_Contracts/contracts/ERC20.sol
- LGCN_VET_Contracts/contracts/ERC20.sol
- LGCN_VET_Contracts/contracts/LGCN.sol
- LGCN_VET_Contracts/contracts/SafeMath.sol

References

- https://blog.soliditylang.org/2021/04/21/custom-errors

Unnecessary import

The `LGCN` contract imports the *builtin for VeChain* library to access native functions of the blockchain, but apparently the token does not use these functionalities.

As can be seen in the following image, the import of the `Builtin` library is executed, however, it is not necessary since said library only contains interfaces, and they are not used throughout the execution flow of the `LGCN` contract.

```
import "./ERC20.sol";
import "./builtin.sol";

contract LGCN is ERC20 {
```

References

- https://github.com/vechain/thor-builtins

This behavior has been observed in:

- LGCN_VET_Contracts/contracts/LGCN.sol: 6

Wrong visibility

Considering that the methods `name`, `decimals` and `symbol` are constants and do not access the storage information (as long as the changes mentioned above are made), it is convenient to mark it as `pure` instead of `view`.

This behavior has been observed in:

- `name()`, `symbol()`, `decimals()`:
  - LGCN_ERC_Contracts/contracts/ERC20.sol: 62, 70, 87
- `decimals()`:
  - LGCN_VET_Contracts/contracts/ERC20.sol: 87

Unnecessary variable declaration

In certain `ERC20` methods, the value of `_msgSender` is unnecessarily cached in a variable, by directly using the return method it is possible to save gas by avoiding the declaration of a variable.

```
function transfer(address to, uint256 amount)
    public
    virtual
    override
    returns (bool)
{
    address owner = _msgSender();
    _transfer(owner, to, amount);
    return true;
}
```

This behavior has been observed in:

- LGCN_ERC_Contracts/contracts/ERC20.sol: 125, 159, 185

Fixes Review

- The issue has been partially addressed in the last review, as most of the optimizations has been applied.

Solidity literals

Identifier	Category	Risk	State
LGC-07	Codebase Quality	**Informative**	**Fixed**

In order to make the code easier to read and to minimize human errors, Solidity recommends the use of literals which consequently makes it more user friendly.

A literal number can take a suffix of `wei`, `gwei` or `ether` to specify a subdenomination of `ether`, where `ether` numbers without a postfix are assumed to be `wei`.
As can be seen, in the `ERC20` contract the number of tokens to be minted is calculated through an arithmetic operation.

```
constructor() ERC20("LEGACY TOKEN", "LGCN") {
    _mint(msg.sender, 720000000 * 10**decimals());
}
```

Since the value returned by `decimals()` is always *18*, you can save gas and unify cross-chain logic by using solidity literals, as it is done in the *VeChain* contract.

```
constructor(address wallet) {
    require(wallet != address(0), "Can not be zero wallet");
    _mint(wallet, 720000000 ether);
}
```

Recommendations

- Use literals instead of arithmetic operations when possible.

References

- https://docs.soliditylang.org/en/latest/units-and-global-variables.html#units-and-globally-available-variables

Source Code References

- LGCN_ERC_Contracts/contracts/LGCN.sol: 9
- LGCN_ERC_Contracts/contracts/ERC20.sol: 88

Fixes Review

- The issue has been addressed in the latest review.

Annexes

Annex A – Vulnerabilities Severity

Red4Sec determines the vulnerabilities severity in the following levels of risk according to the impact level defined by CVSS v3 (Common Vulnerability Scoring System) by the National Institute of Standards and Technology (NIST):

Vulnerability Severity

The risk classification has been made on the following 5 value scale:

Severity	Description
Critical	Vulnerabilities that possess the highest impact over the systems, services and/or sensitive information. The existence of these vulnerabilities is dangerous and should be fixed as soon as possible.
High	Vulnerabilities that could compromise severely compromise the service or the information it manages even if the vulnerability requires expertise to be exploited.
Medium	Vulnerabilities that on their own can have a limited impact and/or that combined with other vulnerabilities could have a greater impact.
Low	These vulnerabilities do not suppose a real risk for the systems. Also includes vulnerabilities which are extremely hard to exploit or whose impact on the service is low.
Informative	It covers various characteristics, information or behaviours that can be considered as inappropriate, without being considered as vulnerabilities by themselves.

Invest in Security, invest in your future