WHITE HAT CYBER

Bridge Contract Audit
Code Review Assessment

TLC & LSO SideBridge

Date: 27.05.2022
Version : 1.0

Presented by:
White Hat Cyber

For Public Distribution

DOCUMENT PROPERTIES

Version: 1.0
File Name: Report_TLC_LSO_Side_Bridge.docx
Publication Date: 27.05.2022
Confidentiality Level: For Public Distribution
Document Recipient: The Luxury Network
Document Status: Approved

EXECUTIVE SUMMARY

The assessment was conducted remotely by the White Hat Cyber.

1. To help the Client to better understand its security posture
2. To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place
3. To identify potential issues and include improvement recommendations

This report summaries the tests performed and findings in terms of strengths and weaknesses. It also contains detailed descriptions of any discovered vulnerabilities, steps the White Hat Cyber Team took to exploit each vulnerability, and recommendations for re-mediation.

1.1 Engagement Limitations

The architecture and code review are based on the documentation and code provided by The Luxury Network. The code resides in a public repository at https://github.com/TlChainNetwork/tlc-side-bridge

Token-bridge:

0xbBcdB8C8E0D17b76F5ED2B9FC36A60154c2142F2 - BSC(TLC)
0x160d97B05a7E7D249f8FcEb47b2A35170f3eF949 - AVAX(TLC)
0xd3f978dc308C0441A435bE8D67b15Ec2cFF7776f - FTM(TLC)
0x25C6330D43445985f564285f3735600c8A94b74a - BSC(LSO)
0xE79f6aefB16AdeFF38dAA435372a7A166815685D - AVAX(LSO)
0xBb4b426deA349e1F7283ee36dbbb77B1Eb6DB1bF - FTM(LSO)

1.2 Engagement Analysis

The code review was conducted by the White Hat Cyber team on the code provided by The Luxury Network, in the form of a Github repository. The code review focused on the handling of secure and private information handling in the code.

As a result of our work, we identified 0 High, 0 Medium, 4 Low, and 0 Informational findings.

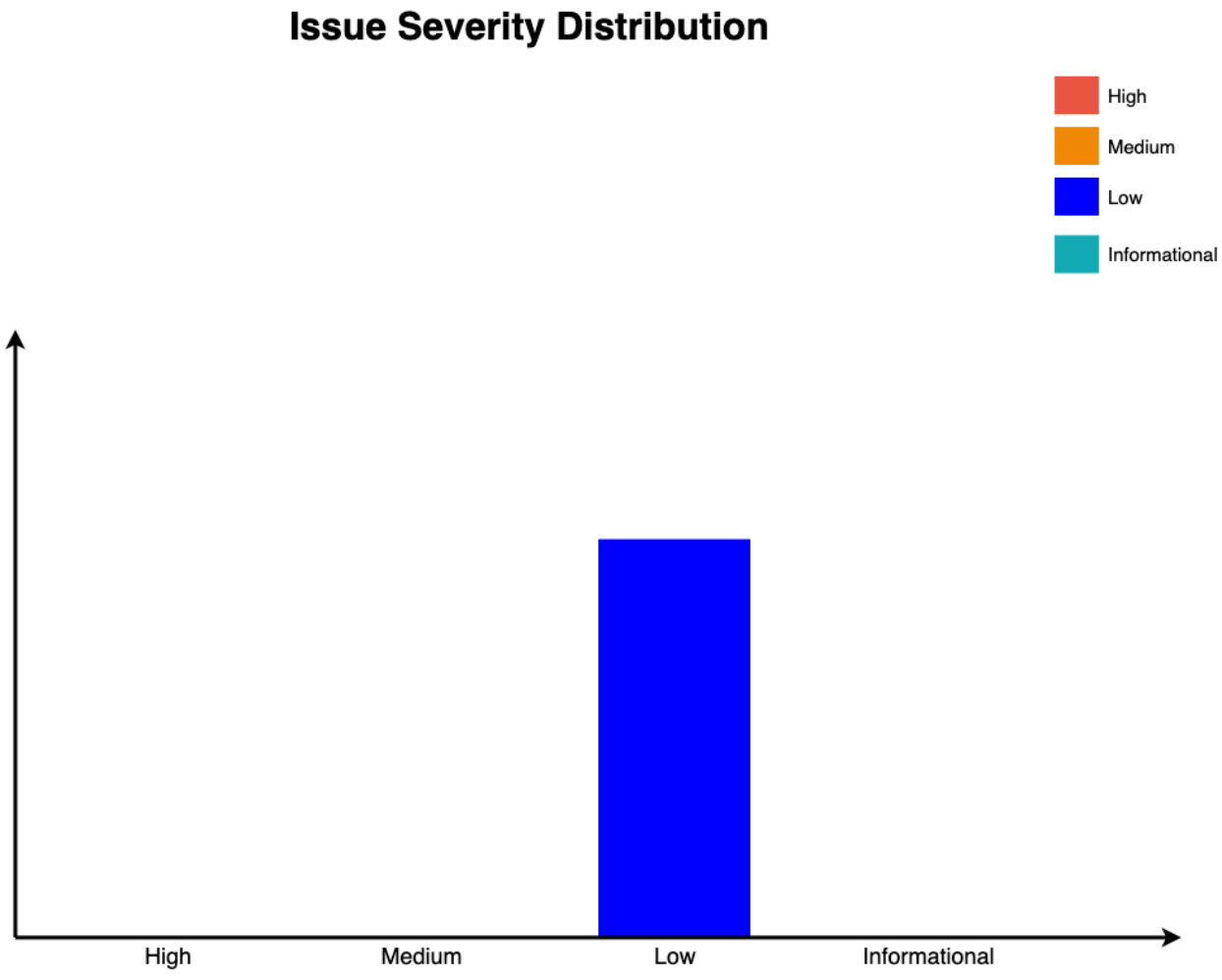

Figure 1 Issue Severity Distribution

1.3 Observations

The code is generally well written and for the most part documented. This facilitates reading of the execution flow. It is worth to mention that there are plenty of hardcoded values that should be re-written as constants. This is only an issue during versioning of the contract if these values need to be changed, versioned, or externalised.

As with any blockchain program, it is important to do error checking to ensure that the data that you pass into the program is what you intend, because programs will gladly attempt to execute transfers between incorrect wallets or with incorrect seeds. It is important that when using this contract, that all documentation is followed and the “caller” ensures that this contract is appropriate for their use case.

1.4 Issue Summary List

ID	SEVERITY	FINDING
SIDE-BRIDGE-01	**LOW**	**A call to a user-supplied address is executed.**
SIDE-BRIDGE-02	**LOW**	**State variable visibility is not set.**
SIDE-BRIDGE-03	**LOW**	**Multiple calls are executed in the same transaction.**
SIDE-BRIDGE-04	**LOW**	**Requirement violation.**

2. METHODOLOGY

Static Analysis, Manual Review, DevNet deployment

2.1 Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyse the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and tools, utilising the experience of the reviewer. No dynamic testing was performed, only the use of custom built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analysed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behaviour

- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general list and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analysed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

3. TECHNICAL DETAILS

3.1 A call to a user-supplied address is executed.

Finding ID: SIDE-BRIDGE-01

Severity: LOW

Status: Acknowledged

```
52
53  function bridgeTokens (address _requester, uint _bridgedAmount) verifyInitialization onlyGateway external returns (bool){
54  sideToken.mint(_requester,_bridgedAmount);
55  return true;
56  }
```

The resulting error is only printed and forwarded to the calling method without any further handling.

3.2 State variable visibility is not set.

Finding ID: **SIDE-BRIDGE-02**

Severity: **LOW**

Status: **Acknowledged**

```
6  contract SideBridge {
7  IERC20Child private sideToken;
8  bool bridgeInitState;
9  address owner;
10 address gateway;
```

It is best practice to set the visibility of state variables explicitly. The default visibility for "bridgeInitState" is internal. Other possible visibility settings are public and private.

3.3 Multiple calls are executed in the same transaction.

Finding ID: **SIDE-BRIDGE-03**

Severity: **LOW**

Status: **Acknowledged**

```
40 require(_bridgedAmount>commission,"You cannot send less than comisision");
41 require(_bridgedAmount <= sideToken.balanceOf(requester),"Check to have enough tokens (including the commision)");
42 sideToken.burnFrom(requester, _bridgedAmount);
43 emit TokensReturned(requester, _bridgedAmount, _addressTo, block.timestamp);
44 return true;
```

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such

that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

3.4 Requirement violation.

Finding ID: **SIDE-BRIDGE-04**

Severity: **LOW**

Status: **Acknowledged**

```
39  address requester = msg.sender;
40  require(_bridgedAmount>commission,"You cannot send less than comisision");
41  require(_bridgedAmount <= sideToken.balanceOf(requester),"Check to have enough tokens (including the comision)");
42  sideToken.burnFrom(requester, _bridgedAmount);
43  emit TokensReturned(requester, _bridgedAmount, _addressTo, block.timestamp);
```

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

SEVERITY RATING DEFINITIONS

SEVERITY	DEFINITION
High	The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
Medium	The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
Low	The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
Informational	Informational findings are best practice steps that can be used to harden the application and improve processes.