SMART CONTRACT SECURITY AUDIT

GraphLinq

Jorge Rodriguez

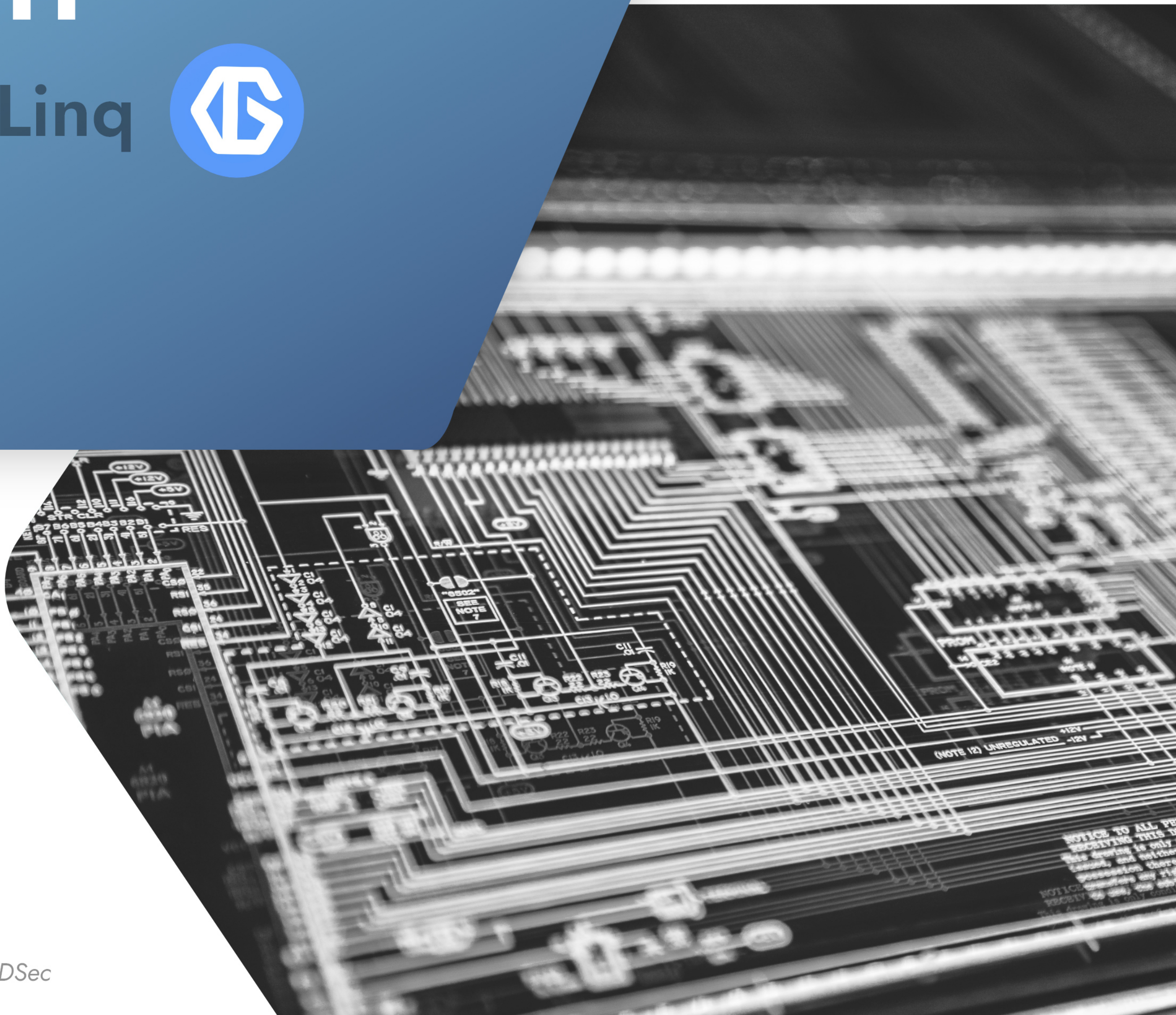


Table of Contents

1.0 Introduction

1.1 Project engagement

During March of 2021, GraphLinq engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. GraphLinq provided CTDSec with access to their code repository and whitepaper.

Graphlinq is a tool that offers an interface providing users the possibility of generating and managing their own blockchain data without any past coding experience.

1.2 Disclaimer

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that’s able to withstand the network’s fast-paced and rapidly changing environment, we at CTDSec recommend that GraphLinq team put in place a bug bounty program to encourage further and active analysis of the smart contract.

2.0 Coverage

2.1 Target Code and Revision

For this audit, we performed research, investigation, and review of the GraphLinq contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:

Source:

https://github.com/GraphLinq/GraphLinq/blob/master/NodeBlock.GraphLinqPrivateSaleContract/contracts/GraphLinqPrivateSale.sol

https://github.com/GraphLinq/GraphLinq/blob/master/NodeBlock.EngineDepositorContract/contracts/GraphLinqDepositor.sol

File:

GraphLinqDepositor.sol - MD5: 066e4c4ef221516177900c81d45877c6

GraphLinqPrivateSale.sol - MD5: 429C395F3AFF5EC0346FFE1094C9E65A

2.2 Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

№	Issue description.	Checking status
1	Compiler warnings.	PASSED
2	Race conditions and Reentrancy. Cross-function race conditions.	PASSED
3	Possible delays in data delivery.	PASSED
4	Oracle calls.	PASSED
5	Front running.	PASSED
6	Timestamp dependence.	PASSED
7	Integer Overflow and Underflow.	PASSED
8	DoS with Revert.	PASSED
9	DoS with block gas limit.	PASSED
10	Methods execution permissions.	PASSED
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	PASSED
12	The impact of the exchange rate on the logic.	PASSED
13	Private user data leaks.	PASSED
14	Malicious Event log.	PASSED
15	Scoping and Declarations.	PASSED

16	Uninitialized storage pointers.	PASSED
17	Arithmetic accuracy.	PASSED
18	Design Logic.	PASSED
19	Cross-function race conditions.	PASSED
20	Safe Zeppelin module.	PASSED
21	Fallback function security.	PASSED

3.0 Security Issues

3.1 High severity issues [0]

No high severity issues found.

3.2 Medium severity issues [1]

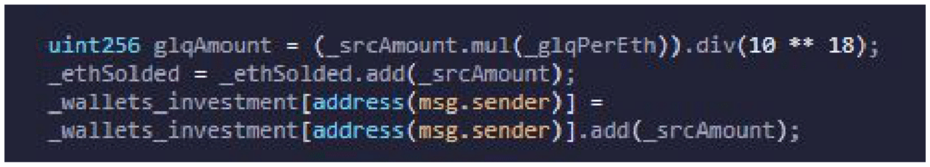
<u>**Funds losing possible:**</u>

Issue:

If someone send less than 10 ** 18 ETH to the contract, he will not receive any tokens, but will spend his money.

Recommendation:

Better to multiply first and only then to divide. In function _transfertGLQ. And we recommend using safe math for all operations. Like this:



```
uint256 glqAmount = (_srcAmount.mul(_glqPerEth)).div(10 ** 18);
_ethSolded = _ethSolded.add(_srcAmount);
_wallets_investment[address(msg.sender)] =
_wallets_investment[address(msg.sender)].add(_srcAmount);
```

File:

GraphLinqPrivateSale.sol

3.3 Low severity issues [1]

Zero Address Checking:

Issue:

There is no zero address checking in the constructor of the contract.

Recommendation:

Add zero address checking.

File:

GraphLinqDepositor.sol

4.0 Summary of the audit

After working with the development teams and reviewing the problems, the issues presented in the report have been resolved.