



SOLIDProof
Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY

GraphLinq Audit

**Security Assessment
14. February, 2023**

For

 **graphlinqchain**



SolidProof_io



@solidproof_io

Disclaimer	3
Description	5
Project Engagement	5
Logo	5
Contract Link	5
Methodology	7
Used Code from other Frameworks/Smart Contracts (direct imports)	8
Tested Contract Files	9
Source Lines	10
Risk Level	10
Capabilities	11
Inheritance Graph	12
CallGraph	13
Scope of Work/Verify Claims	14
Modifiers and public functions	17
Source Units in Scope	18
Alleviation:	18
Critical issues	19
High issues	19
Medium issues	19
Low issues	19
Informational issues	19
Audit Comments	19
SWC Attacks	21

Disclaimer

SolidProof.io reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Uniswap, Uniswap, PancakeSwap etc’...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug- free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof’s position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	9. February 2023	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary
1.1	14. February 2023	<ul style="list-style-type: none">• Reaudit

Network

Ethereum

Website

<https://graphlinq.io>

Twitter

https://twitter.com/graphlinq_proto



Description

With GraphLinq you can generate a set of nodes (blocks) that receive an input and output to a single/multiple other nodes, so you create with a set of tools your 'structure' of code with a path of execution that will be launched on the blockchain or the GraphLinq Engine, then you can deploy it on the test net Engine or the main net Engine, once you tested and want to get in production. One graph can for example track network pairs activities on Binance and report stats to webhook, or slacks, discord, telegram, twitter with any conditions you decide to trigger a possible results of your nodes.

Project Engagement

During the Date of 9 February 2023, **GraphLinq Team** engaged Solidproof.io 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. They provided Solidproof.io with access to their code repository and whitepaper.

Logo



Contract Link

v1.0

- <https://github.com/GraphLinq/GraphLinq.Bridge>
- Commit: f445ef8af3f049b05d1361ec14643c490ac34ad4

v1.1

- <https://github.com/GraphLinq/GraphLinq.Bridge>
- Commit: 9b1a70ede5e6dd2ba3702dd3c77a445e0fb5d42d

Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 - 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 - 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 - 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 - 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - i) Review of the specifications, sources, and instructions provided to SolidProof to make sure we understand the size, scope, and functionality of the smart contract.
 - ii) Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - iii) Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to SolidProof describe.
2. Testing and automated analysis that includes the following:
 - i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
 - ii) Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

Used Code from other Frameworks/Smart Contracts (direct imports)

Imported packages:

```
./interfaces/IERC20.sol  
./libs/SafeMath.sol
```



Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

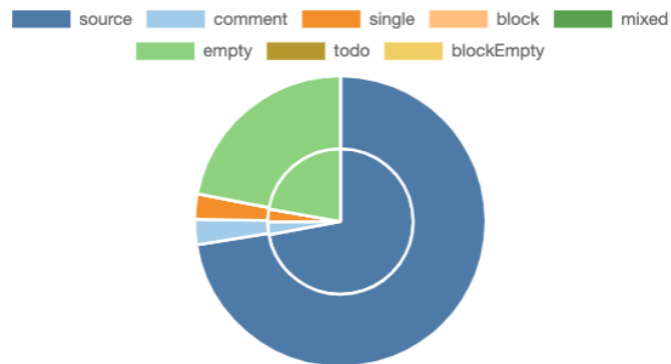
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.1

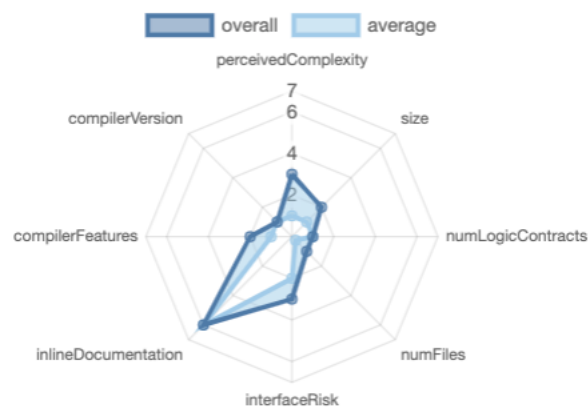
File Name	SHA-1 Hash
contracts/ GraphLinqBridgeIn.sol	0cdc78ef5e912c6a755068f2875b267 6214d5b19
contracts/ GraphLinqBridgeOut.sol	ff4df6dac6ac3ca71112651a598fc08dc a565321
contracts/ GraphLinqBridgeInNative.sol	bc6d685f14018c3ce5f7657fc52afc093 20debef
contracts/ GraphLinqBridgeOutNative.sol	f14b82e7d6d9a76d2a43b61dfc710cb 2bd6df4e0

Metrics

Source Lines v1.1



Risk Level v1.1




Capabilities

Components

 Contracts	 Libraries	 Interfaces	 Abstract
4	0	0	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.












 Public	 Payable
25	2

External	Internal	Private	Pure	View
2	18	2	0	11

StateVariables

Total	 Public
18	11

Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
<div>^0.8.0</div>		<div>yes</div>	<div></div>	<div></div>	
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECRrecover	 New/Create/Create2
<div>yes</div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>
 TryCatch	<div>Σ Unchecked</div>				
<div></div>	<div></div>				

Inheritance Graph

v1.1

GraphlinqBridgeOutNative

GraphlinqBridgeIn

GraphLinqBridgeOut

GraphlinqBridgeInNative



Scope of Work/Verify Claims

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Is contract an upgradeable
2. Overall checkup (Smart Contract Security)



Is contract an upgradeable

Name	
Is contract an upgradeable?	No



Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Write functions of contract v1.1

- ◆ setOneBridged
- ◆ setMultipleBridged
- ◆ withdrawGlq
- ◆ withdrawBridged

- ◆ setBridgeOutOperator
- ◆ bridgeGlq
- ◆ withdrawGlq

Modifiers and public functions v1.1

◆ `setBridgeOutOperator`
◆ `bridgeGlq`
◆ `withdrawGlq`

◆ `setOneBridged`
◆ `setMultipleBridged`
◆ `withdrawGlq`
◆ `withdrawBridged`

Ownership Privileges:

- Only the Deployer can withdraw GLQ from the “GraphLinqBridgeIn” contract, and set the bridge out operator address.
- **GraphLinqBridgeOut:**
 - Deployer can set bridged token amount in the owner’s address or any other address with any arbitrary value.
- **GraphLinqBridgeInNative:**
 - Withdraw GLQ from the contract.
- **GraphLinqBridgeOutNative:**
 - Deployer can set bridged token amount in the owner’s address or any other address with any arbitrary value.

Please check if an `OnlyOwner` or similar restrictive modifier has been forgotten.

Source Units in Scope v1.1

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/GraphLinqBridgeOutNative.sol	1	————	78	76	56	2	44
contracts/GraphLinqBridgeIn.sol	1	————	81	81	60	3	49
contracts/GraphLinqBridgeOut.sol	1	————	87	85	64	2	54
contracts/GraphLinqBridgeInNative.sol	1	————	64	64	47	2	32
Totals	4	————	310	306	227	9	179

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalised lines of the source unit (e.g. normalises functions spanning multiple lines)
nSLOC	normalised source lines of code (only source-code lines; no comments, no blank lines)
Comment Lines	lines containing single or block comments
Complexity Score	a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Alleviation:

- KryxivaaBridgeOut.sol, GraphLinqCoin.sol, and KryxivaaCoin.sol was removed from the scope in the reaudit in v1.1

Audit Results

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

Issue	File	Type	Line	Description
#1	All	Contract doesn't import npm packages from source (like OpenZeppelin etc.)	—	We recommend to import all packages from npm directly without flatten the contract. Functions could be modified or can be susceptible to vulnerabilities
#2	All	A floating pragma is set	—	The current pragma Solidity directive is „^0.8.0”.
#3	All Bridge Out Contracts	Missing Zero Address Validation (missing-zero-check)	25, 39	Check that the address is not zero

Informational issues

Issue	File	Type	Line	Description
#1	Main	NatSpec documentation missing	—	If you started to comment your code, also comment all other functions, variables etc.

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information <https://docs.soliditylang.org/en/latest/natspec-format.html>) for your contracts to provide rich

documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

14. February 2023:

- There is still an owner (Owner still has not renounced ownership)
- Read whole report and modifiers section for more information



SWC Attacks

ID	Title	Relationships	Status
SW C-1 36	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SW C-1 35	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SW C-1 34	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SW C-1 33	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SW C-1 32	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SW C-1 31	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SW C-1 30	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SW C-1 29	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SW C-1 28	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SW C-1 27	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SW C-1 25	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SW C-1 24	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SW C-1 23	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SW C-1 22	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SW C-1 21	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SW C-1 20	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SW C-11 9	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-11 8	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SW C-11 7	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SW C-11 6	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 5	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SW C-11 4	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SW C-11 3	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SW C-11 2	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 1	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SW C-11 0	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SW C-1 09	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-1 08	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-1 07	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-1 06	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SW C-1 05	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SW C-1 04	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SW C-1 03	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	PASSED
SW C-1 02	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SW C-1 01	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SW C-1 00	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

*Solid
Proofed*

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**


MADE IN GERMANY