LightLink

ARCHITECTURE & CODEBASE SECURITY AUDIT

12.01.2023

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer ... 3
2. About the Project and Company ... 4
   2.1 Project Overview ... 5
3. Vulnerability & Risk Level ... 6
4. Auditing Strategy and Techniques Applied ... 7
   4.1 Methodology ... 7
5. Metrics ... 8
   5.1 Tested Files ... 8
   5.2 Used Packages ... 13
   5.3 CallGraph (__MAIN__) ... 14
   5.4 Codebase Overview ... 15
6. Scope of Work ... 16
   6.1 Findings Overview ... 17
   6.2 Manual and Automated Vulnerability Test ... 18
   6.2.1 Spelling Errors ... 19
   6.2.2 Single nested if ... 21
   6.2.3 Insufficient Documentation ... 22
   6.3 Common Blockchain Vulnerabilities ... 23
   6.4 Test Coverage ... 31
7. Executive Summary ... 37
8. About the Auditor ... 38

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Pellar Technology Pty Ltd. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (13.12.2022)	Layout
0.2 (14.12.2022)	Test Deployment
0.5 (20.12.2022)	Automated Security Testing Manual Security Testing
0.6 (21.12.2022)	Development Productivity (Code Conventions Check, packages)
0.7 (22.12.2022)	Performance (Connection pooling, caching, Transaction Concurrency)
0.8 (23.12.2022)	Maintainability & Ease of Deployment
0.9 (24.12.2022)	Summary and Recommendation
1.0 (25.12.2022)	Final document
1.1 (12.01.2023)	Re-check 45d2c427720c548fba6da22a34c1540dcebbcb9b

2. About the Project and Company

Company address:
Pellar Technology Pty Ltd
Level 3, 162 Collins St
Melbourne VIC 3000
Australia

Website: https://lightlink.io

Twitter: https://twitter.com/LightLinkChain

Documentation: https://pellartech.github.io/lightlink-chain-docs

Telegram: https://t.me/lightlinkLL

2.1 Project Overview

The LightLink Team aims to create a lightweight Ethereum layer 2 scaling solution by building a new kind of Optimistic Rollup. LightLink utilizes Ethereum's popular go-ethereum (Geth) codebase. This means that LightLink can leverage the existing Ethereum ecosystem of tools and libraries.

LightLink uses the Ethereum Virtual Machine (EVM) to execute transactions and interact with smart contracts. Account state on LightLink is stored the same way as Ethereum using the Merkle Patrice tree data structure. LightLink transactions have the same format as Ethereum, currently the legacy transaction format is used with plans to integrate more transaction types in the future. Users can interact with a LightLink node via JSON RPC, a light weight remote procedure call protocol familiar to Ethereum users. Users can also opt to utilize popular web3 libraries such as ethers.js to interact with LightLink programmatically. This means that LightLink can be used as a drop-in replacement for Ethereum from an end users perspective.

This allows LightLink to process transactions off-chain much faster than Ethereum. LightLink can process up to 5,712 transactions per second compared to Ethereum's 15 transactions per second. Unlike Ethereum, LightLink has a very low transaction fee, averaging at around $0.01.

Organizations can pay a monthly fee to simplify user experiences when transacting using ERC20 and ERC721 smart contract standards, this mode can bypass native gas costs.

3. 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 code functioning in a number of scenarios or creates a risk that the code may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a codebase, or provides the opportunity to use an application 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 code 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 code 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

4. 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 pen testers and developers, documenting any issues as there were discovered.

4.1 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 Chainsulting to make sure we understand the size, scope, and functionality of the codebase.
   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 Chainsulting 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 codebase 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 codebase.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	code	comment	blank	total	Fingerprint (MD5)
pellar-prime-develop/api/api.go	25	4	5	34	61b64666891416e296677cf153d113b6
pellar-prime-develop/api/middleware.go	14	2	3	19	7934d01be94068c5856e73f313a82089
pellar-prime-develop/api/organisation.go	131	20	27	178	6581268b64dd605cbb6eff0cc1bc768a
pellar-prime-develop/api/requests.go	15	0	5	20	7d23168d5bed9de9c29440a48af4d22f
pellar-prime-develop/api/routes.go	12	55	4	71	74540b693ca04c29f28f86e1eb6d334c
pellar-prime-develop/api/server.go	37	7	8	52	97e188d5c886be957e094be6868c5d56
pellar-prime-develop/api/validator.go	51	17	15	83	78567b5608ed07a6dac5c2bcf15e6ba4
pellar-prime-develop/cmd/prime/cmd/config.go	21	17	10	48	c52d27685ce12561b9a5fe9abb8943ec
pellar-prime-develop/cmd/prime/cmd/genesis.go	32	17	10	59	bc4ca4beba993e9c20fd2d329c3fca8e
pellar-prime-develop/cmd/prime/cmd/keygen.go	28	17	10	55	73ed7bc686c795f1536cd24499dae8d3

pellar-prime-develop/cmd/prime/cmd/peers.go	21	17	10	48	b4290710aef1cf33e0b03b7ff4ccbd4
pellar-prime-develop/cmd/prime/cmd/replicator.go	84	26	22	132	032034168419d48f9706e4ad14dfe71a
pellar-prime-develop/cmd/prime/cmd/root.go	27	18	12	57	20082b8707c32b39fce7b0c524337930
pellar-prime-develop/cmd/prime/cmd/sequencer.go	90	27	23	140	597eef6578c970a78f2f8d96a0177053
pellar-prime-develop/cmd/prime/cmd/version.go	16	17	10	43	d0bf14db8796f836261445364b38653d
pellar-prime-develop/cmd/prime/main.go	7	17	8	32	708412539f07d9ec8bf6fd38f76af64e
pellar-prime-develop/config/config.go	118	19	17	154	5511777ef94e9c10de9b43efd0c88f23
pellar-prime-develop/core/account.go	45	22	15	82	c9a8a3bc1f7207782279b594be9d124a
pellar-prime-develop/core/account_test.go	31	4	9	44	6666b327b6af05e1386fbb0cf488bd2a
pellar-prime-develop/core/block.go	196	56	35	287	aa4fb6671ad416c3ccd955c0be3e75be
pellar-prime-develop/core/block_test.go	307	41	56	404	72348f1e0963acc3f24dcab255894285
pellar-prime-develop/core/blockchain.go	356	112	90	558	68565af52f9672872c42047a267125fa
pellar-prime-develop/core/blockchain_test.go	372	136	113	621	5299e9f2c4d4627aa64fedd45585e359
pellar-prime-develop/core/evm.go	145	42	25	212	708b01f6b1cd36378f6a4dfd9d544f48
pellar-prime-develop/core/evm_test.go	15	19	10	44	b5d2d4d57775f52a43b2203b7e8df63e
pellar-prime-develop/core/gas.go	45	21	17	83	9c7b8aad41d68d10ac7c8d6e81e672f2
pellar-prime-develop/core/gas_test.go	51	27	18	96	7d62a08544454addc99ef487c8df0af2
pellar-prime-develop/core/genesis.go	46	23	19	88	738fbb87790d6db5375e677965f27962
pellar-prime-develop/core/genesis_test.go	10	21	10	41	33e8bdfac60ba305e4a2996131f843a2

pellar-prime-develop/core/memory_pool.go	224	79	44	347	8150f83c4ebf5e4080ad575ae6f12626
pellar-prime-develop/core/memory_pool_test.go	679	217	217	1,113	552bfb31f127f786bc90ffadb4f80fdc
pellar-prime-develop/core/miner.go	106	45	38	189	7f5bac92c494a0de6c6f05cec87cb077
pellar-prime-develop/core/miner_test.go	85	29	37	151	7d5c2a4d8afaaac84c71e9a2f029c3f7
pellar-prime-develop/core/organisation.go	39	25	13	77	37ee5c37e0d10bd1371c33dd8975f3f1
pellar-prime-develop/core/organisation_test.go	29	21	14	64	b7a94a62674a78c96d730d6d2064842b
pellar-prime-develop/core/receipt.go	58	27	14	99	add22e2d3053bfdbb2e19842ee1fc731
pellar-prime-develop/core/receipt_test.go	27	22	14	63	43e5e28c9d4ae961dd73192e49f89b33
pellar-prime-develop/core/storage.go	247	65	45	357	a1329bb75a9bd1c7ee486445adfe7207
pellar-prime-develop/core/storage_test.go	207	120	71	398	ea47a762054616f65f9d68d7db2d5bc5
pellar-prime-develop/core/sync_block_queue.go	70	32	22	124	bbf059464b250ffbae195e4a3ee4a948
pellar-prime-develop/core/sync_block_queue_test.go	19	26	13	58	efe51bdca4a877ee136a69300b455a42
pellar-prime-develop/core/tracer.go	132	29	18	179	56bd92a7effca5e5e0fb522cf58f1b13
pellar-prime-develop/core/tracer_test.go	4	18	9	31	91ea935f4672766b775227caf69537ca
pellar-prime-develop/core/transaction.go	98	26	21	145	47b4ee879ac69d2cf4a8a7cb91a6445d
pellar-prime-develop/core/transaction_test.go	64	29	24	117	6198022bb60623869c918f323bd7759e
pellar-prime-develop/logger/logger.go	39	18	16	73	945ad10d98bbde40c106637482a9076a
pellar-prime-develop/networking/message.go	185	48	38	271	f7b7f64b2ff47fa0991572d501802118
pellar-prime-develop/networking/message_handlers.go	409	93	81	583	a4f9b0b3113d63c590f6b61b84070832

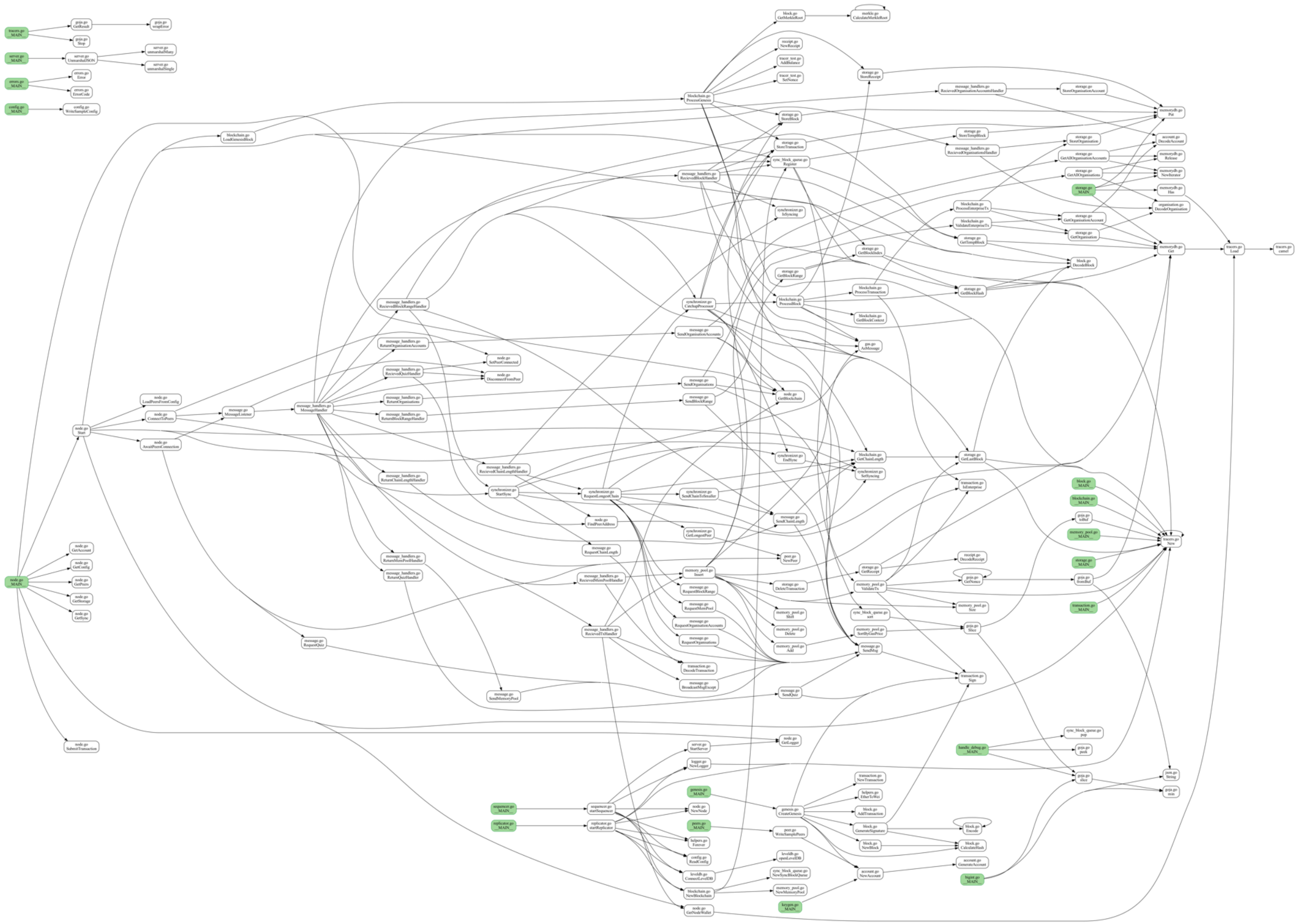
pellar-prime-develop/networking/node.go	205	45	52	302	a1718c7818b7ddd3a952da79ad18198c
pellar-prime-develop/networking/peer.go	49	22	16	87	8d0451868cc3c2d5ea3b4df2d5b9f49e
pellar-prime-develop/networking/synchronizer.go	151	33	34	218	3f41e7e9295f628ef0565899d369d200
pellar-prime-develop/replicator/replicator.go	40	23	17	80	dcb814915cda16e39781f41cab856dab
pellar-prime-develop/rpc/errors.go	41	24	30	95	4cc07e044db7578ed4e3ce28bbe191a1
pellar-prime-develop/rpc/errors_test.go	61	5	5	71	ecb27a1a53e521685a49ce656d0dda38
pellar-prime-develop/rpc/handle_debug.go	382	33	99	514	feb876046a8f82031d6415ada891f946
pellar-prime-develop/rpc/handle_eth.go	632	67	120	819	b5336f2b04489afe89a9986f1be97736
pellar-prime-develop/rpc/handle_eth_test.go	202	10	13	225	23f6f36733bc8d266b6bf9934cc426dc
pellar-prime-develop/rpc/handle_net.go	15	20	11	46	b1f3238df255e44d16eb0a7e9508cbe4
pellar-prime-develop/rpc/handle_net_test.go	133	9	10	152	aa50ac1155c9b8ac13d420b3467c652b
pellar-prime-develop/rpc/handle_prime.go	109	23	31	163	a1252a94528bcc54a388158266ebddb2
pellar-prime-develop/rpc/handle_prime_test.go	34	2	5	41	adfba46411486d76ee01dda00584f748
pellar-prime-develop/rpc/handle_txpool.go	4	18	7	29	00e98793e6c389c1acacb4da26235b53
pellar-prime-develop/rpc/handle_web3.go	18	21	10	49	b97f71462d815f96e7f24a2bd8975c32
pellar-prime-develop/rpc/handle_web3_test.go	74	4	7	85	7546f346734461b107a518226a09bcb5
pellar-prime-develop/rpc/handler.go	95	17	43	155	17b6cce42759034c7147665381a4af9b
pellar-prime-develop/rpc/json.go	70	18	17	105	997b8ffa19c08404dcbc593f0f04d390
pellar-prime-develop/rpc/routes.go	12	45	5	62	33e9c7510c2d0a3e086b6b440210130f

pellar-prime-develop/rpc/server.go	93	26	21	140	1c6c210472ff77857a4d8e8c39e7ff28
pellar-prime-develop/rpc/server_test.go	126	48	33	207	48c8b9f230d73d2097285ad7e3282977
pellar-prime-develop/rpc/utils.go	77	24	15	116	93578944e0b50df04c4a9e6215ff48c1
pellar-prime-develop/sequencer/sequencer.go	134	43	39	216	91edbf908c8f3251719a260c1b6fc442
pellar-prime-develop/storage/leveldb.go	38	18	14	70	e4e0335b84c75606bd4eb989dcaf083c
pellar-prime-develop/storage/leveldb_test.go	29	2	8	39	a01c2f1b6afff63e47d48592b1daabe5
pellar-prime-develop/storage/memorydb.go	104	0	25	129	d4a0280c81784d5ea11a7e0f3a019fd1
pellar-prime-develop/storage/memorydb_test.go	66	5	23	94	dc1045347cfe3fcab6248f027f2c4cd7
pellar-prime-develop/storage/storage.go	19	0	4	23	7bd4617d20e385f01d71c74c8cc51a5b
pellar-prime-develop/tracers/js/bigint.go	2	16	3	21	76ea8b88c2cb60c4b68b7061aa326d20
pellar-prime-develop/tracers/js/goja.go	800	72	92	964	5dc12057ecd3cc3a1d632cc29ad15745
pellar-prime-develop/tracers/js/internal/tracers/tracers.go	34	20	6	60	ceb6e612280d978f263f3c8887c20384
pellar-prime-develop/tracers/js/tracer_test.go	273	23	24	320	1ce7bf44d50eae1847d44baf42748533
pellar-prime-develop/tracers/tracers.go	36	27	10	73	24dcfaccc82a6809cc45177566f68cf3
pellar-prime-develop/utils/helpers.go	17	19	11	47	97f853f647d2db7c0157235d3b67548d
pellar-prime-develop/utils/merkle.go	57	0	13	70	cd375b34c90c8640753e51b680ed3fdf
pellar-prime-develop/utils/merkle_test.go	69	8	27	104	03c02f8584836b579e3f21ed31b1aec1

5.2 Used Packages

Dependency / Import Path	Version
github.com/ethereum/go-ethereum	v1.10.26
github.com/go-chi/chi	v1.5.4
github.com/go-chi/cors	v1.2.1
github.com/joho/godotenv	v1.4.0
github.com/mitchellh/mapstructure	v1.5.0
github.com/sirupsen/logrus	v1.9.0
github.com/stretchr/testify	v1.8.1
github.com/syndtr/goleveldb	v1.0.1-0.20220614013038-64ee5596c38a
github.com/spf13/cobra	v1.6.1

5.3 CallGraph (__MAIN__)



5.4 Codebase Overview

Source: https://github.com/pellartech/pellar-prime
Branch: develop
Commit: ee8f0ffedf775bee3d9dd118360ff16204fc0cef

Total : 119 files, 10723 codes, 2850 comments, 2384 blanks, all 15957 lines

language	files	code	comment	blank	total
Go	84	9,370	2,580	2,265	14,215
YAML	19	632	4	19	655
JavaScript	11	560	262	62	884
XML	1	56	0	5	61
Markdown	1	53	0	26	79
JSON	1	31	0	0	31
Makefile	1	11	1	5	17
Docker	1	10	3	2	15

6. Scope of Work

The Pellar Technology Team provided us with the files that needs to be tested. The scope of the audit is the LightLink / Prime architecture and codebase.

1. Automated Vulnerability Test (OWASP, SonarQube, Snyk, gosec, gokart)
2. Manual Security Testing (Line by line, CVE, Common Go-Eth Vulnerabilities, etc.)
3. Test environment deployment
4. Evaluating and testing software architecture
   - Development Productivity (Code Conventions Check, packages)
   - Functions & Logic Testing
   - Performance (Connection pooling, caching, Transaction Concurrency)
   - Reliability & Availability
   - Maintainability & Ease of Deployment

The main goal of this audit was to make sure the infrastructure is built according to newest standards and securely developed. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview

No	Title	Severity	Status
6.2.1	Spelling Errors	LOW	FIXED
6.2.2	Single Nested if	INFORMATIONAL	FIXED
6.2.3	Insufficient Documentation	INFORMATIONAL	FIXED

6.2 Manual and Automated Vulnerability Test

Manual security testing involves the auditor manually reviewing the codebase and testing the Layer-2 solution to identify vulnerabilities and security issues. This process involves a variety of techniques, such as code review, testing the system with various inputs to identify unexpected or malicious behavior, and analyzing the system's architecture and design for potential vulnerabilities. Automated security testing involves the use of specialized software tools to scan the codebase and test the system for vulnerabilities and security issues.

Our auditors have used a combination of both manual and automated testing. The audit team covered a range of issues during the audit, including, but not limited to, the following ones: Memory Leak, Unreleased Resources, Overflow/Underflow, Concurrency, Integer Truncation, Inappropriate Setting, Go-routine Unsafe Exiting, Closing Channel Twice, Missing Check for Unsafe External Input, Unsafe Encryption, Unsafe Random Number Generation, Data Security, DOS Attack, Security in Consensus Algorithm, Handling of Block, Transaction Security, Database Security, Oracle Security , Security of Third-party Libraries, Logic Vulnerability, Code Improvement etc.

The below issues summaries the findings:

CRITICAL ISSUES

During the audit, Chainsulting‘s experts found **0 Critical issues** in the codebase and architecture

HIGH ISSUES

During the audit, Chainsulting’s experts found **0 High issues** in the codebase and architecture

MEDIUM ISSUES

During the audit, Chainsulting’s experts found **0 Medium issues** in the codebase and architecture

LOW ISSUES

6.2.1 Spelling Errors

Severity: LOW
Status: FIXED
Code: CWE-703
File(s) affected: message_handlers.go, node.go, peer.go
Update: https://github.com/pellartech/pellar-prime/commit/000cada52b67338ab47396db9f4362570de3ebee
https://github.com/pellartech/pellar-prime/commit/b055d4ea0e8fe94514e63506387b8622ff40b09c

Attack / Description	We have identified several spelling errors in the codebase or comments. These errors can make it difficult for other developers to understand the purpose and functionality of the code, and can also lead to confusion and potentially even errors if the code is being used as a reference while writing or modifying other code.
Code	Line 110 – 127 (message_handlers.go) `// Data received from peer` `    case "block":` `        n.RecievedBlockHandler(m, p)` `    case "chain_length":` `        n.RecievedChainLengthHandler(m, p)` `    case "transaction":` `        n.RecievedTxHandler(m, p)` `    case "block_range":` `        go n.RecievedBlockRangeHandler(m, p)` `    case "quiz":` `        n.RecievedQuizHandler(m, p)` `    case "mempool":` `        n.RecievedMemPoolHandler(m, p)` `    case "organisations":` `        n.RecievedOrganisationsHandler(m, p)`

	`` case "organisation_accounts": n.RecievedOrganisationAccountsHandler(m, p) } } ``` Line 189 (message_handlers.go) `// Returns all known organisations to áthe requesting peer.` Line 301 (message_handlers.go) `// chain length. We request peers chain lenth when the node is syncing with` Line 533 (message_handlers.go) `// Verfiy peer are who they say they are` Line 161 (node.go) `// file. If we can connect, starts a new MessageListener() on a sepreate` Line 36 (peer.go) `// on the netowrk.`
Result/Recommendation	We recommend conducting a thorough review of the code to identify and correct any spelling errors and issues with grammar and punctuation. This will improve the readability and clarity of the code and make it easier for other developers to understand the purpose and functionality of the code. To address these issues, we recommend using a spell checker tool and reviewing the code carefully to ensure that it is clear and accurately reflects the intended functionality. e.g. Recieved = Received lenth = length sepreate = separate netowrk = network

INFORMATIONAL ISSUES

6.2.2 Single nested if
Severity: INFORMATIONAL
Status: FIXED
Code: NA
File(s) affected: message_handlers.go, memory_pool.go
Update: https://github.com/pellartech/pellar-prime/commit/29db9f72cecb9e9520040a35718848caee87b144

<table>
<tr><td>Attack / Description</td><td>A single nested if inside an else block can be replaced with an else if.</td></tr>
<tr><td>Code</td><td>Line 569 – 580 (message_handlers.go)
<pre>
// Check if this node is sequencer
    addr, _ := n.GetNodeWallet()
    if common.HexToAddress(n.Config.Sequencer) == *addr {
        // Broadcast tx to all peers except peer that sent it.
        n.BroadcastMsgExcept(&networkMessageIn, p)
    } else {
        // Verify tx is from sequencer.
        if !strings.EqualFold(p.Account.Address.Hex(), n.Config.Sequencer) {
            n.Logger.Log.WithFields(logrus.Fields{"error": err}).Error("non sequencer tx received")
            return
        }
    }
</pre>
Line 102 - 107 (memory_pool.go)
<pre>
} else {
        // Check if the nodes min gasPrice is higher than the transaction's gasPrice
        if big.NewInt(int64(chain.Config.MinimumGasPrice)).Cmp(tx.GasPrice()) == +1 {
            return ErrTxGasPriceLow
        }
</pre></td></tr>
</table>

	}
Result/Recommendation	It's better to have as little nesting as possible. Hence, it's cleaner to replace a single nested if inside an else with an else-if.

6.2.3 Insufficient Documentation
Severity: INFORMATIONAL
Status: FIXED
Code: CWE-1059
File(s) affected: NA
Update: https://pellartech.github.io/lightlink-chain-docs/developer-guide/

Attack / Description	Technical documentation is insufficient and needs to be improved. We missing several topics like, hardware requirements, node setup steps on different operating systems, tools, logs, etc. That are important topics or features that are not covered in the documentation at all. This is a significant issue as it can lead to decreased developer satisfaction and increased support requests.
Code	https://pellartech.github.io/lightlink-chain-docs/
Result/Recommendation	We recommend to prioritize improving the technical documentation to ensure that developers have the information they need to effectively use the product. For example: https://community.optimism.io/docs/developers/releases/# https://geth.ethereum.org/docs/getting-started

6.3 Common Blockchain Vulnerabilities

Legend:

✓ **Don’t need any further attention and checked**

X Needs to be checked and addressed

Vulnerability Title	Note / Recommendation	Status
Sybil Attack	*Description:* The attacker can subvert the blockchain by creating a large number of pseudonymous identities (i.e. Fake user accounts) and push legitimate entities in the minority. Such virtual nodes can act like genuine nodes to create a disproportionately large influence on the network. This may lead to several other attacks like DoS, DDoS, etc. *Recommendation:* Sequencer should be always whitelisted and known to the L2 Network provider.	✓
Eclipse Attack	*Description:* The attacker monopolizes all of the victim’s incoming and outgoing connections, isolating the victim from the rest of her peers in the network. *Recommendation:* Relative unlikely with the whitelisting process, but in case that changes in the future, we recommend to implement a maximum number of connections to a node and limit the number of hosts with a single IP address.	✓
Eavesdropping Attack	*Description:* The attacker passively listens to network communications to gain access to private information, such as node identification numbers, routing updates, or application-sensitive data. The attacker can use this private information to compromise nodes in the network, disrupt routing, or degrade application performance.	✓

	Recommendation: Enforce encrypted communications using encryption protocols, such as HTTPS or add the usage of reverse proxy to secure the connection, to the documentation. The reverse proxy setup can be also a standard for new nodes to get whitelisted. **Update:** We use a reverse proxy and SSL on live networks to encrypt traffic.	
Denial of Service Attack	*Description:* a denial-of-service attack (DoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting the services of a host connected to the Internet. Denial of service is typically accomplished by flooding the targeted machine or resource with an massive amount of requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled. *Recommendation:* a. Increase the number of nodes in different regions. b. Prevent malformed parameters from crashing the software. c. Limit memory queue size. **Update:** a. We deploy nodes in different availability zones on AWS to mitigate this. b. We validate all RPC & REST parameters. c. We have a size limit on the memory queue	✓
BGP Hijack Attack	*Description:* BGP hijacking (sometimes referred to as prefix hijacking, route hijacking, or IP hijacking) is the	✓

	illegitimate takeover of groups of IP addresses by corrupting Internet routing tables maintained using the Border Gateway Protocol (BGP). *Recommendation:* Increase the number of nodes in different regions.	
Timejacking	*Description:* Timejacking exploits a theoretical vulnerability in Bitcoin timestamp handling. During a timejacking attack, a hacker alters the network time counter of the node and forces the node to accept an alternative blockchain. This can be achieved when a malicious user adds multiple fake peers to the network with inaccurate timestamps. *Recommendation:* A timejacking attack can be prevented by restricting acceptance time ranges or using the node's system time.	✓
Cross-Domain Phishing Attack	*Description:* The hacker tricks the victim into opening a malicious webpage, connects to the cryptocurrency wallet RPC port through a cross-domain request, and then steals crypto assets. *Recommendation:* Prohibit nodes from enabling cross-domain access.	✓
[Long Range Attack]()	*Description:* A Long-Range attack is an attack scenario where the adversary goes back to the genesis block and forks the blockchain. The new branch is populated with a partially, or even completely, different history than the main chain. The attack succeeds when the branch that is crafted by the adversary becomes longer than the main chain, hence it overtakes it. *Recommendation:* The exchange or receiver should	✓

	complete payment after the transaction was confirmed by enough blocks.	
Bribery Attack	*Description:* Also referred to as Short-Range attack relies on bribing validators to work on specific blocks or forks. By doing that, the attacker can present arbitrary transactions as valid and having dishonest nodes paid to verify them. By paying them an amount equal to or more than the block rewards (in case the block is reverted by the network). *Recommendation:* PoS tackles this issue by either enforcing a slashing condition or by releasing violators from their position. Whitelisting nodes with extensive on-boarding checks makes that attack obsolete.	✓
P+epsilon attack	Description: P+epsilon attack states that it is possible to bribe users without having to pay them, as the system will award the bribe to the dishonest nodes by making that branch the main chain. For these case, the attacker faces a more significant problem as in case the malicious branch is reverted for some reason (attacker cannot continue the bribe, dishonest nodes stop working on that branch) the attacker would have to pay an enormous amount of bribes as the bribes will accumulate for every maliciously minted block. *Recommendation*: PoS tackles this issue by either enforcing a slashing condition or by releasing violators from their position. Whitelisting nodes with extensive on-boarding checks makes that attack obsolete.	✓
Cross-Domain Phishing Attack	*Description:* The hacker tricks the victim into opening a malicious webpage, connects to the cryptocurrency wallet	✓

	RPC port through a cross-domain request, and then steals crypto assets. *Recommendation:* Prohibit nodes from enabling cross-domain access.	
[Liveness Denial]()	*Description:* Liveness Denial is a form of Denial of Service attack in PoS protocols. In this attack, some or all of the validators decide to take action and purposefully block transactions by stopping publishing blocks. By avoiding to perform their validator duties, the blockchain will come to a halt as new blocks would not be able to be validated and published in the blockchain. A liveness requirement which slowly drains the stake of inactive validators will ensure that even if the majority of validators are either offline or performing a liveness denial attack, they would not compromise the network. *Recommendation*: In cases where liveness cannot be assessed, the community will be able to decide (off-chain communication) to fork the blockchain and remove the inactive validators. In all cases, validators who conduct this type of attack jeopardize their position in the network as validators and their stake if a slashing condition exists	✓
[Censorship]()	*Description:* Censorship in blockchains is big a thorny issue that has sparked many discussions as it can be considered an attack or a feature simultaneously, depending on the nature of the blockchain. Validators have control of which transactions will be added to a block which gives them the power to blacklist certain addresses. Transactions lie in the transaction pool, and validators take transactions and add them to their soon-to-be-published block. Validators might decide to remove some	✓

	transactions from their blocks. In the scenario of a single validator performing the censorship, some transactions might be delayed or be invalidated due to time constraints. The danger of censorship becoming more real is amplified once the number of validators performing this attack increases. *Recommendation:* Liveness requirements, can ensure the eventual process of transactions and eliminate censorship on the blockchain. In addition to that, the protocol can punish nodes which do not create blocks in a protocol-defined order. In another more effective solution, Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs), can be used to hide the identity of the transaction sender	
Finney Attack	*Description:* A Finney attack is possible when one transaction is premined into a block and an identical transaction is created before that premined block is released to the network, thereby invalidating the second identical transaction. *Recommendation:* The exchange or receiver should complete payment after the transaction was confirmed by enough blocks.	✓
Vector76 Attack	*Description*: Vector76 is a combination of Race attack and Finney attack. In this case, a malicious miner creates two nodes, one of which is connected only to the exchange node and the other of which is connected to well-connected peers in the blockchain network. After that, the miner creates two transactions, one high-value and one low-value. Then, the attacker premines and withholds a block with a high-value transaction from an exchange	✓

	service. After a block announcement, the attacker quickly sends the premined block directly to the exchange service. It along with some miners will consider the premined block as the main chain and confirm this transaction. Thus, this attack exploits the fact that one part of the network sees the transaction the attacker has included into a block while the other part of the network doesn't see this transaction. After the exchange service confirms the high-value transaction, the attacker sends a low-value transaction to the main network, which finally rejects the high-value transaction. As a result, the attacker's account is credited the amount of the high-value transaction. *Recommendation*: The exchange or receiver should complete payment after the transaction was confirmed by enough blocks.	
Grinding Attack	*Description:* Also known as precomputation attack. It is an implementation-specific issue and affects PoS systems. By exploiting the lack of randomness in the slot leader election process, a slot leader is capable of manipulating the frequency of them being elected in subsequent blocks. This issue can be solved by enforcing randomness to the process and minimizing or even eliminating influence factors of this process which are controlled by the validators. *Recommendation*: The exchange or receiver should complete payment after the transaction was confirmed by enough blocks.	✓
Cryptographic Attack	*Description:* Common attack methods: Analytic Attack / Implementation Attack / Statistical Attack / Brute Force /	✓

	Frequency Analysis and the Ciphertext Only Attack / Known Plaintext / Chosen Ciphertext / Chosen Plaintext / Meet in the Middle / Man in the Middle / Birthday attack / Replay attack / Collision attack *Recommendation:* Don't use unknown encryption library.	
Transaction Replay Attack	*Description:* Also known as Double Spend attack. Double spending is one of the problems that blockchain technologies attempt to solve since their very inception. Most, if not all of the attacks in the blockchain, aim to perform a double spend at some point in their execution. In this attack scenario, an attacker attempts to spend the same currency at least two times, hence double-spend. This attack is definitely not possible in the physical terms of currency. It is not possible to buy a resource from one vendor and then spend the exact same coins to another vendor. The attacker attempts to perform a transaction, wait for the merchant to approve it, and then reverts it and spends the same currency in another transaction. In blockchains, this can be achieved by presenting a conflicting transaction possibly in a different branch. BFT systems with the use of absolute finality are considered to be robust against the double spend problem. *Recommendation:* a. Check whether a UTXO has been spent. b. Use nonce to prevent transaction replay.	✓

6.4 Test Coverage

STATUS	ELAPSED	TEST	PACKAGE
PASS	1.01	TestMineBlock_LowBalance	core
PASS	1.01	TestMineBlock_SortNonceOrdering	core
PASS	1	TestMineBlock	core
PASS	1	TestDelete_TransactionProcessed	core
PASS	0.01	TestProcessGenesis	core
PASS	0.01	TestMemoryPool_Avg	core
PASS	0.01	TestValidateTx_TxSize	core
PASS	0	TestProcessBlock_InvalidTx	core
PASS	0	TestProcessEnterpriseTx	core
PASS	0	TestValidate_ProtocolGreater	core
PASS	0	TestValidate_Index	core
PASS	0	TestValidate_PreviousHash	core
PASS	0	TestValidate_Timestamp	core
PASS	0	TestValidate_TimestampOld	core
PASS	0	TestValidate_TimestampFuture	core
PASS	0	TestValidate_Mined	core
PASS	0	TestValidate_SigValid	core
PASS	0	TestValidate_ToEth	core
PASS	0	TestValidateTx_NegativeValue	core
PASS	0	TestLoadGenesisBlock	core
PASS	0	TestValidate	core
PASS	0	TestProcessBlock	core
PASS	0	TestProcessBlock_MissingPrevBlock	core

PASS	0	TestNewAccount	core
PASS	0	TestProcessTransaction	core
PASS	0	TestGetChainLength	core
PASS	0	TestGetBlockRangeMerkleRoot	core
PASS	0	TestCreateOrganisation	core
PASS	0	TestUpdateOrganisation	core
PASS	0	TestCreateOrganisationAccount	core
PASS	0	TestCreateRemoveOrganisationAccount	core
PASS	0	TestValidateEnterpriseTx	core
PASS	0	TestValidateEnterpriseTx_SenderNotOrg	core
PASS	0	TestValidateEnterpriseTx_QuotaExceeded	core
PASS	0	TestGetMerkleRoot	core
PASS	0	TestToMessage	core
PASS	0	TestAsMessage	core
PASS	0	TestEstimateGas	core
PASS	0	TestCreateGenesis	core
PASS	0	TestInsert	core
PASS	0	TestInsert_FundsLow	core
PASS	0	TestInsert_DuplicateTx	core
PASS	0	TestInsert_TxWithHigherGasPrice	core
PASS	0	TestInsert_TxWithLowerGasPrice	core
PASS	0	TestInsert_TxWithHigherNonce	core
PASS	0	TestInsert_AccountLimit	core
PASS	0	TestInsert_MempoolFull	core
PASS	0	TestInsert_MempoolFullHigherGasPrice	core
PASS	0	TestInsert_MempoolFullEqualGasPrice	core
PASS	0	TestTraceTransaction	core

PASS	0	TestValidate_ProtocolLow	core
PASS	0	TestBlockEncodeDecode	core
PASS	0	TestValidateTx_GasLimitTooHigh	core
PASS	0	TestValidateTx_InvalidSig	core
PASS	0	TestValidateTx_LowNonce	core
PASS	0	TestValidateTx_LowBalance	core
PASS	0	TestPop	core
PASS	0	TestShift	core
PASS	0	TestSize	core
PASS	0	TestAdd	core
PASS	0	TestDelete	core
PASS	0	TestDelete_RemovedFromDB	core
PASS	0	TestAddTransaction	core
PASS	0	TestVerifySignature	core
PASS	0	TestMemoryPool_AvgLowMaximumPoolSize	core
PASS	0	TestSortByGasPrice	core
PASS	0	TestDelete_HashHash	core
PASS	0	TestAddressList	core
PASS	0	TestGenerateSignature	core
PASS	0	TestNewBlock	core
PASS	0	TestAccountEncodeDecode	core
PASS	0	TestNewOrganisation	core
PASS	0	TestOrganisationEncodeDecode	core
PASS	0	TestNewReceipt	core
PASS	0	TestReceiptEncodeDecode	core
PASS	0	TestStoreBlock	core
PASS	0	TestGetBlockHash	core

PASS	0	TestGetBlockIndex	core
PASS	0	TestGetLastBlock	core
PASS	0	TestGetBlockRange	core
PASS	0	TestStoreTempBlock	core
PASS	0	TestDeleteTempBlock	core
PASS	0	TestStoreTransaction	core
PASS	0	TestDeleteTransaction	core
PASS	0	TestStoreReceipt	core
PASS	0	TestStoreOrganisation	core
PASS	0	TestStoreOrganisationAccount	core
PASS	0	TestRegister	core
PASS	0	TestValidateTx	core
PASS	0	TestNewTransaction	core
PASS	0	TestSign	core
PASS	0	TestIsEnterprise	core
PASS	0	TestTransactionEncodeDecode	core
FAIL	0	TestCalculateHash	core
PASS	0.00	TestErrors/Test_Unkown_Method	rpc
FAIL	0.00	TestErrors/Test_Invalid_JSON	rpc
FAIL	0.00	TestErrors/Test_No_ID	rpc
PASS	0.00	TestEthProtocolVersion	rpc
PASS	0.00	TestEthProtocolVersion/Test_Happy_Path	rpc
PASS	0.00	TestEthProtocolVersion/Test_No_Params	rpc
FAIL	0.00	TestEthGetBalance/Test_Happy_Path	rpc
PASS	0.00	TestEthSyncing/Test_Happy_Path	rpc
FAIL	0.00	TestEthSyncing/Test_No_Params	rpc
PASS	0.00	TestEthCoinbase	rpc

PASS	0.00	TestEthCoinbase/Test_Happy_Path	rpc
PASS	0.00	TestEthCoinbase/Test_No_Params	rpc
PASS	0.00	TestEthBlockNumber	rpc
PASS	0.00	TestEthBlockNumber/Test_Happy_Path	rpc
PASS	0.00	TestEthBlockNumber/Test_No_Params	rpc
FAIL	0.00	TestNetVersion/Test_Happy_Path	rpc
PASS	0.00	TestNetListening	rpc
PASS	0.00	TestNetListening/Test_Happy_Path	rpc
PASS	0.00	TestNetPeerCount	rpc
PASS	0.00	TestNetPeerCount/Test_Happy_Path	rpc
PASS	0.00	TestEstimateGas	rpc
PASS	0.00	TestEstimateGas/EstimateGas_–_Simple_Tx	rpc
PASS	0.00	TestEstimateGas/EstimateGas_–_Tx_with_Messa	rpc
FAIL	0.00	TestPrimeGetTxnProof/Test_Happy_Path	rpc
FAIL	0.00	TestWeb3ClientVersion/Test_Happy_Path	rpc
PASS	0.00	TestWeb3Sha3	rpc
PASS	0.00	TestWeb3Sha3/Test_Happy_Path	rpc
PASS	0.00	TestWeb3Sha3/Test_Missing_Param_0	rpc
PASS	0.68	TestConnectLevelDB	storage
PASS	0	TestMemoryDB	storage
PASS	0	TestMemoryIterator	storage
PASS	0	TestMemoryIterator_Empty	storage
PASS	0	TestCalculateMerkleRoot	utils
PASS	0	TestCalculateMerkleRootOdd	utils
PASS	0	TestCalculateMerkleProof	utils
PASS	0	TestCalculateMerkleProofOdd	utils
PASS	0	TestCalculateMerkleProofSingle	utils

PASS	1	TestHalt	js
PASS	0.01	TestEnterExi	js
PASS	0.01	TestSetup	js
PASS	0	TestHaltBetw	js
PASS	0	TestNoStepEx	js
PASS	0	TestIsPrecom	js
FAIL	0.03	TestTracer	js

Summary

ELAPSED	PACKAGES	PASS	FAIL	SKIP
8.49s	core, rpc, storage,tracer,utils	129	15	0

Update:
https://github.com/pellartech/pellar-prime/commit/e578f8f4bf5d1613b57832c6bfed16cc1cd437a6

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the architecture and codebase. The final debriefs took place on the December 25, 2022.

The main goal of this audit was to make sure the codebase is built according to newest standards and securely developed. During the audit, no critical, high, or medium issues were found, after the manual and automated security testing. Overall, the code quality and architecture had a high grade of professionalism. Several changes and recommendations were proposed to reduce the code's attack surface and improve its overall quality.

As a final remark, we must highlight that given the sandboxed environment, the number of possible interactions cannot be audited to exhaustion. We therefore highly advise:

1. Following best practices of secure software development, thorough test-driven development, and mandatory peer-reviews.
2. Further promoting a public bug bounty program to engage independent security researchers from the community in uncovering further misbehaviours in the system as the code base evolves.
3. Continuing with beta testing phases until several projects have been onboarded to the Layer 2 scaling solution, and their dynamics have become battle-tested.

Update (12.01.2023): Re-check has been successfully done and the LightLink team fixed all issues.

8. About the Auditor

Chainsulting is a professional software development firm, founded in 2017 and based in Germany. They show ways, opportunities, risks and offer comprehensive Web3 solutions. Their services include Web3 development, security and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Solana, Tezos, Ethereum, Binance Smart Chain, and Polygon to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of numerous other top DeFi projects.

Chainsulting currently secures [$100 billion] in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the web3 sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

Check our website for further information: https://chainsulting.de