HACKEN

Smart Contract Platform Security Analysis Report

Customer: Waterfall.network

Date: 24/06/2024

We express our gratitude to the Waterfall.network team for the collaborative engagement that enabled the execution of this Security Assessment.

Waterfall Protocol is a high-performance, scalable, Proof-of-Stake smart contract platform. It addresses blockchain's scalability, security, and decentralization challenges by leveraging Directed Acyclic Graph (DAG) technology for parallel block production. Key features include low transaction fees, EVM compatibility, and a highly decentralized network with minimal hardware requirements for nodes.

Platform: Waterfall

Language: Golang

Tags: Layer 1, BlockDAG

Timeline: 22/04/2024 - 24/06/2024

Methodology: Blockchain Protocol and Security Analysis Methodology

Review Scope

Repository	https://gitlab.waterfall.network/waterfall/protocol/coordinator
Commit	9b3570c351d927004e4c5da26f908fda5f6ce65a
Repository	https://gitlab.waterfall.network/waterfall/protocol/gwat
Commit	6782049b74c92e58a8ca077a3015ec67dc7ef943

The system users should acknowledge all the risks summed up in the risks section of the report

11	9	2	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	1
High	1
Medium	1
Low	8

Vulnerability	Status
F-2024-3344 - Inadequate Sender Validation in Deposit Transaction Processing	Accepted
F-2024-3521 - Divergence from ERC-721 Standard in Token Implementation	Accepted
F-2024-1616 - Critical Vulnerabilities in Go Standard Library	Fixed
F-2024-1864 - Deprecated Elliptic Curve Cryptography	Fixed
F-2024-2188 - Inherited Issues from Go-Ethereum	Fixed
F-2024-2256 - Compatibility Concerns Arising from Outdated EVM Implementation	Fixed
F-2024-2993 - Inherited Issues from Prysm	Fixed
F-2024-3084 - Utilization of Non-Supported Fork Choice Storage Mechanism	Fixed
F-2024-3227 - Insufficiencies in Light Client Implementation	Fixed
F-2024-3346 - Bypassing Execution Layer During Validator Exit	Fixed
F-2024-3522 - Incorrect Event Emitted During Token Operations	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Layer 1 Protocol Code Review and Security Analysis Report for Waterfall.network
Audited By	Sofiane Akermoun, Nino Lipartiia
Approved By	Sofiane Akermoun
Website	https://waterfall.network/
Changelog	03/06/2024 - Preliminary Report
Changelog	24/06/2024 - Final Report

Table of Contents

System Overview

Waterfall is a Layer 1 solution derived from Prysm and Go-Ethereum, implementing consensus and execution clients in line with the Ethereum 2.0 architecture. This foundation is enhanced with several features to improve scalability and decentralization.

Key innovations include the implementation of BlockDag logic in the execution client, which allows for parallel block production. Moreover, the adoption of an optimistic consensus mechanism expedites the process by furnishing blocks with a high likelihood of finalization.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's Layer 1 protocol project. Detailed scoring criteria can be referenced in the corresponding section of the Blockchain Protocol and Security Analysis Methodology.

Documentation quality

- Source code documentation draws from both Geth and Prysm projects, ensuring a strong foundational understanding and continuity within the project.
- Waterfall-specific features and changes are meticulously documented, emphasizing the platform's unique aspects.
- The in-code documentation of features falls short of providing comprehensive insights into the codebase.
- There's a noticeable lack of comprehensive documentation available on the project's website, hindering accessibility and understanding for stakeholders.
- Building instructions are insufficient, posing challenges for developers and node operators seeking to engage with the platform effectively.
- The absence of adequate documentation in English presents a significant barrier to wider adoption and collaboration.
- The Waterfall team has indicated their intention to significantly enhance the documentation prior to the public release of the code.

Code quality

- The project inherits strong code quality from Geth and Prysm, forming a sturdy foundation for development.
- Adequate test coverage, which could benefit from enhancement.
- The presence of numerous lint warnings suggests areas where refinement is needed to align with coding best practices.
- Unresolved TODO comments and "implement me" panics.
- Notable residue code underscores the importance of thorough review and cleanup to optimize performance and maintainability.
- The Waterfall team has conveyed their intent to improve code quality and elevate the overall standard of the codebase before its public release.

Architecture quality

- Waterfall derives its architecture from Prysm and Go-Ethereum, aligning its consensus and execution clients with the Ethereum 2.0 framework.
- An innovative architectural design in the execution client replaces the traditional blockchain structure with BlockDAG, enhancing scalability.
- The consensus client architecture is grounded in blockchain technology, ensuring the system's robustness and integrity.
- The integration of an optimistic consensus mechanism showcases well-designed and effectively implemented advancements.

Findings

Vulnerability Details

F-2024-1616 - Critical Vulnerabilities in Go Standard Library - Critical

Description:

Our security analysis, performed using the `govulncheck` tool, has uncovered several vulnerabilities within the standard library dependencies outlined in the Go version 1.20 specified within the `go.mod` files of both the `coordinator` and `gwat` codebases. These vulnerabilities, detailed below, span various security concerns, including incorrect handling of HTTP headers and cookies, memory exhaustion risks, issues with certificate verification, denial of service (DoS) vectors, timing side channels in cryptographic operations, excessive CPU usage, insufficient sanitization, unsafe runtime behavior, and excessive resource consumption.

Key Vulnerabilities Identified:

GO-2024-2600: Incorrect Forwarding of Sensitive Headers and Cookies

- Issue: Incorrect forwarding of sensitive headers and cookies on HTTP redirect in net/http.
- Found in: net/http@go1.20
- Fixed in: net/http@go1.21.8
- Details: pkg.go.dev/vuln/GO-2024-2600

GO-2023-2382: DoS via Chunk Extensions

- Issue: Denial of service via chunk extensions in net/http.
- Found in: net/http/internal@go1.20
- Fixed in: net/http/internal@go1.20.12
- Details: pkg.go.dev/vuln/GO-2023-2382

GO-2023-2102: HTTP/2 Rapid Reset Issue

- Issue: HTTP/2 rapid reset can cause excessive work in net/http.
- Found in: net/http@go1.20
- Fixed in: net/http@go1.20.10
- Details: pkg.go.dev/vuln/GO-2023-2102

GO-2023-1878: Insufficient Sanitization of Host Header

- Issue: Insufficient sanitization of the Host header in net/http.
- Found in: net/http@go1.20
- Fixed in: net/http@go1.20.6
- Details: pkg.go.dev/vuln/GO-2023-1878

GO-2023-1704: Excessive Memory Allocation

- Issue: Excessive memory allocation in net/http and net/textproto.
- Found in: net/textproto@go1.20
- Fixed in: net/textproto@go1.20.3
- Details: pkg.go.dev/vuln/GO-2023-1704

GO-2023-1621: Incorrect Calculation on P256 Curves

- Issue: Incorrect calculation on P256 curves in crypto/internal/nistec.
- Found in: crypto/internal/nistec@go1.20
- Fixed in: crypto/internal/nistec@go1.20.2
- Details: pkg.go.dev/vuln/GO-2023-1621

GO-2023-1571: Denial of Service via Crafted HTTP/2 Stream

- Issue: Denial of service via crafted HTTP/2 stream in net/http and golang.org/x/net.
- Found in: net/http@go1.20
- Fixed in: net/http@go1.20.1
- Details: pkg.go.dev/vuln/GO-2023-1571

Impact Analysis:

The exploitation of these vulnerabilities, particularly in the context of inducing a Denial of Service (DoS) via RPC requests or any user-facing features that are based on the HTTP protocol, presents a critical threat to the stability and security of the network infrastructure. Such vulnerabilities compromise the integrity and availability of nodes, which are foundational to the network's resilience. The implications of these security flaws extend beyond mere disruptions, potentially paving the way for more sophisticated attacks. Notably, they could facilitate conditions conducive to a 51% attack, wherein an adversary could gain disproportionate control over the network, thereby compromising its security and governance. This scenario underscores a severe risk to the network's operational continuity and trustworthiness.

Exploit Likelihood:

Given the detailed documentation and the awareness of these vulnerabilities within the cybersecurity community, the likelihood of exploitation is considerably high. The public availability of information regarding these vulnerabilities significantly reduces the barrier to entry for potential attackers, making it imperative to assume a proactive defensive stance. The combination of their documented nature, ease of exploitation, and the high stakes involved in terms of network integrity and availability amplifies the urgency of addressing these vulnerabilities promptly. Consequently, it is vital to prioritize remediation efforts to mitigate the risk of malicious exploitation that could undermine the network's foundational security principles.

Assets:

- Dependencies

Status: Fixed

Classification

Impact: 5/5

Likelihood: 5/5

Severity: Critical

Recommendations

Remediation: In light of the vulnerabilities identified within the Go standard library dependencies as specified by the project's use of Go version 1.20, urgent action is required to safeguard the integrity and availability of the network, particularly given the significant risks these vulnerabilities pose, including the potential for a D.O.S. attack scenarios. The following measures are strongly recommended:

- **Immediate Upgrade**: Consider upgrading both the `gwat` and `coordinator` repositories to the latest stable version of Go, specifically Go 1.22, as it addresses not only the documented vulnerabilities but also additional security issues introduced in Go 1.21. This upgrade is critical for eliminating the identified security risks and ensuring robust protection against the vulnerabilities present in current and previous versions.
- **Dependency Audit and Update**: Conduct a comprehensive audit of all project dependencies. This audit should not only verify that each dependency is updated to the most secure version but also assess the necessity of each dependency to minimize the attack surface.
- **Security Patch Application**: For dependencies that cannot be immediately updated, apply available security patches or workarounds to mitigate known vulnerabilities. This stopgap measure should only be temporary while plans for a more sustainable update are enacted.
- **Optimize CI/CD Pipeline**: Implement a vulnerability dependency checker within your CI/CD workflow. This proactive strategy facilitates early detection and resolution of potential security vulnerabilities in project dependencies. By integrating this tool into your development lifecycle, you can effectively mitigate security risks before they impact your application's security integrity.
- **Enhanced Monitoring and Logging**: Implement enhanced monitoring and logging of network activity and system performance to detect unusual patterns that may indicate an attempted or successful exploitation of these vulnerabilities. Early detection is key to preventing widespread impact.

- **Vulnerability Management Process**: Establish or refine a vulnerability management process that includes regular scans, assessments, and updates of dependencies. This process should also involve staying informed on new vulnerabilities and threats as they are discovered.
- **Education and Awareness**: Increase awareness among the development and operations teams regarding the importance of security practices, particularly around dependency management and vulnerability mitigation. Encourage a culture of security-first thinking.

F-2024-3346 - Bypassing Execution Layer During Validator Exit - High

Description:

The current architecture of the Waterfall network stipulates that a validator must send a transaction with the `Exit` operation through the execution layer, which is then validated and passed to the consensus layer. However, an inconsistency arises due to the presence of CLI commands in the `coordinator` such as `VoluntaryExitPublicKeysFlag` and `ExitAllFlag`, which allow validators to exit directly from the consensus layer. These flags are remnants from the `prysm` implementation, which employs a different approach to validator exits compared to `coordinator`.

This inconsistency can lead to various issues and potential malicious behavior, resulting in security vulnerabilities. One significant concern is that this direct exit does not trigger the synchronization process on the `gwat` side, leaving the execution layer unaware of the change. Consequently, the states of the execution and consensus layers can become inconsistent, potentially leading to severe consequences.

Moreover, this incorrect flow bypasses several critical checks performed by `gwat`, including the validation of whether the address is permitted to withdraw and adherence to trial period rules. Bypassing these checks can have serious repercussions on the system's reliability and security.

Assets:

- validator module [https://gitlab.waterfall.network/waterfall/protocol/coordinator]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 5/5

Severity: High

Recommendations

Remediation:

It is advisable to deactivate the `VoluntaryExitPublicKeysFlag` and `ExitAllFlag` flags and remove the functions responsible for validator exits through the Command-line interface of the `coordinator`. Implementing these measures will help maintain consistency between the

execution and consensus layers, thereby enhancing the overall security and integrity of the network.

Additionally, thorough documentation should be provided to outline the correct procedure for validator exits, ensuring all stakeholders are aware of the required process.

By establishing a clear and secure exit protocol, the network can mitigate risks associated with unauthorized exits and maintain a higher standard of system reliability and security.

F-2024-2188 - Inherited Issues from Go-Ethereum - Medium

Description:

Despite substantial deviations from the original `geth` codebase, the current `gwat` implementation still retains known issues inherited from `geth`. Although these issues were previously resolved in the `geth` repository, they have yet to be addressed in `gwat`. The primary aim is to emphasize the crucial importance of ongoing codebase updates to rectify identified bugs and uphold system stability and security. Integrating comparable fixes from Go-Ethereum is essential to enhance the system's resilience and security posture.

The following list comprises bugs and their corresponding pull requests for fixes discovered across releases, commencing from v1.10.11. It is imperative to emphasize that the list encompasses only those issues that potentially remain pertinent to the `gwat` codebase, while others have been omitted.

- Rectify chain tracing to prevent encountering an Out of Memory (OOM) state (#23736).
- Address the simulated backend gas price suggestion to align with EIP-1559 standards (#23838, #23840). However, note that this fix introduces a data race condition issue, which necessitates the implementation of the follow-up fix included in #23898.
- Resolve a crash occurring in `abigen` generated code when backend header retrieval fails (#23781).
- Mitigate an issue where a malicious response could lead to the crashing of the syncing node (#23960).
- Amend the `receiptsRoot` field name in the EVM command output (#23924).
- Fix transaction sender recovery functionality in `ethclient` (#23877).
- Adjust the maximum Time to Live setting on Clouflare (#23885).
- Enhance intrinsicGas output in the t9n tool (#23889).
- Address a corner-case scenario in the transaction hash indexer (#24024).
- Update log messages related to RPC method invocations to prevent a name clash when using the JSON output format (#24112).
- Fix inconsistencies in block tracing via `debug.traceBlockByHash` (#24286).
- The issue concerning EIP-712, particularly when the compilation to WASM fails, has been resolved (#24029). Additionally, another issue related to this Ethereum Improvement Proposal, specifically regarding the incorrect signing-hash, has been fixed (#24220).
- Ensure accurate EVM execution times exported via metrics (#24304).
- Resolve edge cases in the range prover functionality (#24266, #24257).
- Rectify an error related to HTTP2 handling (#24292).

- Address several data races related to snapshot sync (#24685). Various improvements to snapshot sync are introduced (#25831, #25694, #25666, #25651).
- Prevent the JSON-RPC client from hanging when invalid batch results are returned by the server (#26064).
- Resolve a corner-case issue in the filter system (#26054).

It is noteworthy that many of these bugs do not pose an immediate security risk. However, others could lead to node halting, incorrect behavior, memory leakage, or even node crashes (see #23781, #23960, #23736).

Assets:

- geth [https://gitlab.waterfall.network/waterfall/protocol/gwat]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 2/5

Severity: Medium

Recommendations

Remediation: To mitigate the identified issues, it is advisable to address each specified problem individually. This entails conducting comprehensive assessments and implementing targeted solutions to resolve them. Alternatively, if any of the listed issues are not applicable to the project's current state, it is recommended to add a comment explaining the situation or remove the code if it is no longer in use.

Moreover, maintaining vigilant oversight over forthcoming releases of `geth` is imperative to promptly identify and address any pertinent issues that may arise. Regular monitoring ensures that the project remains up-to-date with the latest advancements and fixes in the underlying codebase, further fortifying its resilience and security posture.

By staying vigilant and proactively addressing potential vulnerabilities, the project can maintain a secure and stable codebase over the long term.

F-2024-1864 - Deprecated Elliptic Curve Cryptography - Low

Description:

The issue at hand concerns the use of deprecated methods and functions for elliptic curve cryptography. Specifically, in Go version 1.21.0 and onwards, functions such as `elliptic.Marshal`, `elliptic.Unmarshal` and `elliptic.GenerateKey`, alongside methods like `IsOnCurve`, `ScalarMult`, and `ScalarBaseMult`, have been deprecated. This deprecation is documented in the commit from the official Go repository.

In particular, the `coordinator` codebase employs the `ScalarBaseMult` function within the `p2p` package. Additionally, deprecated functions are also found across various packages within the `gwat` codebase, including:

- *crypto*
- *ecies*
- *rlpx*
- *scwallet*
- *v4wire*
- *v5wire*
- *discover*

Issues associated with these deprecated functions include potential panics and incorrect results. It's notable that `elliptic.Marshal` behavior becomes undefined if the provided point is not on the curve. Moreover, methods such as `IsOnCurve`, `ScalarMult`, and `ScalarBaseMult` are marked as low-level and unsafe in the documentation. Additionally, there have been reports of incorrect results from `ScalarMult` and `ScalarBaseMult`, indicating potential vulnerabilities (see issue **F-2024-1616** and report GO-2023-1621).

Such unexpected behaviors present a risk of introducing security vulnerabilities or inconsistencies, compromising the integrity and security of the codebase. Consequently, areas of the code utilizing these functions may become susceptible to exploitation, highlighting the urgent need for remediation.

Assets:

- prysm [https://gitlab.waterfall.network/waterfall/protocol/coordinator]
- beacon-chain/p2p module [https://gitlab.waterfall.network/waterfall/protocol/coordinator]
- geth [https://gitlab.waterfall.network/waterfall/protocol/gwat]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 1/5

Severity: Low

Recommendations

Remediation: To address this issue proficiently, consider the following strategies:

- Develop custom elliptic cryptography functionalities by referencing established implementations, such as those found in geth, tailored to meet your specific requirements.
- Transition to the usage of `crypto/ecdh` where appropriate, ensuring adherence to recommended practices for each function as specified in the documentation.
- In situations where direct remedies may not be the best fit, one potential approach is to incorporate logic for catching and recovering from panics.
- Keep your codebase up-to-date by upgrading to the latest stable version of Go, currently version 1.22, which integrates critical fixes and enhancements.
- Maintain a proactive stance towards maintenance by diligently monitoring updates in geth and promptly integrating relevant patches and improvements into your project. This approach will bolster the resilience and effectiveness of your cryptographic implementations, safeguarding them against potential vulnerabilities and ensuring the continued security of your applications.

Adopting these recommendations will not only mitigate potential security vulnerabilities but also ensure the continued reliability and effectiveness of your cryptographic implementations in Go.

F-2024-2256 - Compatibility Concerns Arising from Outdated EVM Implementation - Low

Description:

The current issue stems from an outdated implementation of the Ethereum Virtual Machine (EVM) within the `gwat` codebase. This implementation fails to integrate crucial modifications and improvements introduced in Go-Ethereum.

Notably, significant changes have been made to the EVM codebase, particularly in the *Sharblu* and *Hourglass Nebula* releases. One critical alteration is the implementation of EIP-3855, which introduces the `PUSH0` opcode. This alteration seeks to optimize contract size, reduce the likelihood of contract misuse of instructions, and diminish the necessity for `DUP` instructions. Nonetheless, the absence of support for the `PUSH0` opcode in `gwat` may potentially result in discrepancies in contract behavior, thereby posing risks to the integrity and reliability of the system.

Furthermore, it is pertinent to acknowledge the existence of several modifications and enhancements that notably enhance the speed and performance of the EVM. These enhancements encompass various aspects, including:

- Optimization of the EVM `MSTORE` opcode, resulting in 75% increase in speed (#24847, #24860).
- Streamlining of the EVM implementation and enhancement of interpreter loop performance through comprehensive code cleanup (#24120, #24048, #24085, #24026, #24031, #24040, #23970, #23952, #23974, #23977, #23967, #24066).
- Introduction of changes regarding the `INVALID` opcode 0xFE (#24017).
- Modernization of internal opcode names to align with Solidity (#23976, #24022, #24016).
- Resolution of issues related to generating Go/Java bindings for contracts with struct-typed constructor parameters (#23940).
- Migration of built-in EVM trace loggers from the core/vm to a dedicated package. (#23892).

While these changes may not be strictly essential for security purposes, their implementation can significantly enhance the performance of the EVM. Therefore, considering the adoption of these enhancements could lead to notable improvements in the efficiency and effectiveness of the EVM within the `gwat` framework.

Assets:

- geth [https://gitlab.waterfall.network/waterfall/protocol/gwat]

Status: Fixed

Classification

Impact: 1/5

Likelihood: 3/5

Severity: Low

Recommendations

Remediation: To address the issue effectively, it is imperative to prioritize the integration of critical updates from the Go-Ethereum codebase into `gwat`. Specifically, adding support for the `PUSH0` opcode, as per the implementation `geth` (see #24039), is essential.

Furthermore, it is advisable to consider implementing the additional modifications and enhancements outlined above to improve EVM compatibility, enhance the correctness of smart contract behaviors, and optimize overall system performance.

By diligently incorporating these proactive measures, `gwat` can ensure the seamless compatibility of its EVM implementation, thus fortifying the reliability and performance of the system over the long term.

F-2024-2993 - Inherited Issues from Prysm - Low

Description:

Despite significant deviations from the original `prysm` codebase, the current `coordinator` implementation still harbors known bugs and security vulnerabilities inherited from `prysm`. Although these issues were previously resolved in the `prysm` repository, they have yet to be addressed in `coordinator`. It is vital to prioritize continuous updates to the codebase to rectify identified security flaws and maintain system stability and security. Integrating similar fixes from Prysm is essential to bolster the system's resilience and security posture.

The following list includes security issues and their corresponding pull requests for fixes identified across releases, starting from v2.1.1. Additionally, several non-security-related bugs are worth addressing. The complete list of bugs can be found in the `prysm` repository. It is important to note that this list only includes issues that are potentially relevant to the `coordinator` codebase, while others have been omitted.

- Updated Base Docker Images (#11958).
- Fixed Pagination Panic (#12932).
- Updated Bazel Nogo configuration to run the `ineffassign` static analyzer for all Go files (#12578).
- Released Lock Before Panicking (#12464).
- Addressed incomplete key deletions via the key manager API (#12284).
- Mitigated potential DoS vulnerabilities via beacon node API endpoints if exposed to untrusted parties (issue #9247).

It is important to note that the last issue, although not fully resolved in the Prysm implementation, has been partially mitigated through comprehensive documentation on API public exposure, helping validators avoid risky behavior.

Assets:

- prysm [https://gitlab.waterfall.network/waterfall/protocol/coordinator]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 1/5

Severity: Low

Recommendations

Remediation:

To effectively resolve the identified issues, it is prudent to tackle each problem individually. This entails conducting thorough assessments, implementing precise solutions, and adding appropriate documentation where required. If any of the listed issues are not pertinent to the project's current status, it is advisable to provide explanatory comments or remove the redundant code altogether.

Moreover, maintaining vigilant oversight over forthcoming releases of `prysm` is essential to promptly identify and address any relevant issues that may arise. Regular monitoring ensures that the project remains up-to-date with the latest advancements and fixes in the underlying codebase, further fortifying its resilience and security posture.

By remaining vigilant and proactively addressing potential vulnerabilities, the project can maintain a secure and stable codebase over the long term.

F-2024-3084 - Utilization of Non-Supported Fork Choice Storage Mechanism - Low

Description:

The `prysm` implementation of the Beacon node, which served as the foundational code for the `coordinator` implementation, originally permitted the configuration of the storage option for the LMD-GHOST forkchoice. By default, the proto array is utilized, but there was an option to configure a doubly linked proto array node structure.

In the current iteration of the Waterfall project, the `coordinator` implementation exclusively supports the proto array option, rendering the doubly linked proto array unsupported. The issue arises because the configuration feature for the LMD-GHOST forkchoice store remains present. Specifically, the `enableForkChoiceDoublyLinkedTree` flag allows this configuration option when setting up the beacon-chain client:

coordinator/config/features/config.go:140

```
// ConfigureBeaconChain sets the global config based
// on what flags are enabled for the beacon-chain client.
func ConfigureBeaconChain(ctx *cli.Context) {
/*
Other features configuration
*/
if ctx.Bool(enableForkChoiceDoublyLinkedTree.Name) {
logEnabled(enableForkChoiceDoublyLinkedTree)
cfg.EnableForkChoiceDoublyLinkedTree = true
}
Init(cfg)
}
```

Enabling this feature and using the doubly linked proto array as a store for the LMD-GHOST forkchoice implementation can lead to unpredictable behaviors, as it has not been fully integrated into the `coordinator` implementation. This could result in critical issues such as panics due to unimplemented functions:

coordinator/beacon-chain/forkchoice/doubly-linked-tree/types.go:20

```
func (f *ForkChoice) CollectForkExcludedBlkRoots(leaf common.Hash) c
ommon.HashArray {
//TODO implement me
panic("implement me")
}

func (f *ForkChoice) GetParentByOptimisticSpines(ctx context.Context
, optSpines []common.HashArray, jCpRoot [32]byte) ([32]byte, error)
{
//TODO implement me
panic("implement me")
}
```

These potential issues significantly compromise the overall stability, robustness, and security of the client application, as well as the integrity and reliability of the blockchain network.

Assets:

- beacon-chain/forkchoice module [https://gitlab.waterfall.network/waterfall/protocol/coordinator]

Status: Fixed

Classification

Impact: 2/5

Likelihood: 1/5

Severity: Low

Recommendations

Remediation: It is advisable to deactivate the option to configure the node with the `EnableForkChoiceDoublyLinkedTree` flag enabled. Furthermore, it is suggested to contemplate the removal of the `doublylinkedtree` package if there are no intentions to maintain it in the foreseeable future.

Implementing these measures will significantly enhance the robustness and security of the system, thereby fortifying its resilience against potential vulnerabilities and ensuring its long-term stability.

F-2024-3227 - Insufficiencies in Light Client Implementation - Low

Description:

The current issue concerns the incomplete support for light clients within the execution layer of the `gwat` implementation. While `gwat` ostensibly supports light client operation via the `LightServeFlag`, and full node operators can activate light server functionality using the `SyncModeFlag`, several methods essential to the light client remain unimplemented. These methods currently trigger a panic response with the message `"implement me"` or `"not implemented"`, which undermines the functionality of the light client. The methods in question are as follows:

- `func (lc *LightChain) GetLastCoordinatedCheckpoint`
- `func (lc *LightChain) EnterNextEra`
- `func (lc *LightChain) StartTransitionPeriod`
- `func (lc *LightChain) EpochToEra`
- `func (lc *LightChain) GetValidatorSyncData`
- `func (lc *LightChain) GetTransaction`
- `func (lc *LightChain) GetTransactionReceipt`
- `func (lc *LightChain) GetEpoch`
- `func (lc *LightChain) IsSynced`
- `func (lc *LightChain) Synchronising`
- `func (lc *LightChain) FinSynchronising`
- `func (lc *LightChain) DagSynchronising`
- `func (lc *LightChain) IsRollbackActive`
- `func (lc *LightChain) CurrentHeader`
- `func (lc *LightChain) StateCache`
- `func (lc *LightChain) StateAt`
- `func (w *lightPeerWrapper) RequestHashesBySlots`

The absence of these implementations severely limits the light client's ability to access the same data as full nodes, effectively rendering them non-functional. The potential of these functions to induce panics undermines the reliability and security of the nodes.

Furthermore, a similar issue pertains to the consensus layer. The `coordinator` implementation currently lacks support for light clients. Although this does not constitute a direct security vulnerability, supporting light clients would significantly enhance the network's reliability and decentralization.

Assets:

- Light client

Status: Fixed

Classification

Impact: 1/5

Likelihood: 1/5

Severity: Low

Recommendations

Remediation: To address these issues, it is recommended to fully implement light client and light server functionality across both the execution and consensus layers. This will ensure that light clients have equivalent access to data as full nodes, thereby bolstering the overall reliability and security of the network.

For guidance on implementing light clients in the `coordinator`, it is advisable to review the following issue, which tracks the progress of light client implementation in the `prysm` project. While leveraging the techniques implemented in `prysm` can significantly expedite the process and facilitate the support of light clients in the Waterfall project, the substantial logical differences between Waterfall and `prysm` make full integration less straightforward.

If supporting light nodes is not deemed necessary and is postponed, it is highly recommended to disable the corresponding `LightServeFlag` for light clients and `SyncModeFlag` for full nodes' light server functionality. This will prevent any attempts to run the light client or activate the light server on full nodes, thereby avoiding unexpected panics and enhancing the overall security posture of the project.

F-2024-3344 - Inadequate Sender Validation in Deposit Transaction Processing - Low

Description:

The issue arises from the current implementation of transaction validation for deposit operations within the execution layer. When a transaction to create a new deposit is received, several checks are performed within the `checkDepositOperation` function to ensure its validity. However, a significant oversight exists: the function does not verify that the `CreatorAddress` provided in the deposit data matches the transaction sender. While this check is unnecessary for existing validators, it is essential for the initial registration of a validator. This omission allows anyone to create a deposit transaction with an arbitrary `CreatorAddress`, generate a signature using their own BLS keys, and successfully execute the transaction.

Although this behavior is typically not incentivized, it can pose a threat under certain circumstances. Specifically, the described action prevents the legitimate owner of the `CreatorAddress` from running a validator, thereby compromising their ability to participate in the network.

Assets:

- core module [https://gitlab.waterfall.network/waterfall/protocol/gwat]

Status: Accepted

Classification

Impact: 1/5

Likelihood: 5/5

Severity: Low

Recommendations

Remediation:

To address this issue, it is recommended to update the `checkDepositOperation` function to include a verification step that ensures the transaction sender matches the `CreatorAddress` in the deposit data when registering a new validator. A possible solution is illustrated by the following code:

```
validator, err := pool.chain.ValidatorStorage().GetValidator(pool.cu
rrentState, op.CreatorAddress())
if err == valStore.ErrNoStateValidatorInfo {
if op.CreatorAddress() != from {
return valOperation.ErrInvalidCreator
}
```

```
    return nil
}
```

This additional check will prevent unauthorized creation of deposit transactions and protect the integrity of the validator registration process.

F-2024-3521 - Divergence from ERC-721 Standard in Token Implementation - Low

Description:

The current NFT token implementation in the `gwat` codebase deviates significantly from the ERC-721 standard, despite references implying support for it.

The primary issue is the absence of the `safeTransferFrom` method. Although some necessary methods for the `safeTransferFromOperation` type exist, the `safeTransferFrom` method itself remains commented out with the note `// TODO: Implement safeTransferFrom for NFTs`.

This deviation from the ERC-721 standard increases the likelihood of integration issues with other systems that expect standard-compliant behavior. As a result, the token implementation may face interoperability challenges, leading to potential errors and decreased reliability. Furthermore, such inconsistencies can hinder the adoption and trustworthiness of the token, making it more susceptible to bugs and security vulnerabilities.

Assets:

- token module [https://gitlab.waterfall.network/waterfall/protocol/gwat]

Status: Accepted

Classification

Impact: 1/5

Likelihood: 2/5

Severity: Low

Recommendations

Remediation:

To align the token implementation with the ERC-721 standard and enhance system reliability, the following actions are recommended:

Implement `safeTransferFrom`:

- Develop and integrate the `safeTransferFrom` method to fully support the ERC-721 standard.
- Ensure token transfers are safe and compatible with other ERC-721 compliant systems.

Update Documentation:

- Provide a detailed explanation of the current state of ERC-721 support.
- Include a comprehensive guide on the usage and limitations of the current token implementation.
- Outline the implementation roadmap, specifying timelines for completing the `safeTransferFrom` method and any other pending features.

Stakeholder Communication:

- Inform stakeholders about the current limitations and future improvements.
- Provide regular updates on the progress of implementing the `safeTransferFrom` method and other enhancements.

By following these recommendations, the project can ensure compliance with the ERC-721 standard, improving interoperability, reliability, and overall security of the token implementation.

F-2024-3522 - Incorrect Event Emitted During Token Operations - Low

Description:

In the current implementation of token-related methods, there are discrepancies in event emission, which could lead to confusion and lack of clarity regarding token transfers and approvals.

The `transferFrom` method emits an `ApprovalWrc20` event with an empty sender address. This lack of essential information about the address whose allowance has changed can be misleading and obscure.

The `buy` method affects balances and approvals, yet corresponding events are not triggered, leaving stakeholders uninformed about these changes.

Failure to emit accurate and informative events during token operations can lead to misunderstandings, hinder transparency, and impede the effective tracking of token-related activities.

Assets:

- token module [https://gitlab.waterfall.network/waterfall/protocol/gwat]

Status: Fixed

Classification

Impact: 1/5

Likelihood: 4/5

Severity: Low

Recommendations

Remediation:

It is advised to rectify the event emission in the mentioned methods to ensure accurate and informative event logs. This includes emitting the correct sender address in the `ApprovalWrc20` event and triggering events corresponding to changes in balances and approvals during the `buy` method execution. Implementing these changes will enhance transparency, facilitate accurate tracking of token-related activities, and foster better understanding among stakeholders.

Observation Details

F-2024-1866 - Remnants of the Ethereum Implementation - Info

Description:

Despite moving away from Ethereum implementations, the `coordinator` repository, originally forked from the `prysm` codebase, still retains certain redundancies.

Although the project no longer employs Proof-of-Work (PoW) consensus, remnants of PoW-related functionalities persist within the codebase. Notably, the files `coordinator/beacon-chain/blockchain/pow_block.go` and its associated test file continue to exist, despite being rendered obsolete. The sole function within `pow_block.go`, `validateTerminalBlockDifficulties`, remains unused and serves no practical purpose, contributing to code clutter and hindering the readability of the coordinator component.

Another concern is the presence of the `AltairForkEpoch` configuration for the beacon chain node. While this value may be relevant in the context of Ethereum 2.0, it is unnecessary for the Waterfall project and may lead to incorrect node behavior if misconfigured.

Furthermore, a significant portion of the code employs perplexing names containing "pow" and "eth1", despite these functionalities being reworked for project-specific requirements. The persistence of these unchanged names adds complexity and diminishes the readability of the codebase, particularly within the `powchain` package.

Assets:

- prysm [https://gitlab.waterfall.network/waterfall/protocol/coordinator]
- beacon-chain/powchain module [https://gitlab.waterfall.network/waterfall/protocol/coordinator]

Status: Accepted

Recommendations

Remediation:

To address these issues effectively, the following actions are recommended:

- Removal of both `coordinator/beacon-chain/blockchain/pow_block.go` and `coordinator/beacon-chain/blockchain/pow_block_test.go` from the project repository.
- Consideration of removing the `AltairForkEpoch` from node configuration, along with any code related to Beacon Chain Phase 0,

unless it serves a specific purpose for the Altair fork and remains relevant to the Waterfall project.
- Refactoring of confusing variable, method, function, and interface names associated with PoW functionalities, ensuring clarity and coherence throughout the codebase.
- Enhancement of documentation to provide comprehensive in-code explanations of functionalities, aiding developers in understanding and navigating the coordinator component effectively.

Implementing these recommendations will streamline the codebase, enhance clarity, and facilitate ease of development within the coordinator component.

F-2024-3375 - Test Coverage - Info

Description:

While the project benefits from a solid unit test coverage inherited from Go-Ethereum and Prysm, deficiencies in coverage have been identified. These shortcomings compromise the robustness of the codebase and elevate the risk of potential security vulnerabilities.

To determine the coverage, the following commands can be utilized:

```
go test ./... -coverprofile cover.out
go tool cover -func cover.out
```

The breakdown of code coverage across several selected folders that are pivotal in the `gwat` implementation is as follows:

Component	Coverage
core	47.8%
dag	9.3%
eth	26.7%
token	67.9%
validator	70.1%

It is noteworthy that **core** and **eth** folders, inherited from Go-Ethereum, have notably lower coverage compared to the current `geth` implementation, which stands at 62.0% for **core** and 51.9% for **eth**.

However, there is a notable issue within the test suite for the **token** package, as a majority of the tests are failing. The failed tests include:

- `TestProcessorTransferFromOperationCall`
- `TestProcessorMintOperationCall`
- `TestProcessorTransferOperationCall`
- `TestProcessorBurnOperationCall`
- `TestProcessorApprovalForAllCall`
- `TestProcessorIsApprovedForAll`
- `TestProcessorPropertiesWRC20`
- `TestProcessorApproveCall`

Additionally, another critical area of concern is the **dag** component, which represents a custom implementation for the BlockDAG logic. Deficiencies or bugs within this area could significantly undermine the network's correct operation.

While the test coverage for validator is satisfactory, it still require further enhancement to ensure comprehensive testing and robustness.

The code coverage breakdown for essential folders in the `coordinator` implementation is as follows:

Component	Coverage
beacon-chain	58.6%
validator	61.4%

Although these components maintain satisfactory coverage levels, there is room for improvement to ensure comprehensive testing of critical functionalities.

Assets:

- Test coverage

Status: Accepted

Recommendations

Remediation: To enhance the project's reliability and security, it is essential to strengthen the existing test suite.

Enhance Test Coverage: Expand the test suite to comprehensively cover critical functions. Emphasize implementing a substantial number of unit tests for the **dag** component to ensure the accuracy of the BlockDAG logic. Although **token** and **validator** coverages are adequate, there is still room for improvement.

Address Failing Tests: Resolve issues contributing to test failures, thereby ensuring a stable and reliable testing environment while increasing overall test coverage.

Ensure Comprehensive Testing: Ensure that all pivotal functions and components, particularly those integral to the BlockDAG logic, undergo thorough testing. This encompasses both the existing functionality and any future updates to the system.

Implementing these recommendations will elevate the project's overall code quality, align it with established best practices in software development, and enhance its reliability and security posture. Moreover, it will instill confidence in stakeholders and users alike, ensuring a robust and dependable system.

Appendix 1. Severity Definitions

Severity	Description
Critical	Vulnerabilities that can lead to a complete breakdown of the blockchain network's security, privacy, integrity, or availability fall under this category. They can disrupt the consensus mechanism, enabling a malicious entity to take control of the majority of nodes or facilitate 51% attacks. In addition, issues that could lead to widespread crashing of nodes, leading to a complete breakdown or significant halt of the network, are also considered critical along with issues that can lead to a massive theft of assets. Immediate attention and mitigation are required.
High	High severity vulnerabilities are those that do not immediately risk the complete security or integrity of the network but can cause substantial harm. These are issues that could cause the crashing of several nodes, leading to temporary disruption of the network, or could manipulate the consensus mechanism to a certain extent, but not enough to execute a 51% attack. Partial breaches of privacy, unauthorized but limited access to sensitive information, and affecting the reliable execution of smart contracts also fall under this category.
Medium	Medium severity vulnerabilities could negatively affect the blockchain protocol but are usually not capable of causing catastrophic damage. These could include vulnerabilities that allow minor breaches of user privacy, can slow down transaction processing, or can lead to relatively small financial losses. It may be possible to exploit these vulnerabilities under specific circumstances, or they may require a high level of access to exploit effectively.
Low	Low severity vulnerabilities are minor flaws in the blockchain protocol that might not have a direct impact on security but could cause minor inefficiencies in transaction processing or slight delays in block propagation. They might include vulnerabilities that allow attackers to cause nuisance-level disruptions or are only exploitable under extremely rare and specific conditions. These vulnerabilities should be corrected but do not represent an immediate threat to the system.

Appendix 2. Scope

The scope of the project includes the following components from the provided repository:

Scope Details	
Repository	https://gitlab.waterfall.network/waterfall/protocol/coordinator, https://gitlab.waterfall.network/waterfall/protocol/gwat
Commit	9b3570c351d927004e4c5da26f908fda5f6ce65a, 6782049b74c92e58a8ca077a3015ec67dc7ef943
Whitepaper	https://waterfall.network/wp-content/uploads/2023/09/Whitepaper.pdf

Components in Scope

coordinator

- security related issues reported in prysm after v2.1.1
- validator module
- beacon-chain/core module
- beacon-chain/blockchain module
- beacon-chain/rpc module
- beacon-chain/state module
- beacon-chain/p2p module
- beacon-chain/powchain module
- beacon-chain/forkchoice module
- beacon-chain/sync module

gwat

- security related issues reported in geth after v1.10.11
- core module
- eth module
- token module
- validator module