HACKEN

Blockchain Protocol Security Analysis Report

Customer: SSV Labs

Date: 01/10/2024

We express our gratitude to the SSV Labs team for the collaborative engagement that enabled the execution of this Blockchain Protocol Security Assessment.

SSV Network is a decentralized infrastructure designed to enhance the security and decentralization of Ethereum's Proof of Stake (PoS) mechanism. By leveraging Distributed Validator Technology (DVT), the network enables multiple nodes to collaboratively manage a single Ethereum validator, thereby reducing risks and boosting fault tolerance. This distribution of validator duties across various operators helps eliminate single points of failure, enhancing the security and resilience of Ethereum staking.

Document

Name	Blockchain Protocol Review and Security Analysis Report for SSV Labs
Audited By	Nino Lipartiia, Hamza Sajid
Approved By	Luciano Ciattaglia
Website	https://ssv.network/
Changelog	11/08/2024 - First Preliminary Report for ssv-spec
Changelog	05/09/2024 - Second Preliminary Report
Changelog	01/10/2024 - Final Report
Platform	Ethereum
Language	Golang
Tags	Distributed validator technology, MPC
Methodology	https://hackenio.cc/blockchain_methodology

Review Scope

Repository	https://github.com/ssvlabs/ssv-spec
Commit	ccf408d1ecd87a4ab631885409d679f10c3fd080

Audit Summary

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

3	3	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	2
Medium	0
Low	1

Vulnerability	Severity
F-2024-4371 - Vulnerabilities in External Go Dependencies	High
F-2024-4402 - Vulnerabilities in Go Standard Library	High
F-2024-4394 - Insecure Decryption of RSA Keys	Low

Documentation quality

- The code is well-documented, providing detailed explanations that support a strong foundational understanding.
- Protocol documentation is accessible via the official SSV Network website, ensuring easy reference for developers and users.
- Major updates to the protocol are documented through SSV Improvement Proposals (SIPs), offering transparency and clarity.
- However, minor discrepancies between the SIPs and the actual codebase have been noted.

Code quality

- The project maintains consistently high code quality across its components.
- There is comprehensive test coverage, which contributes to the project's overall reliability.
- Static code analysis has identified several warnings that should be addressed.
- The codebase includes some unresolved TODO comments and "implement me" panics, indicating areas that need further attention.

Architecture quality

- The project leverages the innovative Distributed Validator Technology, enhancing security and decentralization.
- The separation of the `ssv-spec` repository logic from the main SSV node repository improves modularity and maintainability.
- The `ssv-spec` components are well-organized into distinct packages, contributing to a clean and efficient architecture.

Table of Contents

System Overview

This audit focuses on the `ssv-spec` repository, a pivotal element of the SSV Network's operator logic. The `ssv-spec` codebase includes several critical modules:

- **p2p**: Oversees networking and message validation.
- **qbft**: Implements the consensus mechanism for achieving coordination among the operators within the cluster.
- **ssv**: Contains the core logic of the SSV specification.

These modules are essential for the effective operation of SSV nodes, as they define the specific requirements for crucial node components and significantly enhance overall maintainability. This audit is dedicated to evaluating the code within the `ssv-spec` repository to ensure its integrity and functionality.

Findings

Vulnerability Details

F-2024-4371 - Vulnerabilities in External Go Dependencies - High

Description:

Security analysis using `govulncheck` and `nancy` identified several vulnerabilities in external libraries. Specifically, 5 vulnerabilities were found in `go-ethereum`, and 1 each in `protobuf` and `go-libp2p`. While some vulnerabilities may not pose immediate risks, others could significantly compromise the project's security posture.

Key Vulnerabilities Identified:

GO-2024-2819: Denial of Service via malicious p2p message

- Issue: A vulnerable node can be made to consume very large amounts of memory when handling specially crafted p2p messages sent from an attacker node.
- Found in: github.com/ethereum/go-ethereum@v1.12.0
- Fixed in: github.com/ethereum/go-ethereum@v1.13.15
- CVSS Score: 7.5 (High)
- Details: pkg.go.dev/vuln/GO-2024-2819

GO-2023-2046: Unbounded Memory Consumption

- Issue: This vulnerability results in unbounded memory consumption, potentially leading to system crashes and instability.
- Found in: github.com/ethereum/go-ethereum@v1.12.0
- Fixed in: github.com/ethereum/go-ethereum@v1.12.1
- CVSS Score: 7.5 (High)
- Details: pkg.go.dev/vuln/GO-2023-2046

CVE-2022-23328: Uncontrolled Resource Consumption (Resource Exhaustion)

- Issue: A design flaw in all versions of Go-Ethereum allows an attacker node to send 5,120 pending transactions of a high gas price from one account. These transactions fully spend the account's balance to a victim Geth node, purging all pending transactions in the victim's memory pool and occupying it, preventing new transactions from entering. This results in a denial of service (DoS).
- CVSS Score: 7.5 (High)
- Details: CVE-2022-23328

CVE-2023-42319: Denial of Service via GraphQL

- Issue: Geth through version 1.13.4, when using `--http --graphql`, allows remote attackers to cause memory consumption and daemon hang through crafted GraphQL queries. The vendor notes the GraphQL endpoint is not designed to handle hostile client attacks or high traffic.
- CVSS Score: 7.5 (High)
- Details: CVE-2023-42319

CVE-2022-37450: Riskless Uncle Making (RUM)

- Issue: Go Ethereum (geth) through version 1.10.21 allows attackers to increase rewards by mining blocks in specific situations using time-difference manipulation, achieving main-chain block replacement, known as Riskless Uncle Making (RUM).
- CVSS Score: 5.9 (Medium)
- Details: CVE-2022-37450

CVE-2024-24786: Loop with Unreachable Exit Condition (Infinite Loop)

- Issue: The `protojson.Unmarshal` function can enter an infinite loop when processing invalid JSON, leading to DoS.
- CVSS Score: 7.5 (High)
- Fixed in: v1.33.0
- Details: CVE-2024-24786

CVE-2023-39533: CWE-770: Allocation of Resources Without Limits or Throttling

- Issue: A malicious peer can use large RSA keys to run a resource exhaustion attack & force a node to spend time doing signature verification of the large key.
- CVSS Score: 7.5 (High)
- Fixed in: v0.29.1
- Details: CVE-2023-39533

Assets:

- Dependencies

Status: Fixed

Classification

Impact: 4/5

Likelihood: 4/5

Severity: High

Recommendations

Remediation: Given the high severity of the vulnerabilities identified in the dependencies of `ssv-spec`, it is imperative to update the following libraries as soon as possible:

- **go-ethereum:** Upgrade to the latest version, or at a minimum, version 1.13.15.
- **protobuf:** While this is an indirect dependency, updating the `libp2p` module, which utilizes `protobuf`, will address the issue.
- **go-libp2p:** Upgrade to the latest version, or at a minimum, version 0.29.1. Additionally, to fully resolve the issue, it is necessary to update the Go compiler to version 1.20.7 or later.

Implementing these updates will significantly improve the security posture of the project.

F-2024-4402 - Vulnerabilities in Go Standard Library - High

Description:

Our security analysis using the `govulncheck` tool has identified multiple vulnerabilities within the standard library dependencies specified by Go version 1.20 in the project's `go.mod` file. These vulnerabilities range from memory exhaustion risks and certificate verification issues to denial of service (DoS) vectors and incorrect results.

Key Vulnerabilities Identified:

GO-2024-2687: HTTP/2 CONTINUATION flood in net/http

- Issue: An attacker may cause an HTTP/2 endpoint to read arbitrary amounts of header data by sending an excessive number of CONTINUATION frames.
- Found in: net/http@go1.20
- Affected Versions: before go1.21.9, from go1.22.0-0 before go1.22.2
- CVSS Score: 5.2 (Moderate)
- Details: pkg.go.dev/vuln/GO-2024-2687

GO-2024-2598: Verify panics on certificates with an unknown public key algorithm in crypto/x509

- Issue: Verifying a certificate chain which contains a certificate with an unknown public key algorithm will cause `Certificate.Verify` to panic.
- Found in: crypto/x509@go1.20
- Affected Versions: before go1.21.8, from go1.22.0-0 before go1.22.1
- Details: pkg.go.dev/vuln/GO-2024-2598

GO-2023-2185: Insecure parsing of Windows paths with a ??\ prefix in path/filepath

- Issue: The filepath package does not recognize paths with a ??\ prefix as special.
- Found in: internal/safefilepath@go1.20
- Affected Versions: before go1.20.11, from go1.21.0-0 before go1.21.4
- CVSS Score: 7.5 (High)
- Platforms: windows
- Details: pkg.go.dev/vuln/GO-2023-2185

GO-2023-1987: Large RSA keys can cause high CPU usage in crypto/tls

- Issue: Extremely large RSA keys in certificate chains can cause a client/server to expend significant CPU time verifying signatures.
- Found in: crypto/tls@go1.20

- Affected Versions: before go1.19.12, from go1.20.0-0 before go1.20.7, from go1.21.0-0 before go1.21.0-rc.4
- CVSS Score: 5.3 (Moderate)
- Details: pkg.go.dev/vuln/GO-2023-1987

GO-2023-1840: Unsafe behavior in setuid/setgid binaries in runtime

- Issue: On Unix platforms, the Go runtime does not behave differently when a binary is run with the setuid/setgid bits.
- Found in: runtime@go1.20
- Affected Versions: before go1.19.10, from go1.20.0-0 before go1.20.5
- CVSS Score: 7.8 (High)
- Details: pkg.go.dev/vuln/GO-2023-1840

GO-2023-1621: Incorrect calculation on P256 curves in crypto/internal/nistec

- Issue: The `ScalarMult` and `ScalarBaseMult` methods of the P256 Curve may return an incorrect result.
- Found in: crypto/internal/nistec@go1.20
- Affected Versions: before go1.19.7, from go1.20.0-0 before go1.20.2\
- CVSS Score: 5.3 (Moderate)
- Details: pkg.go.dev/vuln/GO-2023-1621

GO-2023-1570: Panic on large handshake records in crypto/tls

- Issue: Large handshake records may cause panics in crypto/tls. Both clients and servers may send large TLS handshake records which cause servers and clients, respectively, to panic when attempting to construct responses.
- Found in: crypto/tls@go1.20
- Affected Versions:before go1.19.6, from go1.20.0-0 before go1.20.1
- CVSS Score: 7.5 (High)
- Details: pkg.go.dev/vuln/GO-2023-1570

GO-2023-1568: Path traversal on Windows in path/filepath

- Issue: On Windows, the `filepath.Clean` function could transform an invalid path such as "a/../c:/b" into the valid path "c:\b".
- Found in: path/filepath@go1.20
- Affected Versions: before go1.19.6, from go1.20.0-0 before go1.20.1
- Platforms: windows
- CVSS Score: 7.5 (High)
- Details: pkg.go.dev/vuln/GO-2023-1568

Vulnerable Dependencies Affecting Tests Only:

GO-2024-2599: Memory exhaustion in multipart form parsing in net/textproto and net/http

- Issue: When parsing a multipart form, limits on the total size of the parsed form were not applied to the memory consumed while reading a single form line.
- Found in: net/textproto@go1.20
- Affected Versions: before go1.21.8, from go1.22.0-0 before go1.22.1
- Details: pkg.go.dev/vuln/GO-2024-2599

GO-2023-1705: Excessive resource consumption in net/http, net/textproto and mime/multipart

- Issue: Multipart form parsing can consume large amounts of CPU and memory when processing form inputs containing very large numbers of parts.
- Found in: mime/multipart@go1.20
- Affected Versions: before go1.19.8, from go1.20.0-0 before go1.20.3
- Details: pkg.go.dev/vuln/GO-2023-1705

GO-2023-1704: Excessive memory allocation in net/http and net/textproto

- Issue: HTTP and MIME header parsing can allocate large amounts of memory, even when parsing small inputs, potentially leading to a denial of service.
- Found in: net/textproto@go1.20
- Affected Versions: before go1.19.8, from go1.20.0-0 before go1.20.3
- Details: pkg.go.dev/vuln/GO-2023-1704

While some of these vulnerabilities might not affect the codebase, others, such as panics during certificate verification, high CPU usage due to large RSA keys, or incorrect calculations on P256 curves, pose significant risks.

Assets:

- Dependencies

Status: Fixed

Classification

Impact: 4/5

Likelihood: 4/5

Severity: High

Recommendations

Remediation: Given the severity of the identified vulnerabilities within the Go standard library dependencies specified by the project's use of Go version 1.20, urgent action is required to safeguard the integrity and availability of the network. The following measures are strongly recommended:

- **Immediate Upgrade**: Consider upgrading to the latest stable Go version, 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. Additionally, to fully resolve the issues, utilize the Go compiler version 1.22.2 or later.
- **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.
- **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-2024-4394 - Insecure Decryption of RSA Keys - Low

Description:

The issue originates from the implementation of the `PemToPublicKey` and `PemToPrivateKey` functions, which are intended to decrypt bytes into `rsa.PublicKey` and `rsa.PrivateKey`, respectively. These functions heavily rely on the deprecated and insecure methods `x509.IsEncryptedPEMBlock` and `x509.DecryptPEMBlock`.

The deprecation is rooted in the RFC 1423 encryption format, which is considered legacy and insecure by design. Consequently, the `IsEncryptedPEMBlock`, `DecryptPEMBlock`, and `EncryptPEMBlock` methods in the `x509` package, which refer to this encryption format, compromise security and can lead to issues such as false negative results.

Another reason for the deprecation is the use of the `deriveKey` method within `DecryptPEMBlock`, which employs the insecure MD5 hash function. Although this does not necessarily imply a vulnerability in the context of `DecryptPEMBlock`, the security of using MD5 in this method has not been confirmed by the Go team, leaving it as a potential security risk.

```
// PemToPrivateKey return rsa private key from pem
func PemToPrivateKey(skPem []byte) (*rsa.PrivateKey, error) {
block, _ := pem.Decode(skPem)
// nolint
enc := x509.IsEncryptedPEMBlock(block)
b := block.Bytes
if enc {
var err error
// nolint
b, err = x509.DecryptPEMBlock(block, nil)
if err != nil {
return nil, errors.Wrap(err, "Failed to decrypt private key")
}
}
parsedSk, err := x509.ParsePKCS1PrivateKey(b)
if err != nil {
return nil, errors.Wrap(err, "Failed to parse private key")
}
return parsedSk, nil
}

// PemToPublicKey return rsa public key from pem
func PemToPublicKey(pkPem []byte) (*rsa.PublicKey, error) {
block, _ := pem.Decode(pkPem)
// nolint
enc := x509.IsEncryptedPEMBlock(block)
b := block.Bytes
if enc {
var err error
// nolint
b, err = x509.DecryptPEMBlock(block, nil)
if err != nil {
return nil, errors.Wrap(err, "Failed to decrypt private key")
}
}
parsedPk, err := x509.ParsePKIXPublicKey(b)
if err != nil {
return nil, errors.Wrap(err, "Failed to parse public key")
}
if ret, ok := parsedPk.(*rsa.PublicKey); ok {
return ret, nil
}
```

```
    return nil, errors.Wrap(err, "Failed to parse public key")
}
```

Although these discrepancies are unlikely to pose critical security risks, they have the potential to lead to unexpected behaviors and failures.

Assets:

- Cryptography

Status: Fixed

Classification

Impact: 2/5

Likelihood: 1/5

Severity: Low

Recommendations

Remediation: To address the issue and bolster the project's security, the following steps are recommended:

- **Replace Deprecated Methods**: Update the `PemToPublicKey` and `PemToPrivateKey` functions to use modern and secure alternatives such as AES-GCM for encryption and PKCS#8 with PBES2 for private key storage.
- **Implement Password Protection**: Ensure that proper password protection is enforced when decrypting PEM blocks to enhance security.
- **Regular Security Audits**: Conduct regular security audits and code reviews to identify and mitigate potential vulnerabilities in cryptographic implementations.
- **Follow Best Practices**: Adhere to industry best practices and guidelines for secure cryptographic operations and key management.

Observation Details

F-2024-4814 - Potential Issues Identified Via Static Analysis - Info

Description:

During the static analysis using the `gosec` tool, several potential issues were identified in the `ssv-spec` codebase:

- **Weak Random Number Generator**: In `shuffle.go:31`, a weak random number generator (`math/rand`) is used instead of a cryptographically secure one.
- **Potential File Inclusion via Variable**: In `helpers.go:49,67` and `controller_spectest.go:212`, potential file inclusion vulnerabilities were identified due to variable-based file paths.
- **Implicit Memory Aliasing**: In `test.go:81`, there is a risk of implicit memory aliasing within a `for` loop.
- **Insecure File Permissions**: Multiple files (`main.go` in various directories) use `os.WriteFile` with permissions set to `0644`, which is considered insecure.

While these warnings do not present immediate security threats, they highlight areas for improvement. Moreover, false positive warnings can obscure real issues that may arise in the future.

Assets:

- Code quality

Status: Fixed

Recommendations

Remediation:

To address these issues, carefully investigate each warning and resolve them individually to mitigate potential risks effectively.

However, it's important to recognize that there may be situations where `gosec` could return false positives. In such cases, using `nosec` comments might be beneficial to suppress these rules intentionally.

When using `nosec`, ensure that it is applied judiciously and includes a proper justification. The `nosec` directive can be added to specific lines of code that are deemed safe despite the security warning. For more detailed guidance on using `nosec`, refer to the gosec documentation.

F-2024-4837 - Unnecessary Exploitation of nolint Comments - Info

Description:

During the audit, it was observed that the codebase of `ssv-spec` includes unnecessary usage of `nolint` comments, which suppress linter warnings. Upon removing these comments and running the `golangci-lint` static analysis tool, the following warnings were revealed:

Deprecated Functions: In `types/encryption.go:53,58,74,79`, the code utilizes deprecated functions `x509.IsEncryptedPEMBlock` and `x509.DecryptPEMBlock`. These functions have been deprecated since Go 1.16 due to security concerns associated with legacy PEM encryption, which is vulnerable to padding oracle attacks. For more details, see issue *F-2024-4394*.

Unused Variables: In `qbft/timeout.go:10,11,12`, the variables `quickTimeoutThreshold`, `quickTimeout`, and `slowTimeout` are declared but not used. This indicates unnecessary code that should be removed or revised to enhance clarity and maintainability.

Assets:

- Code quality

Status: Fixed

Recommendations

Remediation:

While using `nolint` to suppress unnecessary warnings is not inherently a bad practice, it should be approached with more caution. To address this issue, it is recommended to:

- **Reevaluate Each nolint Usage**: Carefully reconsider the necessity of each `nolint` comment.
- **Resolve Lint Warnings**: Address the lint warnings identified by static analysis tools.
- **Provide Specificity and Justification**: For cases where `nolint` is still deemed appropriate, specify the exact linter to be ignored and include comments explaining the necessity of `nolint`.

Implementing these recommendations will improve code quality and robustness.

F-2024-4892 - Runners Implementations Lack Share Length Validation - Info

Description:

The problem involves the handling of the `Share` map in several runner types. Specifically, types such as `AggregatorRunner`, `ProposerRunner`, `SyncCommitteeAggregatorRunner`, `ValidatorRegistrationRunner`, and `VoluntaryExitRunner` assume that their `BaseRunner` contains only a single `Share`.

This assumption is not explicitly validated. Consequently, some methods in the `ssv-spec` package operate under the expectation that there is only one `Share`, using only the first entry in the map and ignoring others.

This inconsistency leads to confusing logic, as the code relies on implicit assumptions rather than explicit validation. It diminishes code readability and maintainability, increasing the likelihood of bugs and making future modifications more error-prone.

Assets:

- spec-ssv

Status: Fixed

Recommendations

Remediation:

To address this issue and improve code robustness, the following actions are recommended:

- **Validation Check**: For runners expected to handle a single `Share` (such as in `NewAggregatorRunner`), implement a validation step to ensure that the `Share` map contains exactly one entry. This will enforce the assumption and help detect discrepancies early.
- **Enhanced Documentation**: Improve the code documentation to clearly specify scenarios where only one `Share` is expected. Highlight functions and methods that depend on this condition to avoid misunderstandings and reduce the risk of incorrect implementations.

These changes will enhance code clarity, enforce proper assumptions, and mitigate the likelihood of future errors.

F-2024-5171 - Unnecessary Error Return in basePartialSigMsgProcessing - Info

Description:

The `basePartialSigMsgProcessing` function currently includes an error return value that is always set to `nil`. This function is designed to add validated partial signature messages to a container, check for a quorum, and return a boolean indicating whether a quorum exists, along with the corresponding roots. However, since no errors are generated within the function, the inclusion of the error return value serves no practical purpose.

This redundant error return complicates the function's interface unnecessarily, adding clutter and potential confusion for developers maintaining the code.

Assets:

- Code quality

Status: Fixed

Recommendations

Remediation:

- **Remove the Error Return Value**: Simplify the function's interface by eliminating the error return from the function signature.
- **Refactor Dependent Code**: Update all instances where this function is called to remove the unused error return value.

By removing this redundant error return, the function will become more streamlined, and the overall codebase will be cleaner and easier to maintain. This adjustment also minimizes the potential for confusion or mistakes in future code modifications.

F-2024-5173 - Excessive Validation in Decided Messages Handling - Info

Description: The `ValidateDecided` function performs redundant validation of messages. Specifically, the function includes two separate calls to `msg.Validate()`, even though the function `baseCommitValidationVerifySignature`, invoked within `ValidateDecided`, already performs a similar validation through `baseCommitValidationIgnoreSignature`. Since `baseCommitValidationVerifySignature` includes a validation step, the additional `msg.Validate()` calls within `ValidateDecided` introduce unnecessary computational overhead without providing any incremental benefit to security or correctness.

Assets:

- spec-qbft

Status: Fixed

Recommendations

Remediation: To improve performance and code quality, remove the two redundant `msg.Validate()` calls from the `ValidateDecided` function. This will streamline the function, reduce computational overhead, and enhance overall code efficiency and maintainability.

F-2024-5698 - Residual DKG Code Segments - Info

Description:

During the audit, it was observed that remnants of the Distributed Key Generation (DKG) logic, initially intended for implementation within the SSV-Spec repository but later moved to a separate repository, still remain within the current codebase.

Examples of the outdated code can be found in the following paths:

- *types/messages.go:83*
- *types/signer.go:21*
- *README.md:79*

The presence of this obsolete code can lead to confusion, maintenance difficulties, and potential integration issues. Moreover, it adds unnecessary clutter, hindering code readability and overall project maintainability.

Assets:

- Code quality

Status: Fixed

Recommendations

Remediation:

It is advisable to remove the remaining DKG-related code from the repository to ensure a clean and well-organized codebase. This will enhance clarity, improve the ease of maintenance, and ensure that the repository accurately reflects the actively used components.

Disclaimers

Hacken Disclaimer

The blockchain protocol given for audit has been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in the protocol source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other protocol statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the blockchain protocol.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Blockchain protocols are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the protocol can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited blockchain protocol.

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://github.com/ssvlabs/ssv-spec
Commit	ccf408d1ecd87a4ab631885409d679f10c3fd080

The remediation check has been conducted based on commit hash 93ad50e, which reflects the status of each issue following this process. It is important to acknowledge that this commit may include changes made subsequent to the initial review commit, which were not part of the audit assessment.

Components in Scope

The scope includes the entire codebase of the `ssv-spec` repository, focusing on the following key components:

- **p2p**: Handles networking and message validation.
- **qbft**: Implements the consensus mechanism.
- **ssv**: Contains the core logic for the SSV spec.