HACKEN

Blockchain Protocol Security Analysis Report

Customer: SSV Labs

Date: 15/10/2024

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	05/09/2024 - Preliminary Report
Changelog	15/10/2024 - 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
Commit	20dba00cb02f98e52124158ad36c0bb28839f8a4

Audit Summary

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

8	7	0	1
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	2
Medium	2
Low	4

Vulnerability	Severity
F-2024-5437 - Vulnerabilities in Go Standard Library	High
F-2024-5438 - Vulnerabilities in External Go Dependencies	High
F-2024-5608 - Erroneous Threshold Logic in Key Splitting Mechanism	Medium
F-2024-5656 - Incomplete Validator Exit Logic Due to OwnValidator Oversight	Medium
F-2024-5502 - Insufficient Processing of Operator Removal Events	Low
F-2024-5505 - Deprecated Elliptic Curve Cryptography	Low
F-2024-5790 - Build Failure Due to Outdated dnsutils Package	Low
F-2024-5833 - Deficient Validation of Operator Public Key Uniqueness	Low

Documentation quality

- Protocol documentation is readily accessible via the official SSV Network website, facilitating easy reference for both developers and users.
- Major updates to the protocol are documented through SSV Improvement Proposals (SIPs), ensuring transparency and providing clarity on changes.
- The `ssv-spec` offers a comprehensive foundation and specification for the node codebase.
- While the README file and building documentation are present, they are not fully up to date and may require further revision to reflect the latest developments.

Code quality

- The project maintains a high standard of code quality across its components.
- Comprehensive test coverage is in place, which enhances the project's reliability and stability.
- Static code analysis has flagged several warnings that need to be addressed to ensure code robustness.
- The codebase contains a notable number of unresolved `TODO` comments, highlighting areas that require additional attention.

Architecture quality

- The project leverages the innovative Distributed Validator Technology, enhancing security and decentralization.
- The `ssv-spec` repository supports the clarity and maintainability of the node codebase, serving as a robust foundation for its development.
- The codebase maintains modularity, enhancing both maintainability and scalability.
- The cluster size is currently limited to 4, 7, 10, or 13 operators, as specified in the implementation.
- The node architecture is capable of supporting further scaling without necessitating a complete rewrite of the codebase.

Table of Contents

System Overview

SSV Network is a decentralized infrastructure designed to optimize the operation of Ethereum validators by enabling distributed validator technology. The network is engineered to ensure security, fault tolerance, and decentralization.

- **Consensus Mechanism:**
  SSV Network employs the QBFT (Quorum Byzantine Fault Tolerance) consensus algorithm. This mechanism facilitates the collective management of a single validator by multiple operators, reducing the risk of a single point of failure and enhancing the overall security of the staking process.
- **Operator Cluster Coordination:**
  The SSV protocol enables coordination among a cluster of operators who share control over a validator's key. This shared responsibility ensures that no single operator can compromise the validator, significantly improving the robustness and security of Ethereum staking.
- **Slashing Protection:**
  SSV Network includes a robust slashing protection mechanism, designed to prevent double-signing and other activities that could lead to slashing. This feature is crucial for maintaining the integrity of validators and avoiding slashing incidents.
- **Ethereum Compatibility:**
  SSV Network maintains compatibility with the Ethereum ecosystem, supporting the latest version (Deneb, as of the writing of this report).

Risks

- The SSV node is heavily reliant on events emitted by the smart contract, which governs the operators and validators within the SSV Network. However, since the smart contract review falls outside the scope of this audit, the accuracy and reliability of these events cannot be fully verified in this context.

Findings

Vulnerability Details

F-2024-5437 - Vulnerabilities in Go Standard Library - High

Description:

During the static analysis of the node's main repository, the `govulncheck` tool identified several potential vulnerabilities within the Go standard library. The current project utilizes Go version 1.20, which is susceptible to various security risks that have been addressed in later versions. These vulnerabilities include memory exhaustion risks, certificate verification issues, denial of service (DoS) vectors, and potential for incorrect results, all of which could lead to discrepancies and security flaws in the system.

The following vulnerabilities have been identified:

GO-2024-2963: Denial of service due to improper 100-continue handling in net/http

- Affected versions: before go1.21.12, from go1.22.0-0 before go1.22.5
- More info: GO-2024-2963

GO-2024-2887: Unexpected behavior from Is methods for IPv4-mapped IPv6 addresses in net/netip

- Affected versions: before go1.21.11, from go1.22.0-0 before go1.22.4
- More info: GO-2024-2887

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

- Affected versions: before go1.21.9, from go1.22.0-0 before go1.22.2
- CVSS Score: 6.3 (Medium)
- More info: GO-2024-2687

GO-2024-2610: Errors returned from JSON marshaling may break template escaping in html/template

- Affected versions: before go1.21.8, from go1.22.0-0 before go1.22.1
- More info: GO-2024-2610

GO-2024-2600: Incorrect forwarding of sensitive headers and cookies on HTTP redirect in net/http

- Affected versions: before go1.21.8, from go1.22.0-0 before go1.22.1
- More info: GO-2024-2600

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

- Affected versions: before go1.21.8, from go1.22.0-0 before go1.22.1
- More info: GO-2024-2599

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

- Affected versions: before go1.21.8, from go1.22.0-0 before go1.22.1
- More info: GO-2024-2598

GO-2023-2382: Denial of service via chunk extensions in net/http

- Affected versions: before go1.20.12, from go1.21.0-0 before go1.21.5
- CVSS Score: 5.3 (Medium)
- More info: GO-2023-2382

GO-2023-2186: Incorrect detection of reserved device names on Windows in path/filepath

- Affected versions: before go1.20.11, from go1.21.0-0 before go1.21.4
- More info: GO-2023-2186

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

- Affected versions: before go1.20.11, from go1.21.0-0 before go1.21.4
- CVSS Score: 7.5(High)
- More info: GO-2023-2185

GO-2023-2102: HTTP/2 rapid reset can cause excessive work in net/http

- Affected versions: before go1.20.10, from go1.21.0-0 before go1.21.3
- CVSS Score: 7.5(High)
- More info: GO-2023-2102

GO-2023-2043: Improper handling of special tags within script contexts in html/template

- Affected versions: before go1.20.8, from go1.21.0-0 before go1.21.1
- CVSS Score: 6.1 (Medium)
- More info: GO-2023-2043

GO-2023-2041: Improper handling of HTML-like comments in script contexts in html/template

- Affected versions: before go1.20.8, from go1.21.0-0 before go1.21.1
- CVSSScore: 6.1 (Medium)
- More info: GO-2023-2041

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

- 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 (Medium)
- More info: GO-2023-1987

GO-2023-1878: Insufficient sanitization of Host header in net/http

- Affected versions: before go1.19.11, from go1.20.0-0 before go1.20.6
- More info: GO-2023-1878

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

- Affected versions: before go1.19.10, from go1.20.0-0 before go1.20.5
- CVSS Score: 6.5 (Medium)
- More info: GO-2023-1840

GO-2023-1753: Improper handling of empty HTML attributes in html/template

- Affected versions: before go1.19.9, from go1.20.0-0 before go1.20.4
- CVSS Score: 7.3 (High)
- More info: GO-2023-1753

GO-2023-1752: Improper handling of JavaScript whitespace in html/template

- Affected versions: before go1.19.9, from go1.20.0-0 before go1.20.4
- CVSS Score: Critical (9.8)
- More info: GO-2023-1752

GO-2023-1751: Improper sanitization of CSS values in html/template

- Affected versions: before go1.19.9, from go1.20.0-0 before go1.20.4
- More info: GO-2023-1751

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

- Affected versions: before go1.19.8, from go1.20.0-0 before go1.20.3
- CVSS Score: High (7.5)
- More info: GO-2023-1705

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

- Affected versions: before go1.19.8, from go1.20.0-0 before go1.20.3
- CVSS Score: High (7.5)
- More info: GO-2023-1704

GO-2023-1703: Backticks not treated as string delimiters in html/template

- Affected versions: before go1.19.8, from go1.20.0-0 before go1.20.3
- CVSS Score: Critical (9.8)
- More info: GO-2023-1703

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

- Affected versions: before go1.19.7, from go1.20.0-0 before go1.20.2
- CVSS Score: Medium (5.3)
- More info: GO-2023-1621

GO-2023-1571: Denial of service via crafted HTTP/2 stream in net/http and golang.org/x/net

- Affected versions: before go1.19.6, from go1.20.0-0 before go1.20.1
- CVSS Score: High (7.5)
- More info: GO-2023-1571

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

- Affected versions: before go1.19.6, from go1.20.0-0 before go1.20.1
- CVSS Score: High (7.5)

- More info: GO-2023-1570

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

- Affected versions: before go1.19.6, from go1.20.0-0 before go1.20.1
- More info: GO-2023-1568

These vulnerabilities, identified within the Go standard library, have been addressed in later versions of Go, and it is crucial to update to mitigate the associated risks.

Assets:

- Dependencies

Status: Fixed

Classification

Impact: 4/5

Likelihood: 3/5

Severity: High

Recommendations

Remediation: To address the identified vulnerabilities and enhance the security of the system, the following actions are recommended:

Upgrade to the Latest Stable Go Version:

- It is highly recommended to upgrade to the latest stable Go version, currently Go 1.23, as it resolves all identified vulnerabilities and incorporates additional security enhancements. This upgrade will protect your system against known threats present in the current Go version (1.20).

Interim Update to Go 1.21.12:

- If upgrading to Go 1.23 is not feasible, use Go version 1.21.12 for building your project. This version includes crucial fixes for many vulnerabilities present in earlier Go versions. Additionally, update the `README.md` file to specify Go 1.21.12 as the required version for building the project. Provide clear instructions on this requirement and highlight the potential risks of not using this version.

Apply Security Patches:

- For dependencies that cannot be updated immediately due to compatibility or other constraints, apply all available security patches or implement workarounds to mitigate the risks associated with the vulnerabilities. This should be considered a temporary measure until a full upgrade can be completed.

Conduct Security Audits Post-Upgrade:

- After upgrading, perform thorough security audits to ensure that the system is functioning correctly and that no new vulnerabilities have been introduced. This step is crucial to confirm that the system's security posture has been enhanced.

Continuous Monitoring:

- Implement continuous monitoring to detect and respond to potential security threats in real time. Regularly check for updates to Go and its associated libraries to ensure that the system remains protected against newly discovered vulnerabilities.

By following these recommendations, the integrity and security of the system can be significantly improved, reducing the risk of exploitation from known vulnerabilities in the Go standard library.

F-2024-5438 - Vulnerabilities in External Go Dependencies - High

Description:

A comprehensive security analysis using the `govulncheck` and `nancy` tools identified several vulnerabilities in the external libraries utilized by the project. These vulnerabilities affect direct and indirect dependencies, including `protobuf`, `quic-go`, `go-ethereum`, and `btcec`.

Direct dependencies:

go-ethereum@v1.13.4

- go-ethereum vulnerable to denial of service via crafted GraphQL query
  - Issues: Geth, when `--http --graphql` is used, allows remote attackers to cause a denial of service (memory consumption and daemon hang) via a crafted GraphQL query
  - Affected versions: <= 1.13.4
  - CVSS Score: 7.5 (High)
  - Details: https://github.com/advisories/GHSA-v9jh-j8px-98vq
- go-ethereum vulnerable to DoS via malicious p2p message
  - Issues: A vulnerable node can be made to consume very large amounts of memory when handling specially crafted p2p messages sent from an attacker node.
  - Affected versions: < 1.13.15
  - CVSS Score: 7.5 (High)
  - Details: https://github.com/advisories/GHSA-4xc9-8hmq-j652

btcec/v2@v2.3.2

- Issue: btcd before 0.24.0 does not correctly implement the consensus rules outlined in BIP 68 and BIP 112, making it susceptible to consensus failures.
- Affected version: < 0.24.0
- Severity: Moderate
- Details: https://github.com/advisories/GHSA-3jgf-r68h-xfqm

Indirect dependencies:

protobuf@v1.30.0

- Issue: The `protojson.Unmarshal` function can enter an infinite loop when unmarshaling certain forms of invalid JSON.
- Affected version: < v1.33.0
- Severity: Moderate
- Details: https://github.com/advisories/GHSA-8r3f-844c-mc37

quic-go/quic-go@v0.33.0

- QUIC's Connection ID Mechanism vulnerable to Memory Exhaustion Attack

    - Issues: In the current version an attacker can cause its peer to run out of memory sending a large number of `NEW_CONNECTION_ID` frames that retire old connection IDs.
    - Affected version: < 0.42.0
    - CVSS Score: 7.5 (High)
    - Details: https://github.com/advisories/GHSA-c33x-xqrf-c478
- quic-go's path validation mechanism can be exploited to cause denial of service
    - Issues: An attacker can cause its peer to run out of memory sending a large number of `PATH_CHALLENGE` frames.
    - Affected version: before v0.37.7, from v0.38.0 before v0.38.2, from v0.39.0 before v0.39.4, from v0.40.0 before v0.40.1
    - CVSS Score: 6.4 (Moderate)
    - Details: https://github.com/advisories/GHSA-ppxx-5m9h-6vxf

Assets:

- Dependencies

Status: Fixed

Classification

Impact: 4/5

Likelihood: 3/5

Severity: High

Recommendations

Remediation: Given the high severity of the vulnerabilities identified, it is strongly recommended that the following libraries be updated promptly:

1. **go-ethereum@v1.13.4**: Upgrade to the latest version or at a minimum to version 1.13.5 or later.
2. **btcec/v2@v2.3.2**: Upgrade to a version that uses `btcd` version 0.24.0 or greater to ensure compliance with consensus rules.
3. **protobuf@v1.30.0 & quic-go/quic-go@v0.33.0:** Although these are indirect dependencies, the vulnerabilities can be addressed by updating the `libp2p` module, which utilizes `protobuf` and `quic-go`. Updating this module will resolve the associated vulnerabilities.

Implementing these updates will substantially enhance the security posture of the project and protect against potential exploits.

F-2024-5608 - Erroneous Threshold Logic in Key Splitting Mechanism - Medium

Description:

The core issue resides in the implementation of the `create-threshold` command, specifically in the following line:

cli/threshold.go:44

```
privKeys, err := threshold.Create(baseKey.Serialize(), keysCount-1,
keysCount)
```

This function is designed to split a private key into shares based on a specified threshold and the total number of keys. However, the current logic incorrectly sets the threshold as `keysCount-1`, which conflicts with the expected calculation for a `3f+1` cluster size, where the correct threshold should be `2f+1`.

While this calculation works correctly when the fault tolerance (`f`) equals 1 (i.e., a cluster size of 4), it fails for other supported cluster sizes. The protocol currently supports cluster sizes of 4, 7, 10, and 13, meaning the `create-threshold` command does not function correctly for the newly supported sizes.

This miscalculation could have significant consequences, particularly when a sufficient number of operators reach consensus but are unable to sign the required data due to the incorrect threshold. This failure could prevent the validator from performing its duties, potentially compromising the network's integrity.

Assets:

- Cryptography

Status: Fixed

Classification

Impact: 2/5

Likelihood: 3/5

Severity: Medium

Recommendations

Remediation: To rectify this issue, it is imperative to implement the correct threshold calculation within the `create-threshold` command. The logic

should be adjusted to account for the `2f+1` formula, ensuring compatibility across all supported cluster sizes.

Moreover, if the key-sharing process is intended to be performed using external tools such as ssv-keys or any similar utility, the documentation should be promptly updated to reflect these changes. Clear guidance should be provided to operators to prevent misconfigurations. Additionally, it is advisable to review and remove any unnecessary commands from the node implementation to reduce the risk of confusion and to streamline the process for future users.

By addressing these areas, the stability and reliability of the threshold calculation process can be significantly enhanced, contributing to the overall robustness of the network.

F-2024-5656 - Incomplete Validator Exit Logic Due to OwnValidator Oversight - Medium

Description:

The issue originates from the execution of the `ExitValidatorTask`. Specifically, it centers around the implementation of the `ExitValidator` function within the `controller` structure:

operator/validator/task_executor.go:118

```
exitDesc := duties.ExitDescriptor{
PubKey: pubKey,
ValidatorIndex: validatorIndex,
BlockNumber: blockNumber,
}

go func() {
select {
case c.validatorExitCh <- exitDesc:
logger.Debug("added voluntary exit task to pipeline")
case <-time.After(2 * c.beacon.GetBeaconNetwork().SlotDurationSec()
):
logger.Error("failed to schedule ExitValidator duty!")
}
}()
```

In this implementation, the `ExitDescriptor` field `OwnValidator` is not explicitly assigned a value, resulting in it defaulting to `false`. Consequently, when the `ExitDescriptor` is transmitted through the `validatorExitCh` channel and subsequently received within `HandleDuties`, it fails to be incorporated into the `dutyQueue`:

operator/duties/voluntary_exit.go:63

```
case exitDescriptor, ok := <-h.validatorExitCh:
/* duty is created and added to the h.duties */
if !exitDescriptor.OwnValidator {
continue
}

h.dutyQueue = append(h.dutyQueue, duty)
```

This oversight results in a substantial portion of the validator exit logic being bypassed, specifically when the operator holds a share in the exiting validator. In situations where the operator does not hold such a share, the `OwnValidator` field is correctly set to `false`, which is the expected behavior.

The root of the issue is that while the `handleValidatorExited` function correctly generates an `ExitDescriptor` with the `OwnValidator` field properly set, this information is disregarded during the creation of a new `ExitValidatorTask`. Consequently, the flawed creation of the `ExitDescriptor` within the `ExitValidator` function leads to an incomplete and incorrect execution of the validator exit process.

Assets:

- Event Handler

Status: Fixed

Classification

Impact: 2/5

Likelihood: 4/5

Severity: Medium

Recommendations

Remediation: It is imperative to integrate the `OwnValidator` information into the `ExitValidatorTask` structure, enabling this value to be accurately passed from the `handleValidatorExited` function. This enhancement will allow the `ExitValidator` function within the `controller` to ensure that the exit task is correctly created with the appropriate `OwnValidator` value.

This refinement is vital for preserving the integrity of the validator exit process, particularly in scenarios where the operator holds a share in the exiting validator. By ensuring the accurate passing and application of the `OwnValidator` value, the validator exit logic will operate as intended, preventing any bypasses and guaranteeing the proper execution of the exit procedure.

F-2024-5502 - Insufficient Processing of Operator Removal Events - Low

Description:

The SSV node software currently lacks proper implementation for handling the removal of operators. While this functionality is documented on the SSV website and implied by the `OperatorRemoved` event that can be emitted by the smart contract, the actual implementation in the node code does not perform as expected.

The `handleOperatorRemoved` method within the SSV node code is designed to process this event and update the node's state accordingly. However, despite the method being in place, it currently fails to make any meaningful changes to the node's state when the `OperatorRemoved` event is received. This behavior is contrary to the intended functionality.

A notable implication is that an operator who suspects their keys may have been compromised would be unable to exit the network voluntarily. This failure to exit could prevent the operator from mitigating potential risks to the cluster, leaving the network vulnerable to further consequences that could arise from compromised keys.

Assets:

- Event Handler

Status: Mitigated

Classification

Impact: 1/5

Likelihood: 4/5

Severity: Low

Recommendations

Remediation:

To address this issue, it is essential to implement the necessary state changes within the `handleOperatorRemoved` method to ensure that the SSV node accurately reflects the removal of an operator. This will help maintain consistency between the smart contract's state and the node's state, thereby preserving the integrity and reliability of the network.

If implementing this change is not planned, it is strongly recommended to update the documentation to clearly state the current behavior. The documentation should include any limitations or potential workarounds, so users and developers are fully informed about the node's handling (or lack thereof) of operator removal events. This will help prevent misunderstandings and mitigate risks associated with the current implementation.

Resolution: The issue has been mitigated by documenting that the operator removal feature is still in its early stages of development.

The SSV team has expressed their intent to fully implement this functionality; however, due to its complexity, the feature requires further detailed assessment. Completing both the evaluation and implementation within the audit remediation period was considered unfeasible and has been deferred to future updates.

F-2024-5505 - Deprecated Elliptic Curve Cryptography - Low

Description:

The `ECDSAPrivFromInterface` function in the codebase is responsible for converting a `crypto.PrivKey` into an `ecdsa.PrivateKey`. However, this function currently relies on the `ScalarBaseMult` method to compute the public key coordinates from the private key bytes:

ssv/network/commons/keys.go:17

```
func ECDSAPrivFromInterface(privkey crypto.PrivKey) (*ecdsa.Private
Key, error) {
//
// other operations
//
privKey.X, privKey.Y = gcrypto.S256().ScalarBaseMult(rawKey)
return privKey, nil
}
```

The `ScalarBaseMult` method is deprecated and considered a low-level, unsafe API. It is prone to potential cryptographic vulnerabilities and undefined behavior, which can compromise the security of the application.

Assets:

- Cryptography

Status: Fixed

Classification

Impact: 1/5

Likelihood: 1/5

Severity: Low

Recommendations

Remediation:

To enhance both the security and maintainability of the codebase, it is strongly recommended to refactor the `ECDSAPrivFromInterface` function to eliminate the reliance on the `ScalarBaseMult` method. Instead, use the `crypto/ecdh` package, which provides a more secure and modern approach to elliptic curve cryptography. Specifically, the `PrivateKey.PublicKey` method from this package should be used to obtain the public key.

Adopting this recommended approach will address the security concerns associated with deprecated methods and ensure that the codebase follows current best practices in cryptographic operations.

F-2024-5790 - Build Failure Due to Outdated dnsutils Package - Low

Description:

During the build process, the following error message is encountered:

```
E: Version '1:9.18.24-1' for 'dnsutils' was not found
```

The issue originates from the Dockerfile, specifically in the line where the version of the `dnsutils` package is fixed:

```
RUN apt-get update && \
DEBIAN_FRONTEND=noninteractive apt-get install -yq --no-install-rec
ommends \
dnsutils=1:9.18.24-1 && \
rm -rf /var/lib/apt/lists/*
```

The unavailability of this specific package version causes the build process to fail, as the package manager cannot locate and install the required dependency. This disruption impacts the continuous integration and deployment pipeline, potentially leading to delays in project delivery. Additionally, it prevents users from successfully building the node when using Docker, which can hinder their ability to deploy and operate the software effectively.

Assets:

- Docker

Status: Fixed

Classification

Impact: 1/5

Likelihood: 1/5

Severity: Low

Recommendations

Remediation: It is recommended to remove the version specification for the `dnsutils` package in the Dockerfile. This will allow the package manager to automatically install the latest stable version available, reducing the risk of future build failures caused by outdated or unavailable package versions. This approach ensures greater flexibility and helps maintain a more resilient build process.

F-2024-5833 - Deficient Validation of Operator Public Key Uniqueness - Low

Description:

In the SSV network, each operator is registered with an `ID` and an associated RSA public key. The `ID` is assumed to be an ever-increasing number, and the `PublicKey` is expected to be unique across all operators. However, during the audit, it was identified that the current implementation lacks adequate verification to ensure the uniqueness of the operator's public key during the registration process.

In the `handleOperatorAdded`, the following code attempts to verify the uniqueness of the operator's public key:

eth/eventhandler/handlers.go:57:

```
// throw an error if there is an existing operator with the same pu
blic key and different operator id
operatorData := eh.operatorDataStore.GetOperatorData()
if operatorData.ID != 0 && bytes.Equal(operatorData.PublicKey, even
t.PublicKey) && operatorData.ID != event.OperatorId {
logger.Warn("malformed event: operator registered with the same ope
rator public key",
zap.Uint64("expected_operator_id", operatorData.ID))
return &MalformedEventError{Err: ErrAlreadyRegistered}
}
```

Although the code claims to check for public key uniqueness, it only compares the `ID` and `PublicKey` of the incoming operator with those of the operator running the node. This approach fails to verify the uniqueness of the public key across the entire network, rendering the check ineffective from a security standpoint.

While the `ID` is regulated by a smart contract, the uniqueness of the public key is not enforced by the node. This oversight can lead to various issues, such as unexpected node behavior, duplicate message transmissions, and potential security vulnerabilities associated with duplicate public keys.

Assets:

- Operator

Status: Fixed

Classification

Impact: 1/5

Likelihood: 4/5

Severity: Low

Recommendations

Remediation: To address this issue, it is recommended to implement a more robust verification mechanism within the node to ensure the uniqueness of the operator's public key across the entire network.

Implementing this change will not only enhance the overall security posture of the network but also ensure that the integrity and reliability of the operator registration process are maintained.

Observation Details

F-2024-5421 - Potential Vulnerabilities Identified Through Static Code Analysis - Info

Description:

During the audit of the SSV codebase, the use of `golangci-lint` and `gosec` static analysis tools uncovered several warnings that warrant attention.

Golangci-lint Findings:

Integer Overflow Conversion Issues (G115):

Multiple instances of potentially unsafe type conversions from `uint64` to smaller types such as `int`, `int64`, `int32`, `uint8`, and `uint16` were identified. Some of the notable instances include:

- *message/validation/genesis/consensus_validation.go:439,442*
- *utils/boot_node/node.go:217,218*
- *networkconfig/config.go:85*
- *network/topics/scoring.go:201,224*
- *protocol/v2/ssv/runner/committee.go:232,294,301,358*
- *network/discovery/enode.go:27,28*
- *beacon/goclient/goclient.go:268,269*
- *protocol/v2/blockchain/beacon/network.go:82,98*

Additionally, comments like `nolint:all`, `nolint`, and `nolint:unused` were found in the codebase. Upon removing these comments, the following warnings emerged:

***Unchecked Error and Improper Type Assertions**:*

- *api/handling.go:24,26*: The return value of the `render.Render` function is not being checked at two locations.
- *api/handling.go:22*: A type switch on `err` fails to handle wrapped errors correctly.

***Unused Functions**:*

- `getRegistryStores` in *migrations/migrations.go*
- `peersWithProtocolsFilter` in *network/p2p/p2p_sync.go*

***Deprecated Function Usage**:*

- *network/p2p/p2p_setup.go:60*: The use of `rand.Seed` to generate a sequence of random numbers can break expectations about the specific sequence of random results.

Gosec Findings:

The Gosec analysis flagged potential security vulnerabilities and compilation errors related to invalid package imports and undefined methods.

Invalid Package Imports:
Several files contain invalid package import statements, causing compilation errors. These include:

- */e2e/beacon_proxy/beacon_proxy.go:24*
- */e2e/beacon_proxy/intercept/slashinginterceptor/slashing.go:15*
- */e2e/cmd/ssv-e2e/beacon_proxy.go:23,24*

Undefined Functions and Methods:
Errors were found due to undefined functions and incorrect method signatures, such as:

- */e2e/cmd/ssv-e2e/share_update.go:79,105,126,136,142*

Gosec Security Warnings:

File Inclusion via Variable:

- */e2e/cmd/ssv-e2e/share_update.go:167*: Potential file inclusion via variable (`os.ReadFile(filePath)`).

Unhandled Errors:

- */scripts/differ/parser.go:88*: Unhandled errors (`printer.Fprint(&buf, p.fset, n)`).

Assets:

- Code quality

Status: Fixed

Recommendations

Remediation:

Address Warnings: Rectify the identified issues, particularly focusing on fixing the unsafe type conversions, unchecked errors, improper type assertions, and deprecated function usage. This will enhance the codebase's safety and reliability.

Review the Use of nolint and nosec: Reevaluate the necessity of the `nolint` and `nosec` comments. If these directives are required, ensure that they are accompanied by clear explanations justifying their use to maintain transparency and understanding for future maintainers.

Enhance Code Quality: Regularly run static analysis tools to proactively identify and resolve potential vulnerabilities, ensuring that the codebase remains secure and maintainable.

F-2024-5503 - Elimination of Redundant slashableAttestationCheck Function - Info

Description:

Upon thorough examination, it has been determined that the `slashableAttestationCheck` function, initially intended for performing slashing protection checks on attestations, is redundant and should be removed from the codebase. This function, though invoked during the attestation submission process, remains unimplemented:

ssv/beacon/goclient/attest_protect.go:7

```
// slashableAttestationCheck checks if an attestation is slashable
by comparing it with the attesting
// history for the given public key in our DB. If it is not, we the
n update the history
// with new values and save it to the database.
func (gc *GoClient) slashableAttestationCheck(ctx context.Context,
signingRoot [32]byte) error {
// TODO: Implement
return nil
}
```

Further analysis has shown that the function's intended purpose is already adequately covered by the existing slashing protection mechanism. Slashing protection is effectively managed during the signing of attestations, and adding another check at the consensus stage would introduce unnecessary complexity without providing additional security benefits. The current system already ensures that slashable attestations are not submitted, making the `slashableAttestationCheck` function redundant.

Assets:

- GoClient

Status: Fixed

Recommendations

Remediation:

It is advisable to remove the `slashableAttestationCheck` function from the codebase to streamline the process and eliminate potential confusion. Removing this unused and unnecessary function will simplify the code, making it more maintainable and reducing the risk of errors or misunderstandings.

By eliminating this redundant code, the overall clarity and efficiency of the codebase will be enhanced, ensuring that focus remains on the critical components of the slashing protection mechanism.

F-2024-5541 - Utilization of Panics Instead of Error Handling - Info

Description:

During the audit of the SSV Node codebase, multiple instances of the `panic` function were identified. While the `panic` function may be appropriate for handling unrecoverable errors where continued execution would be unsafe, its indiscriminate use can lead to abrupt application termination. This not only deteriorates user experience but also complicates code maintenance. Moreover, unexpected shutdowns or halts of the node could expose the system to additional attacks, potentially compromising the decentralization and overall integrity of the blockchain.

Examples of such instances in the codebase include:

- `panic("unknown role")`
- `panic("unreachable: message type assertion should have been done")`
- `panic("unexpected signed message type") // should be checked before`
- `panic("unexpected partial signature message type") // should be checked before`

Many of these panics are located within message validation logic, which should be meticulously handled to avoid triggering panics from maliciously crafted messages. Although it has not been demonstrated that these specific panics can be triggered, their presence introduces a potential risk to the security posture of the project, particularly in light of future code changes.

Assets:

- Code quality

Status: Fixed

Recommendations

Remediation: To effectively mitigate the identified risks, it is advised to:

Replace Panic Statements: Systematically replace `panic` statements with structured error-handling mechanisms. Return errors to the calling functions for appropriate management, ensuring graceful recovery or orderly shutdown.

Refactor Message Validation: Review and refactor message validation logic to ensure it is resilient against malicious inputs, preventing potential security vulnerabilities.

These steps will strengthen the stability, security, and reliability of the SSV Node within the blockchain ecosystem.

F-2024-5546 - Presence of Commented-Out Files in Runner Package - Info

Description:

During the audit of the codebase, it was observed that two files within the `runner` package contain code that has been entirely commented out:

- *ssv/protocol/v2/ssv/runner/attester.go*
- *ssv/protocol/v2/ssv/runner/sync_committee.go*

This commented-out code appears to be residual from a previous implementation of the `Runner` before the introduction of the Committee consensus and is no longer relevant to the current codebase.

Although this issue does not present any immediate security threats, the existence of extensive commented-out code can lead to confusion among developers and auditors, thereby reducing the overall readability of the code. Furthermore, such commented-out sections contribute to increased maintenance overhead and unnecessary clutter, which can complicate future development efforts.

Assets:

- Code quality

Status: Fixed

Recommendations

Remediation:

A thorough review of the commented-out code in the specified files is recommended. Given that this code is no longer pertinent to the current implementation, it should be removed to enhance code clarity, streamline maintainability, and reduce potential confusion.

F-2024-5636 - Unresolved Operator and Validator Setup Logic - Info

Description:

During the codebase audit, it was noted that the `startCommittee` and `StartValidator` methods within the `validator` package contain code that is entirely commented out, with the only active code being a return statement of `nil` as an error.

Although this issue does not currently pose any immediate security risks, the presence of these commented-out methods—referenced elsewhere in the code—can lead to developer confusion, increase the risk of errors, and contribute to unnecessary clutter within the codebase. Such remnants of obsolete code can negatively impact both the maintainability and readability of the code.

Assets:

- Operator

Status: Fixed

Recommendations

Remediation:

It is recommended to remove all references to the `startCommittee` and `StartValidator` methods throughout the codebase to eliminate potential confusion or misuse. If these methods are deemed unnecessary, they should be entirely removed from the codebase to enhance clarity and improve overall code quality.

F-2024-5697 - Outdated DKG Code Remnants - Info

Description:

During the audit, it was observed that remnants of the Distributed Key Generation (DKG) logic, initially intended for implementation within the node 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:

- *message/validation/errors.go:95*
- *message/validation/signed_ssv_message.go:116*
- *message/validation/genesis/validation.go:488*

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
Commit	20dba00cb02f98e52124158ad36c0bb28839f8a4

The remediation check has been conducted based on commit hash 8297d92, 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 consists of the full codebase of the SSV Node repository, including:

- Cryptography and EKM
- QBFT consensus
- Networking
- Storage