Quantstamp

Obol

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Distributed Validator Technology
Timeline	2023-11-20 through 2023-12-19
Language	Go
Methods	Architecture Review, Computer-Aided Verification, Manual Review
Specification	Charon Client
Source Code	• ObolNetwork/charon #67725e4 • ObolNetwork/charon #3e12d09
Auditors	• Jeffrey Kam Auditing Engineer • Michael Boyle Auditing Engineer • Pavel Shabarkin Senior Auditing Engineer

Documentation quality	Medium
Test quality	Undetermined
Total Findings	18 Fixed: 7 Acknowledged: 8 Mitigated: 3
High severity findings	3 Fixed: 3
Medium severity findings	3 Fixed: 2 Mitigated: 1
Low severity findings	7 Fixed: 1 Acknowledged: 4 Mitigated: 2
Undetermined severity findings	0
Informational findings	5 Fixed: 1 Acknowledged: 4

Summary of Findings

The Charon client is Obol's implementation of DVT (Distributed Validator Technology) for the Ethereum blockchain. Charon is a Golang-based HTTP middleware that enables any existing Ethereum validator clients to operate together as part of a distributed validator. Multiple Charon clients are configured to communicate together to come to a consensus on validator duties and behave as a single unified proof-of-stake validator together.

The main objective of this assessment was to evaluate risks related to slashing, downtime, and security of the private key management. The assessment was time-constrained and was not aimed to be a full security review therefore, it does not provide any security guarantees.

Our assessment revealed three high- and three medium-severity issues. The monitoring service could expose sensitive information such as memory addresses, through stack traces and memory allocations in the heap. In addition this service could lead to a Denial of Service attack. In addition, during in the wild exploration, we have confirmed that public relayers had this service exposed.

The review of the codebase also showed that debugging is not finished yet, and implementation highly depends on Golang's panics. The Obol team fixed reachable panics identified by Quantstamp team in OBO-2. Some of the panics are not handled to recover the crashed execution flow, although these situations are unlikely to occur under normal operating conditions. Nonetheless, there exists a risk of potential Denial of Service (DoS) scenarios under abnormal circumstances.

The development team has been professional and helpful throughout the assessment. The project is still in an ongoing development phase; changes to the codebase were done during the security assessment, at the initial review the codebase had more than 50 "TODO" code comments. The Obol team addressed and removed old "TODO" items during the fix review, and planning to work on all the remaining ones.

Generally, the Charon client extends the attack surface on the validator server. This solution introduces additional entry points for external attackers and allows third parties to connect to the server, processing their messages as a standard part of operations. Although, the P2P port requires to have the identity key of whitelisted peer.

Since the protocol is still in active development, running this software without appropriate network restrictions and secure host and setup configurations pose validator to security risks. The security of this protocol warrants careful attention, as it stands as an underlying technology for operator validation duties.

The public Obol API (Launchpad) and usage of public relayers pose DV clusters with security risks and could be a single point of failure. Operators should deploy and interact through their private relayers and also use private launchpads to mitigate risks of single points of failure.

As for improving the project itself, we highly recommend extensively testing the code, considering more edge cases, more in-the-wild testing in the testnet, and having a full security audit of the protocol before going to production. Also, continuous on-chain observation and revision of the

code based on real-time observation are recommended.

During the fix review phase Obol team addressed and fixed all the high- and medium-severity findings. After the fix review Quantstamp team evaluated areas of the code only related to the fixes of the findings.

ID	DESCRIPTION	SEVERITY	STATUS
OBO-1	DoS Through Publicly Exposed `pprof` Utility	High	Fixed
OBO-2	Intentionally Induced Panic Could Lead to DoS of Charon Client	High	Fixed
OBO-3	Malicious Node Can Force Any Remote Peer Sign P2P Message	High	Fixed
OBO-4	DoS of the DKG Ceremony, Sync Step Is Not Constrained	Medium	Fixed
OBO-5	Keymanager Post Keys in Plain-Text	Medium	Mitigated
OBO-6	Deployable Deadcode of the Leadercast Consensus	Medium	Fixed
OBO-7	Charon Nodes Discovery Through Public Relayers	Low	Acknowledged
OBO-8	Connection in Plain-Text Is Possible	Low	Mitigated
OBO-9	Allowlist and Denylist Are Not Used by the P2P Package	Low	Mitigated
OBO-10	Vulnerable Dependencies in Use	Low	Fixed
OBO-11	Leader Selection in QBFT Consensus Is Highly Predictable	Low	Acknowledged
OBO-12	Possible Downtime Under Normal Operation Even with a Supermajority of Honest Nodes	Low	Acknowledged
OBO-13	Unhandled Panic Statements	Low	Acknowledged
OBO-14	Missing Security Check in The `validateMsg()`	Informational	Acknowledged
OBO-15	Not Production-Ready Codebase	Informational	Acknowledged
OBO-16	Consensus Threshold Can Be Inconsistent with Signature Threshold	Informational	Acknowledged
OBO-17	Signer Component Is Deprecated in the Codebase	Informational	Acknowledged
OBO-18	Business Logic Inconsistency Within the Loop	Informational	Fixed

Assessment Breakdown

This assessment was conducted by the Quantstamp team primarily to estimate risks associated with insuring distributed validator technology. The assessment was time-constrained and does not offer any security guarantees. Quantstamp's objective was to evaluate the repository for issues related to slashing, downtime, and security of private key management.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Integer overflow / underflow
- Number rounding errors

- Denial of service / logical oversights
- Authentication / authorization
- Client / server synchronization
- Potential slashing occurrences
- Private key management and leakage
- Consensus splits
- Invalid incoming messages
- Falsified messages
- Replay attacks
- Business logic contradicting the specification
- Code clones, functionality duplication
- Data integrity loss
- Injection type attacks
- Remote code execution
- Vulnerable software in use

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the Charon client code.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
    2. Best practices review, which is a review of the Golang code to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
    3. Specific, itemized, and actionable recommendations to help you take steps to secure your code.

Scope

This security assessment only focuses on the Charon client. It assumes that everything other than the files in the specified folder, including the used libraries, dependencies, cryptography algorithms, etc, is working correctly and as intended.

Findings

OBO-1 DoS Through Publicly Exposed `pprof` Utility

• High Fixed

Update
Obol team removed the default deployment of `pprof` endpoint for monitoring API and relayer.

File(s) affected: `app/monitoringapi.go`

Description: The Charon client hosts 3 network services, where `:3620` port exposes sensitive information to the public access :

- `:3600` - Validator REST API
- `:3610` - Charon P2P
- `:3620` - Monitoring service

The web API service is hosted on port `:3620` that serves prometheus metrics on `/metrics`, a readiness endpoint on `/readyz`, a liveness endpoint on `/livez`, and a pprof server on `/debug/pprof`. The `/debug/pprof` API server could give access to the sensitive information of the charon client through different types of profiles:

- **allocs:** A sampling of all past memory allocations
- **block:** Stack traces that led to blocking on synchronization primitives
- **cmdline:** The command line invocation of the current program
- **goroutine:** Stack traces of all current goroutines. Use debug=2 as a query parameter to export in the same format as an unrecovered panic.
- **heap:** A sampling of memory allocations of live objects. You can specify the gc GET parameter to run GC before taking the heap sample.
- **mutex:** Stack traces of holders of contended mutexes
- **profile:** CPU profile. You can specify the duration in the seconds GET parameter. After you get the profile file, use the go tool pprof command to investigate the profile.
- **threadcreate:** Stack traces that led to the creation of new OS threads
- **trace:** A trace of execution of the current program. You can specify the duration in the seconds GET parameter. After you get the trace file, use the go tool trace command to investigate the trace.

Through this service, there is a potential risk of an attacker retrieving memory addresses of the charon client and relayer. Moreover, an exposed `/debug/pprof` endpoint in a Go application can indeed be a vector for a Denial of Service (DoS) attack, although it's not the most straightforward or common method for such an attack. The primary risk associated with an exposed `pprof` endpoint is the unintended disclosure of sensitive information and details about the application's internal workings. However, it can also impact the application's

performance, because of the potential for a DoS attack. The additional load and resource consumption caused by accessing these profiles can lead to a reduced performance of the application, potentially leading to a DoS condition under certain circumstances.

The documentation advises that this port should never be made public; however, the current build, along with the default settings, does not safeguard against its exposure to the public internet. The `https://0.relay.obol.tech/` service helps identifying public relayers. Such DoS attacks against identified public relayers could prevent other DVT clients identifying other peers in the cluster. An example of an active instance can be found at http://35.204.189.73:3620/debug/pprof/.

Reference to the similar issue: https://github.com/kubernetes/kubernetes/issues/81023

Recommendation: Remove the `/debug/pprof` from production build or make this service by default non-deployable.

OBO-2
Intentionally Induced Panic Could Lead to DoS of Charon Client

High Fixed

Update

1. **core/tracing.go:safeInt64():**

   Obol team implemented the string parsing of integer value with the help of `strconv.FormatUint()` function. The `safeInt64()` function was removed.
2. **core/proto.go:ParSignedDataFromProto():**

   Obol team implemented the `defer { recover() }` function in the `ParSignedDataFromProto()` function that should catch any of the panics.
3. **eth2util/rlp/rlp.go:decodeLength():**

   Obol team removed `panic()` function from the `decodeLength()` function

File(s) affected: `core/tracing.go:safeInt64()`, `core/proto.go:ParSignedDataFromProto()`, `eth2util/rlp/rlp.go:decodeLength()`

Description: The analysis of `panic()` usage within the codebase reveals that the Charon client is significantly reliant on panics, and not all of these panics are recoverable. This susceptibility could result in the Charon client program crashing, subsequently leading to downtime. If the Charon client become widely adopted within the Ethereum staking ecosystem, a large-scale intentional DoS attack could trigger an 'inactivity leak' mode. In such a scenario, each instance of node downtime would incur severe penalties. We have identified several instances, though not exhaustively, where such a scenario could potentially be initiated by an external party:

1. **core/tracing.go:safeInt64():** The `'/charon/parsigex/2.0.0'` P2P handler is employed to receive and broadcast partial signatures to all connected peers. However, this functionality exposes a vulnerability where any remote peer, even those not whitelisted in the cluster, can cause a Charon client to crash due to unrecoverable panics. Specifically, when the handler initiates the tracing procedure for a duty, the duty's slot value is processed by the `safeInt64()` function. This function will panic if the slot value exceeds the maximum limit of the `int64` type. A malicious remote node could exploit this by intentionally sending a `duty.Slot` value higher than the positive range of the `int64` type, triggering a panic as it fails the validation check. `uint64` range: [0 - 18446744073709551615] when `int64` range [-9223372036854775808 - 9223372036854775807].
2. **core/proto.go:ParSignedDataFromProto():** According to the specifications, the `ParSignedDataFromProto()` function may panic due to unexpected data encountered during JSON unmarshalling. This function is used within the `'/charon/parsigex/2.0.0'` P2P handler and is invoked prior to validating the caller's authentication. Consequently, a malicious node could craft a message that disrupts the JSON unmarshalling process, causing the program to panic and crash.
3. **eth2util/rlp/rlp.go:decodeLength():** The Charon client conducts host discovery by parsing peers' ENRs obtained from the trusted relayer. During this process, if an ENR string, decoded using the `eth2util/enr/enr.go:Parse()` function, exceeds 255 (0xf7 + 8) bytes, the Charon client will panic. Consequently, if the relayer hosts ENR strings larger than 255 bytes, all Charon clients attempting to parse these strings will crash.

Recommendation: Implement recover mechanisms to handle errors and prevent intentional Charon client crashes.

OBO-3
Malicious Node Can Force Any Remote Peer Sign P2P Message

High Fixed

Update

Obol team changed the logic of signing. Instead of passing the hash of the message to sign, they are passing a message and hash it before signing. This prevents the attack vector of passing any hash for signing, even outside of Charon client usage.

File(s) affected: `dkg/bcast/server.go`

Description: During the execution of the DKG ceremony by peers in the cluster, the DKG package deploys several P2P handlers to manage incoming messages from other peers. The exact function of the '/charon/dkg/bcast/1.0.0/sig' handler (handleSigRequest()) remains unclear to us. However, it appears that any external node, even those not on the allowlist, can submit a unique message `Id` and `Hash`, and the receiving peer will sign submitted hash with their private identity key and response the signed signature back to the sender.

An attacker can exploit this functionality and use in several places of Charon client:

- An attacker could strategically precompute hashes of malicious messages intended for a future QBFT consensus rounds, then coerce a peer into signing them via the '/charon/dkg/bcast/1.0.0/sig' handler within the time of DKG process. When the time for the QBFT consensus phase arrives, the attacker could replay these messages, seemingly originating from other peers. This tactic could potentially lead to a falsified consensus.
- Additionally, during the Frost DKG process, an attacker could impersonate other peers by sending cast messages on behalf of other peers to modify their commitments. This scenario would enable an attacker to gain control over a number of shares exceeding the required threshold, ultimately allowing them to reconstruct the private signing key.

The main idea behind the impact for this finding that an attacker can force any peer during the DKG process to sign arbitrary hash and retrieve its valid signature. This breaks every authentication mechanism where it is relied on private identity key (multiple handlers of the Charon client). Moreover, if the private identity key is associated with funds external to the Charon client, there is a risk that these funds could be vulnerable to unauthorized access and potential depletion.

Recommendation: Remove the '/charon/dkg/bcast/1.0.0/sig' handler. Or accept message instead of hash and validate its content before signing.

OBO-4 DoS of the DKG Ceremony, Sync Step Is Not Constrained

Medium · Fixed

Update

The Obol team changed `setStep()` to `updateStep()` function and applied checks which verify that step is increasing monotonically one by one.

File(s) affected: `dkg/sync/server.go`

Description: During the DKG ceremony every remote peer should set a step to synchronize between each other. In the current implementation of `setStep()` function, remote peer can provide any value and it will be accepted by the receiving peer. No validation is implemented. Inside the `startSyncProtocol()` function, there is actual implementation of this synchronization mechanism. When all the remote peers are connected to the peer with ongoing execution, then `startSyncProtocol()` function initializes the inline `stepSyncFunc()` function to step synchronically with all the peers.

The server.AwaitAllAtStep() function makes sure that remotely connected peers are at the same step. If any remote peer is two steps ahead, then isAllAtStep() function returns error and breaks the DKG Ceremony. When remote node sets step its values is stored in the `s.steps` mapping.

In the current implementation of `handleStream()` function, only whitelisted node with valid libp2p setup can send message to the peer and their step will be counted. An attacker as a whitelisted peer could connect to any peer from the cluster and arbitrary big step, and when other peers will go thought synchronization process, it will never be finished. This could intentionally or unintentionally block the entire DKG ceremony.

Recommendation: Implement validation of the `step` parameter in the `setStep()` function or revise the function which increments `step` every time when specific peer sends the message.

OBO-5 Keymanager Post Keys in Plain-Text

Medium · Mitigated

Update

Obol team implemented the log warning if the keymanager is connected via HTTP rather than HTTPS protocol.

Update

The client provided the following explanation:

We cannot simply enforce HTTPS requirements for any keymanager address, because we have to support use cases like docker networks. However, we made an improvement and logged a warning in case we detected the missing https scheme. We're also planning to move to url.ParseRequestURI through the codebase in order to validate URLs better: https://github.com/ObolNetwork/charon/issues/2796 PR: https://github.com/ObolNetwork/charon/pull/2795

File(s) affected: `eth2util/keymanager/keymanager.go`

Description: The `postKeys()` function fails to verify the protocol used for uploading encrypted keys and passwords. To mitigate the risk of local eavesdropping and the potential leakage of keys and passwords in the KeyManager's web server logs, uploads should be conducted via HTTPS (over SSL).

Recommendation: Host the KeyManager on a server configured with SSL, and ensure within the Charon client that connections are established using SSL.

OBO-6 Deployable Deadcode of the Leadercast Consensus

Medium · Fixed

Update

The Obol team removed from Leadercast component from the code.

File(s) affected: `core/leadercast/transport.go`

Description: The Charon client codebase consisted both deployable QBFT and Leadercast consensus mechanisms at the same time. Any dead code especially reachable for the external entity could be used in a malicious way and lead to unexpected results. The Obol team has addressed this issue by eliminating this consensus mechanism. As part of their development roadmap, they have transitioned to the QBFT consensus

Recommendation: Remove any code related to the Leadercast consensus component.

OBO-7 Charon Nodes Discovery Through Public Relayers

Low | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Acknowledged. No actions taken. The documentation states the associated risks. The users concerned about their privacy can refrain from using public relay and deploy a self-hosted relay and conduct discovery exclusively through this channel.

File(s) affected: `https://0.relay.obol.tech/`

Description: Public Relayers operated by the Obol team might reveal the IP addresses of clients who do not use private (self-hosted) relayers. If this vulnerability is exploited, an attacker could potentially identify a significant portion of the Charon clients and execute targeted attacks against them. This observation is intended to underscore the risk associated with being identified due to the use of public relayers. Obol team is aware about this and other risks and define them in their documentation.

Recommendation: Refrain from using public relayers. Instead, deploy a private relayer and conduct peer-to-peer discovery exclusively through this channel.

OBO-8 Connection in Plain-Text Is Possible

Low | Mitigated

Update

The client provided the following explanation:

We added warnings to the associated code paths.

However, considering that we need to support docker network cases where using http is normal practice. Therefore we cannot require only https schemes for the urls. PR: https://github.com/ObolNetwork/charon/pull/2813

File(s) affected: `dkg/disk.go`, `p2p/bootnode.go`

Description: When the Charon client loads the cluster definition using the `loadDefinition()` function, it validates the connecting URL through `validURI()`, which permits connections via the plaintext HTTP protocol. In the `NewRelays()` function Charon client permits plain-text connections to the relayer: `if strings.HasPrefix(relayAddr, "http") {}`

Recommendation: Permit connections exclusively through the secure HTTPS protocol

OBO-9 Allowlist and Denylist Are Not Used by the P2P Package

Low | Mitigated

Update

The client provided the following explanation:

Confirmed that the following values are not used anywhere in the code. Having these flags without using them is confusing for the user and can set false expectations. The Protocol team has removed these flags, and will consider re-introducing them in future if needed. PR: https://github.com/ObolNetwork/charon/pull/2801

Update

Obol team removed IP Allowlist and Denylist from the implementation.

File(s) affected: `p2p/config.go`, `cmd/run.go`

Description: In the command line, we can specify a list of CIDR subnets that are allowed (or denied) from peer connections. This information is then stored in the `Config` object by `p2p/config.go`. However, this is not used anywhere in the codebase.

```
cmd.Flags().StringVar(&config.Allowlist, "p2p-allowlist", "", "Comma-separated list of CIDR subnets for allowing only certain peer connections. Example: 192.168.0.0/16 would permit connections to peers on your local network only. The default is to accept all connections.")
```

```
cmd.Flags().StringVar(&config.Denylist, "p2p-denylist", "", "Comma-separated list of CIDR subnets for disallowing certain peer connections. Example: 192.168.0.0/16 would disallow connections to peers on your local network. The default is to accept all connections.")
```

Recommendation: Confirm whether this is intended. If not, consider implementing the features or removing them, and update the documentation accordingly.

OBO-10 Vulnerable Dependencies in Use

Low Fixed

Update

Marked as "Fixed" by the client. The client provided the following explanation:

The team acknowledges this point and reviewed the dependencies in use, updating them as needed. We reviewed the dependencies listed by Nancy ourselves, and noticed that the only remaining dependency with an issue is btcec. That specific dependency comes from Kryptology, which we only use for their FROST implementation. The reported CVE affects Bitcoin message handling: Charon will never reach those code paths. The Obol team has been working in conjunction with the Nethermind research team, and we aim to replace Kryptology completely soon, entirely removing the dependency on btcec. Until then we will not replace this vulnerable dependency due to the requirement to refactor Kryptology to handle a change to the btcec library introduced in btcec v2.

Description: By running Nancy dependency checker we identified that codebase used several vulnerable dependencies. Note, due to the vulnerable libraries being an indirect dependency to Charon, it is unclear whether it is directly exploitable in the codebase. There is a list of them:

- github.com/consensys/gnark-crypto@0.12.0; CVE
- pkg:golang/github.com/btcsuite/btcd@v0.22.3; CVE-2022-44797
- pkg:golang/github.com/hashicorp/consul/api@v1.25.1; CVE-2022-29153
- pkg:golang/github.com/microcosm-cc/bluemonday@v1.0.1; CVE-2021-42576
- pkg:golang/github.com/nats-io/nkeys@v0.4.5; CVE-2023-46129

Recommendation: Update the external libraries and dependencies to up-to-date versions which mitigates flagged issues.

OBO-11 Leader Selection in QBFT Consensus Is Highly Predictable

Low Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Indeed the leader selection function is deterministic, the team is aware of this. This is a requirement of the QBFT protocol (originally named IBFT 2.0, extending the original IBFT paper), all participants must be able to deterministically calculate the correct leader at all times using only an accurate clock and knowledge of the initial conditions. . In future, the Obol team may look at other consensus protocols that do not have predictable leader elections, for now, there is no action to take with respect to this finding.

File(s) affected: `core/consensus/component.go`, `core/qbft/qbft.go`

Description: The leader selection function is defined by the following function in `core/consensus/component.go` as follows:

```
func leader(duty core.Duty, round int64, nodes int) int64 {
    return (int64(duty.Slot) + int64(duty.Type) + round) % int64(nodes)
}
```

This approach is highly predictable, primarily because the duty slot deterministically increments by one each time, and certain duty types occur more frequently than others (e.g., attestation vs proposal, as detailed in this document). For instance, if a malicious actor aims to launch a DOS attack on the servers, they have a much higher probability to guess the involved nodes and subsequently target them. Specifically, this predictability greatly increases the likelihood of an attacker successfully crippling the system by focusing on nodes with consecutive indices. In scenarios where not all nodes in the cluster are trusted, introducing more randomness into the leader selection process could significantly help mitigate this potential DOS vulnerability.

Recommendation: Determine if the highly predictable nature of leader selection is acceptable. If not, consider implementing a mechanism that achieves fairness, uniqueness, and unpredictability.

OBO-12
Possible Downtime Under Normal Operation Even with a Supermajority of Honest Nodes

Low | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

The `increasingRoundTimeout` mode defines the timeout as follows:

```
func increasingRoundTimeout(round int64) time.Duration {
       return incRoundStart + (time.Duration(round) * incRoundIncrease)
}
```

For round 1 the duration will be 1 second, while for each next round the timeout will be incremented by 250ms. Partially synchronous, turn based consensus games will take longer the more faulty nodes you add, and the ethereum spec does only allow fixed durations to produce duties, this is a tradeoff you have to make.

File(s) affected: `core/consensus/component.go`, `core/qbft/qbft.go`

Description: The consensus has a timeout mechanism that helps with leader rotation, in the case of a malicious leader stalling the group from reaching consensus. When a timeout happens, it triggers a round change. By default, the consensus client uses the increasingRoundTimeout mode. For the first round, it has a timeout of "1 second", and then for any subsequent round, the timeout increases by "250 milliseconds".

Since each slot is "12 seconds", there is an upper bound of how many malicious (or just offline) nodes the system can tolerate, even when there are more than a quorum number of honest nodes. To be specific, if there are more than 21 nodes in the cluster and there are 7 offline nodes (7 consecutive timeouts will take 12.25 seconds), the validator may miss its duty (i.e. proposal or attestation) simply because of the time it takes to reach consensus. This is made worse by the fact that leader selection is highly predictable because then the attacker only needs to launch an attack on a small specific subset of nodes to cause system downtime.

Recommendation: This is by nature of the consensus protocol. However, having more empirical data on the consensus latency and perhaps a tool to help gather these timing statistics can help potential users better understand the risks of potential downtime and choose its cluster size correspondingly.

OBO-13 Unhandled Panic Statements

Low | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

In general, there are two operational models for a server application: a) Exit the process when panic occurs, properly unwinding the stack and releasing any locks. Then the orchestration software is in charge to restart the application. This works great for microservices or for applications that have limited external connections, e.g. database. b) Recover on any panic and keep the application running, reconnecting to external services as needed. As discussed within the Protocol team, Charon employed the first model. Therefore some panics can be valid if we expect to not proceed with the operation and are willing to exit the process. The first three cases were fixed in OBO-2. We have also verified all other instances. None of them should never happen in normal operation and are added as invariants and are extra safety checks in case of something extremely unexpected occurring, where the assumption is that a complete fatal end to the execution of the program is preferable to continuing.

File(s) affected: `See the Description`

Description: The application's current codebase makes extensive use of `panic` statements for error handling. This approach is not recommended for production-ready code, as it leads to abrupt program termination, poor error context, and reduced application reliability. Below are some of the files that contains `panic` statements.

File(s):

- eth2wrap.go
- types.go
- select.go
- manager.go
- api.go
- eip712sigs.go
- createdkg.go
- signeddata.go
- tracing.go
- memory.go
- qbft.go
- options.go
- server.go

Recommendation: Consider handling all instances of `panic` statements that can occur during the runtime of Obol. This is especially important if the `panic` can be triggerred by a peer or attacker.

OBO-14 Missing Security Check in The `validateMsg()`

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> The consensus code checks each and every message for validity here. While it is true that the `validateMsg()` function does nothing, it must be noted that this is nothing more than a small tech debt, which we'll address soon.

File(s) affected: `consensus/transport.go`

Description: In the QBFT consensus component incoming messages are validated through `validateMsg()` function, however in the current implementation the function logic in empty.

Recommendation: Implement logic for message validation.

OBO-15 Not Production-Ready Codebase

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> We removed unimportant, old TODOs items in this PR, and we're tracking progress towards removing all the remaining ones here – none of the items the team found were considered high-risk security wise.
>
> None of the TODO items that could be found in the audited Charon version are of concern for the stability or production readiness of the codebase.

Description: The Charon client is not yet ready for production, as evidenced by the numerous 'TODO' comments throughout the codebase, some of them in areas related to security. This indicates that the product still requires further development before it can be considered production-ready.

Recommendation: We highly recommend completing all the key features outlined in the developer roadmap, followed by a comprehensive security review of the entire product, before proceeding to a production release.

OBO-16 Consensus Threshold Can Be Inconsistent with Signature Threshold

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> Acknowledged and not changed. On occasion, software users have expressed interest in setting the private key threshold to either be more restrictive than the minimum safe threshold (in cases of low trust among node operators), as well as more lax than the safe threshold (in case of private key share loss by node operators). We do not encourage either, and set the minimal safe default both in the user interface and with the command line interface. We document the safe threshold values under crash fault tolerant and byzantine fault tolerant situations here.

File(s) affected: `core/qbft/qbft.go`, `cmd/createdkg.go`

Description: The consensus protocol requires a quorum of nodes to be honest in order to reach consensus, namely `2n/3` where `n` is the number of nodes. However, it is possible to set the DKG threshold to be lower than this. This is not possible through the normal flow because, to create the cluster on the Obol's frontend, it automatically sets the threshold to be exactly the default threshold (i.e. same as `quorum`). However, it is technically possible to create cluster definitions manually that have a lower threshold. Because consensus must be reached first before duty data are signed (with partial keys) and broadcasted to each other, there is a logical lower bound to the threshold. While there is logging if the threshold is different from the default value, the lower bound is not enforced in the codebase.

```
func runCreateDKG(ctx context.Context, conf createDKGConfig) (err error) {
    ...
    safeThreshold := cluster.Threshold(len(conf.OperatorENRs))
    if conf.Threshold == 0 {
        conf.Threshold = safeThreshold
    } else if conf.Threshold != safeThreshold {
        log.Warn(ctx, "Non standard `--threshold` flag provided, this will affect cluster safety", nil,
```

```
z.Int("threshold", conf.Threshold), z.Int("safe_threshold", safeThreshold))
    }
    ...
}
```

Similarly, in `cmd/createcluster.go`, we have

```
func safeThreshold(ctx context.Context, numNodes, threshold int) int {
    safe := cluster.Threshold(numNodes)
    if threshold == 0 {
        return safe
    }
    if threshold != safe {
        log.Warn(ctx, "Non standard threshold provided, this will affect cluster safety", nil,
            z.Int("num_nodes", numNodes), z.Int("threshold", threshold), z.Int("safe_threshold", safe))
    }

    return threshold
}
```

Recommendation: When the threshold is less than `safe`, it should return an error instead of just logging a warning.

OBO-17 Signer Component Is Deprecated in the Codebase

• Informational Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> The team will go through the documentation and double-check for Signer component mentions.

File(s) affected: `docs/architecture.md`, `core/interface.go`

Description: In `docs/architecture.md`, the architecture overview diagram mentions an internal `Signer` component that is responsible for interacting with the external `Remote Signer` component. However, this internal `Signer` component does not exist in the codebase. In the section `Stitching the core workflow` of the documentation, it also mentions that the `Signer.sign` function is subscribed to the consensus client. However, this is not shown in `core/interface.go`, the file that wires the core workflow together.

Recommendation: Confirm whether the component is now deprecated and update the documentation accordingly.

OBO-18 Business Logic Inconsistency Within the Loop

• Informational Fixed

> **Update**
> Obol team moved the `db.resolveContribQueriesUnsafe()` function outside the loop.

> **Update**
> Marked as "Fixed" by the client. The client provided the following explanation:
>
> Fixed in PR: https://github.com/ObolNetwork/charon/pull/2800

File(s) affected: `core/dutydb/memory.go`

Description: While exploring the functionality of DutyDB, we identified an inconsistency in the code that could potentially lead to unexpected issues, although this was not manually verified by code inspection.

When peers reach consensus, each peer stores the duty value in DutyDB using the `Store()` function. If the duty type is `DutySyncContribution`, the data is prepared and stored in mappings, and this data is then resolved within the same `for ... range` loop. In contrast, for all other duty types, data resolution occurs right after the `for ... range` loop.

In the `resolveContribQueriesUnsafe()` function each prepared data query is then sent to the 'Response' channel. During the `resolveContribQueriesUnsafe()` function's loop execution, it sends to the channel every contribution that has yet to be resolved. Thus, even though the `resolveContribQueriesUnsafe()` function could be executed in the `for ... range` loop for the `core.DutySyncContribution` case, for consistency and more predictable results, it would be better to align this logic with the implementation used for other duty types.

Recommendation: Move `db.resolveContribQueriesUnsafe()` right after `for ... range` loop.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Automated Analysis

N/A

Test Suite Results

Our team deployed and ran automated scanners, including Nancy, CodeQL, Semgrep, and GoSec, on the codebase. While the majority of the results were false positives, we manually reviewed all credible issues and have included them as specific findings in the report.

Changelog

- 2023-12-20 - Initial report
- 2024-02-02 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

Quantstamp