Obol

Charon Threat Model

This document outlines a threat model for Charon, in the context of it being a Distributed Validator middleware for Ethereum validator clients.

Actors

- Node owner (NO)
- Cluster node operators (CNO)
- Rogue node operator (RNO)
- Outside attacker (OA)

General Observations

This document describes some considerations the Charon core team made about the security of a distributed validator in the context of its deployment and interaction with outside actors.

The goal of this document is to provide transparency, but is by no means a comprehensive audit or complete security reference. It's a sharing of the experiences and thoughts we gained during the last few years building distributed validator technologies.

While to the Beacon Chain, a distributed validator is seen in much the same way as a regular validator, and thus retains some of the same security considerations, Charon's threat model is different from a validator client's threat model because of its general design.

While a validator client owns and operates on a set of validator private keys, the design of Charon allows its node operators to rarely (if ever) see the complete set of validator private key shares, relying on modern cryptography to split them into smaller chunks.

An Ethereum distributed validator employs advanced signature primitives so that **no operator** ever handles the full validator key in any standard lifecycle step: the BLS digital signature scheme employed by the Ethereum network allows distributed validators to individually sign a blob of data and then aggregate the resulting signatures in a transparent manner, never requiring any of the participating parties to know the full secret in advance.

If the subset of the available Charon nodes is lower than a given threshold, the cluster is not able to continue with its duties.

Given the collaborative nature of a Distributed Validator cluster, every operator must prioritize the liveliness and well-being of the cluster. Charon, at the moment of writing this document, cannot reward and penalize operators within a cluster independently.

This implies that Charon's threat model can't quite be equated to that of a single validator client, since they work on a different - albeit similar - set of security concepts.

Identity private key

A distributed validator cluster is made up of a number of nodes, often run by a number of independent operators. For each DV cluster there's a set of Ethereum validator private keys on which they want to validate on behalf of.

Alongside those, each node (henceforth 'operator') holds an SECP256K1 identity private key, referred to as an ENR, that identifies their node to the other cluster operators' nodes.

Exfiltration of said private key could lead to possible impersonation from an outside attacker, possibly leading to intra-cluster peering issues, eclipse attack risks, and degraded validator performance.

Charon client communication is handled via BFT consensus, which is able to tolerate a given number of misbehaving nodes up to a certain threshold: once this threshold is reached, the cluster is not able to continue with its lifecycle and loses liveness guarantees (the validator goes offline). If more than two-thirds of nodes in a cluster are malicious, a cluster also loses safety guarantees (enough bad actors could collude to come to consensus on something slashable).

Identity private key theft and the subsequent execution of a rogue cluster node is equivalent in the context of BFT consensus to a misbehaving node, hence the cluster can survive and continue with its duties up to what's specified by the cluster's BFT protocol's parameters.

The likelihood of this happening is low: an OA with enough knowledge of the topology of the operator's network must steal `fault tolerance of the cluster + 1` identity private keys and run Charon nodes to subvert the distributed validator BFT consensus to push the validator offline.

Obol

Ethereum Validator Private Keys Access

A distributed validator cluster executes Ethereum validator duties by acting as a middleman between the beacon chain and a validator client.

To do so, the cluster must have knowledge of the Ethereum validator's private key.

The design and implementation of Charon minimizes the chances of this by splitting the Ethereum validator private keys into key shares, which are then assigned to each node operator.

A distributed key generation (DKG) process is used in order to evenly and safely create the private key shares without any central party having access to the full private key.

The cryptography primitives employed in Charon allow a subset of the node operator's key shares to reconstruct the original private key, hence signing data.

While the facilities to do this are present in the form of CLI commands, as stated before Charon never reconstructs the key in normal operations since BLS digital signature system allows for signature aggregation.

A distributed validator cluster can be started in two ways:

1. An existing Ethereum validator private key is split by the private key holder, and distributed in a trusted manner among the operators.
2. The operators participate in a distributed key generation (DKG) process, to create private key shares that collectively can be used to sign validation duties as an Ethereum distributed validator. The full private key for the cluster never exists in one place during or after the DKG.

In case 1, one of the node operators K has direct access to the Ethereum validator key and is tasked with the generation of other operator's identity keys and key shards.

It is clear that in this case the entirety of the sensitive material set is as secure as K's environment; if K is compromised or malicious, the distributed validator could be slashed.

Case 2 is different, because there's no pre-existing Ethereum validator key in a single operator's hands: it will be generated using the FROST DKG algorithm.

Assuming a successful DKG process, each operator will only ever handle its own key shares instead of the full Ethereum validator private key.

A set of rogue operators composed of enough members to reconstruct the original Ethereum private keys might pose the risk of slashing for a distributed validator by colluding to produce slashable messages together.

We deem this scenario's likelihood as low, as it would mean that node operators decided to willfully slash the stake that they should be being rewarded for staking.

Still, in the context of an outside attack, purposefully slashing a validator would mean stealing multiple operator key shares, which in turn means violating many cluster operator's security almost at the same time. This scenario may occur if there is a 0-day vulnerability in a piece of software they all run or in case of node misconfiguration.

Rogue Node Operator

Nodes are connected by means of either relay nodes, or directly to one another.

Each node operator is at risk of being impeded by other nodes or by the relay operator in the execution of their duties.

Nodes need to expose a set of TCP ports to be able to work, and the mere fact of doing that opens up the opportunity for rogue parties to execute DDoS attacks.

Another attack surface for the cluster exists in rogue nodes purposefully filling the various inter-state databases with meaningless data, or more generally submitting bogus information to the other parties to slow down the processing or, in the case of a sybil attack, bring the cluster to a halt.

The likelihood of this scenario is medium, because there's no active threat hunting part: there's no need for the rogue node operator to penetrate and compromise other nodes to disturb the cluster's lifecycle.

Outside Attackers Interactions with a Cluster

There are two levels of sophistication in an OA:

1. No knowledge of the topology of the cluster: The attacker doesn't know where each cluster node is located and so can't force fault tolerance +1 nodes offline if it can't find them.
2. Knowledge of the topology of the network (or part of it) is possessed: the OA can operate DDoS attacks or try breaking into node's servers - at that point, the "rogue node operator" scenario applies.

The likelihood of this scenario is low: an OA needs extensive capabilities and sufficient incentive to be able to carry out an attack of this size.

An outside attacker could also find and use vulnerabilities in the underlying cryptosystems and cryptography libraries used by Charon and other Ethereum clients. Forging signatures that fool Charon's cryptographic library or other dependencies may be feasible, but forging signatures or otherwise finding a vulnerability in either the SECP256K1+ECDSA or BLS12-381+BLS cryptosystems we deem to be a low likelihood risk.

Malicious Beacon Node

A malicious beacon node (BN) could prevent the distributed validator from operating its validation duties, and could plausibly increase the likelihood of slashing by serving charon illegitimate information.

If the amount of nodes configured with the malicious BN are equal to the byzantine threshold for the Charon BFT consensus protocol, the validation process can potentially halt since the BFT parameter threshold is reached - most of the nodes are byzantine - the system will reach consensus on a set of data that isn't valid.

We deem the likelihood of this scenario to be medium depending on the trust model associated with the BNs deployment (cloud, self-hosted, SaaS product): node operators should always host or at least trust their own beacon nodes.

Malicious Charon Relay

A Charon relay is used as a communication bridge between nodes that aren't directly exposed on the Internet. It also acts as the peer discovery mechanism for a cluster.

Once a peer's IP address has been discovered via the relay, a direct connection can be attempted. Nodes can either communicate by exchanging data through a relay, or by using the relay as a means to establish a direct TCP connection to one another.

A malicious relay owned by a OA could lead to:

- Network topology discovery, facilitating the "outside attackers interactions with a cluster" scenario
- Impede node communication, potentially impacting the BFT consensus protocol liveness (not security) and distributed validator duties

- DKG process disruption leading to frustration and potential abandonment by node operators: could lead to the usage of a standard Ethereum validator setup, which implies weaker security overall

We note that BFT consensus liveness disruption can only happen if the number of nodes using the malicious relay for communication is equal to the byzantine nodes amount defined in the consensus parameters.

This risk can be mitigated by configuring nodes with multiple relay URLs from only trusted entities.

The likelihood of this scenario is medium: Charon nodes are configured with a default set of relay nodes, so if an OA were to compromise those, it would lead to many cluster topologies getting discovered and potentially attacked and disrupted.

Compromised Configuration Files

Charon operates with two runtime files:

- A lock file used to address operator's nodes, define the Ethereum validator public keys and the public key shares associated with it
- A cluster definition file used to define the operator's addresses and identities during the DKG process

The lock file is signed and validated by all the nodes participating in the cluster: assuming good security practices on the node operator side, and no bugs in Charon or its dependencies' implementations, this scenario is unlikely.

If one or more node operators are using less than ideal security practices an OA could rewire the Charon CLI flags to include the --no-verify flags, which disables lock file signature and hash verification (usually intended only for development purposes).

By doing that, the OA can edit the lock file as it sees fit, leading to the "rogue node operator" scenario. An OA or RNO might also manage to social engineer their way into convincing other operators into running their malicious lock file with verification disabled.

The likelihood of this scenario is low: an OA would need to compromise every node operator through social engineering to both use a different set of files, and to run its cluster with --no-verify.

Obol

Conclusions

Distributed validator technology (DVT) helps maintain a high-assurance environment for Ethereum validators by leveraging modern cryptography to ensure no single point of failure is easily found in the system.

As with any computing system, security considerations are to be expected in order to keep the environment safe.

From the point of view of an Ethereum validator entity running their services through a DVT provider helps greatly with availability, minimizing slashing risks and maximizing participation in the network.

On the other hand, one must take into consideration the risks involved with dishonest cluster operators as well as rogue third-party Beacon Nodes or relay providers.

In the end, we believe the benefits of DVT greatly outweigh the potential threats described in this document.