Shardeum Audit Report

Response and Update

Delivery: Apr 16, 2024

SHM-1: Not all `ViolationTypes` are implemented

Type: Critical
Files affected: server/src/tx/penalty/violation.ts

On line 32, `ViolationType.DoubleVote` does not have a penalty amount implemented for it.

Impact: As it currently stands, a malicious validator won't get penalized for double voting in Shardeum.

Recommendation: Implement a penalty that is a proportion of the offending validator's staked SHM.

Accepted but reduced to medium severity because it does not result in direct loss of funds. This is a mainnet blocker, but will be addressed after our Incentivized Testnet

launch.

SHM-2: Penalties are uniform across all `ViolationTypes`

Type: Critical
Files affected: server/src/tx/penalty/violation.ts

On line 25, the switch statement sets the penalty amount to be 20% of the `stakeLock` regardless of what kind of violation type it is. This fails to distinguish between safety violations and liveness violations. Safety violations, which are attributable in Shardeum's Proof of Quorum consensus protocol, can lead to loss of user funds whereas liveness violations are not attributable in any protocol, and lead to a user not being able to send a transaction. As such, treating both kinds of violations equally might lead to network centralization and affect validators' incentives.

Recommendation: Attributable safety violations such as double voting should have a harsher penalty as these are typically intentional behaviors on the part of a malicious validator. Non-attributable liveness violations should also be penalized, albeit, less harshly as it is hard to detect whether a node is intentionally being malicious or some external factor outside of their control is causing them to be down.

Accepted but reduced to informational severity. Our current plan is to have these tunable and potentially be making changes to the penalty amounts through mainnet. The initial values are suitable enough for mainnet launch.

SHM-3: `SyncTooLong ViolationType` might lead to centralization among Shardeum validators

Type: Critical

The `SyncTooLong` violation type is meant to penalize a validator for taking too long to sync to the network. However, there are several possible reasons outside of the validator's behavior for causing long synching times.

Impact: Penalizing a validator for taking a while to sync to the network will affect who

will join the Shardeum network as a validator resulting in geographical centralization over time due to: 1) needing a well-resourced machine and, 2) being geographically close to all the other nodes in the network to maximize connectivity.

Recommendation: Instead of penalizing this behavior on the on-chain layer, consider not labeling the node as active until it has successfully synced to the network.

Rejected. We would like to penalize nodes that take too long to sync because that will naturally increase the performance of the average node on the network. If the node takes too long to sync, then it likely will not be able to perform its validation duties well either. We understand that this could limit the hardware/networks that can participate in Shardeum but this is by design. We aim to keep the hardware and network requirements generally low but this penalty provides a floor.

SHM-4: Malicious nodes can cause a discouragement attack to new validator nodes leading to the loss of SHM

Type: Critical

It is possible for a malicious node to conduct a targeted discouragement attack on new Shardeum validator nodes. It leverages the fact that taking too long to sync is a penalizable offense in Shardeum (see SHM-3 for the explanation).

The attack assumes that the malicious archiver has a modified Shardeum archive client that keeps on sending valid but redundant Shardeum states to the targeted node. The targeted node does not know this is a priori until sync time. The other nodes in the network are keeping track of how long the targeted node is taking to sync. Once the timeout described in the *Sync Timeout* document is met, these nodes then update their list of currently synching nodes and apply the penalty described in the *Staking, Rewards, Penalties* v1 document and as explained in SHM-3. The targeted node loses a large percentage of its deposited SHM and never becomes an active validator in the Shardeum network.
Impact: New validators may risk losing a substantial portion of their stake prior to commencing validation on the Shardeum network, potentially resulting in getting kicked out of the network

Recommendation: See SHM-3.

Rejected. The node sync does not depend on the archiver. A malicious archiver could not interfere with the sync finishing process on a node as the joining node gets all of its information from other active nodes in the network. The information on the archiver is only used to restore the network in the case of a complete shutdown

SHM-5: Archiver Node Eclipse Attack

Type: Critical
Files affected: archiver.ts, apoptosis.ts

An attacker creates a large number of nodes and connects them to an archiver node. Once the `maxNode` limit is met as defined in config/server.ts, the attacker can then intentionally make all of the connecting servers go down to initiate apoptosis in the victim archiver node.

Impact: Since validator nodes in the Shardeum network need the archiver network to have good uptime and be accessible, the attack can have an impact on how validator nodes get access to the historical state and how RPC servers can serve the historical state.

Rejected. Archive nodes do store historical state data but validator nodes do not reach out to archivers in this way. Additionally, new validator nodes only interact with the RPC server and active validator nodes. Archivers subscribe to validator nodes rather than the other way around. Apoptosis cannot be triggered in this manner in the archiver nodes. Please provide a proof of concept if you disagree.

SHM-6: DDoS attack against Archiver Node

Type: Critical

The network design of the archive network simplifies the process for any archiver node to quickly determine the IP addresses of others. Since validators depend on the archive network for onboarding onto the network, this implies that archiver nodes are public

knowledge. This design makes it easier for an attacker to mount an attack against the archive network to intentionally prevent Shardeum validators from joining and syncing to the network and from RPC servers from serving Shardeum state.

Impact: Validators seeking to join the network are effectively censored and RPC servers cannot serve requests potentially affecting third-party applications relying on RPC services.
Recommendation: We recommend that archiver nodes adopt a sentry node architecture to minimize exposure of their IP addresses to the public.

Accepted, but reduced to High severity. We are looking into cloudflare and similar options for masking the IP addresses of archivers. Work to decentralize the archive servers is scheduled for post mainnet and this finding will be less relevant. Infra of shardeum-run servers was beyond the scope of this engagement.

SHM-7: New validators are vulnerable to long-range attacks

Type: Critical

In the Shardeum network, for new nodes to get onboarded, they need to connect to archiver nodes. Archiver nodes contain all of the state of the Shardeum network, which includes transactions and EVM state. However, a new validator being onboarded has no means to verify whether the state being sent to them is on the valid Shardeum chain with the currently available shards. This is because archiver nodes in the Shardeum system appear to be trusted actors, and such a new validator requires a large amount of social information to participate.

Impact: New validators can be sent a valid version of the Shardeum state by malicious archiver nodes and essentially start validating on a non-canonical but valid version of the Shardeum state.

Recommendation: We recommend that Shardeum adopt a weak subjectivity security model in which a well-defined weak subjectivity period is used to set checkpoints. These checkpoints allow new nodes to retrieve the most up-to-date consistent state across validators in the network from archiver nodes.

Rejected. New nodes join by interacting with active nodes, not archivers. Additionally, nodes do have mechanisms to resolve conflicts in data that they receive when syncing from other nodes in the network. The archivers are not a part of this process.

Shardeum uses BFT PoS. We do not have competing chains and have absolute finality. A new validator will not have to decide between competing chains and cannot be tricked into adopting a malicious chain as only one chain exists. The node will know immediately if its data is not in sync with the greater network and will not be able to join the consensus process. See this: https://ieeexplore.ieee.org/abstract/document/8653269

SHM-8: DoS attack against JSON RPC API Service

Type: Critical

As explained in `config.ts`, the JSON RPC API service has a publicly exposed IP address to enable subscriptions. This makes it easy for an attacker to intentionally bring down Shardeum JSON RPC API services and impact third-party services reliant on them.

Impact: Shardeum JSON RPC API services are unable to promptly handle service requests to third-party services.

Recommendation: See SHM-6

Accepted. Similar response to SHM-6. Outside the scope of this engagement and this issue will be mitigated by multiple parties hosting RPC servers after mainnet launch.

SHM-9: Resolve all TODOs in the codebase

Type: Critical
Files affected: All files in the scope

In the codebases within the scope of this audit, numerous TODOs have been identified related to critical functionalities associated with the Shardeum protocol.

Impact: Not all of the core functionality of Shardeum is implemented.

See this table of TODO status:

TODO	Status/MR
No validation of active node list returned from archiver with any consensor nodes	https://gitlab.com/shardus/archive/archive-server/-/merge_requests/179
Code optimization: exclude could be a Set of node Ids that has linear time lookup	https://gitlab.com/shardus/archive/archive-server/-/merge_requests/176
the consensus radius needs to hold one more node	https://gitlab.com/shardus/global/shardus-global-server/-/commit/498fa45770e60596b7341ac1478d8cf8a247844e#22dea9b570ae7803f0b8919babc33f9d4a0a4348
Gossip data to all connected archiver nodes might create redundant network payload.	Confirmed not a concern. SEC-100
State data validation with other nodes is crucial	Not a concern. Code is deprecated. SEC-103
validate it with multiple randomly selected active nodes	In progress. SEC-158
Contract address must be provided in transaction receipt	TODO is not relevant, removed here: https://gitlab.com/shardus/relayer/collector/-/merge_requests/24
Cross-validation with multiple archivers and active nodes for the list of active nodes is required	In progress. SEC-95
TODO comment and logic needs clarification	https://gitlab.com/shardeum/server/-/merge_requests/407
Cycle Creator TODOs	Blocked, waiting consult. SEC-154

SHM-10: `Crypto.verifyObj` does not verify if an object has a sender field

Type: High
Files affected: archive-server/API.ts, archive-server/Crypto.ts

In API.ts, `validateRequestData` validates the request data from an API request. In

particular, it checks that `data` has been properly signed by a valid public key pair on line 1250 of API.ts, where `data` is of type unknown and `{sender: string, sign: Signature}`. `Crypto.verify` then calls `core.verifyObj` where the core is the shardus-crypto-utils library. In the `verifyObj` implementation, the passed object (which is data in our case) simply checks if the object has a valid sign field but does not check if it has a valid sender field.

Impact: A malicious requester can intentionally use two different public key pairs, one in the sender field of `data` and the other in the validation process performed by `core.verifyObj`. This can lead to the malicious requester getting access to information that they should not be able to request.

Recommendation: Add a check to `verifyObj` to check that the sender field is indeed the owner of the public key used to sign `data`.

Rejected. The specific location in code that is described in this issue looks good to us. Maybe the location was incorrect?

SHM-11: Vulnerable NPM Dependencies

Type: Medium
Files affected: All of the codebases in the scope
Throughout the codebases that rely on Typescript, several npm modules are vulnerable as per the NPM registry.

Impact: A vulnerable NPM package may cause unknown effects on the various Shardeum codebases.

Recommendation: Run npm audit and either update or replace vulnerable modules

Accepted. We are in the process of migrating to GitHub. With this change we get access to real-time alerts rather than our current passive process.

SHM-12: Hardcoded return values in JSON RPC Server

Type: Low

Files affected: json-rpc-server/api.ts

In api.ts, `getCurrentBlock()` returns a hardcoded result.

Impact: Any third-party service needing to query a block will get the same block regardless of what the actual current block is.

Recommendation: Update `getCurrentBlock()` to return the most current Shardeum block.

Accepted, informational. This doesn't require a change as dynamic info is provided when this function is used. However it could use a comment explaining what's going on.

SHM-13: `ShardusCoreMaxID`, `ShardeumMinID`, `ShardeumMaxID` are not `ViolationTypes`

Type: Informational
Files affected: server/shardeumTypes.ts

`ShardusCoreMaxID`, `ShardeumMinID`, and `ShardeumMaxID` appear to be node identification indices for Shardus and Shardeum servers. However, these appear under the `ViolationType` enum.

Recommendation: Create a new enum for these indices and remove them from the `ViolationType` enum.

Rejected. This is a dev practice for signposting enums. Not a security issue but maybe a comment is nice

SHM-14: No behavior when `cycle_q1_start` event is emitted in the Join Protocol v2 init implementation

Type: Informational
Files affected: shardus-global-server/src/p2p/index.ts

In the `init` function for initializing the JOIN protocol v2, upon getting an event for the Q1 cycle to begin, there is no corresponding behavior that is done.

Recommendation: It is unclear what the intended behavior should be based on the available documentation available to us during the audit.

Rejected. This is not a security issue and is so low priority that it will probably not get fixed.

SHM-15: Remove commented-out code completely

Type: Informational
Files affected: All files in the scope

Throughout the codebases in the scope, there are a lot of portions of code that are commented out without any corresponding explanation or apparent use.

Impact: Affects the readability of the codebases.

Rejected. We intentionally leave a lot of commented code as a knowledgebase for previous attempts at solving issues and old implementations. There are probably ways for us to organize the commented out code better but unless there is some specific security issue you can identify in a section of commented out code it is probably gonna stay.