Security Audit Report

Syndicate

v1.0

September 9, 2025

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Syndicate Inc. to perform a security audit of the withdrawal offchain components for the Syndicate platform.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, that could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following targets:

Repository	https://github.com/SyndicateProtocol/syndicate-appchains
Label	Paths referencing this target are prefixed below with `appchain:`
Commit	`2ddcabf03944ff6003e690b8953b95811af7013f`
Scope	The scope is restricted to the following directories: ● `synd-withdrawals/*` ● `synd-translator/*` ● `synd-mchain/*` ● `synd-chain-ingestor/*` ● `shared/src/*`
Fixes verified at commit	`340b56f6a111a2479b75af8c85e1736a020e46d4` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/SyndicateProtocol/terraform
Label	Paths referencing this target are prefixed below with `terraform:`
Commit	`4c4484e16ba212a0dada6228e527ea73c2de1536`
Scope	The scope is restricted to all files inside the `gcp/*` directory.
Fixes verified at commit	`1097dd5063c2f32528730b05767dd097e85d5553` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/SyndicateProtocol/helm
Label	Paths referencing this target are prefixed below with `helm:`
Commit	`94ac5b16902991c90e6df81633ffd0f268ed9091`
Scope	All files are in the scope of the audit, except the following: ● `charts/apps/values/testnet/values.yaml` ● `charts/canary/*`

	• `documentation/*`
Fixes verified at commit	`a1681af533dd589930d325dc289cf8e2f6ddeb90` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/SyndicateProtocol/jsonrpsee
Label	Paths referencing this target are prefixed below with `jsonrpsee:`
Commit	`b20e85ccc31295e0d5aed81a418100eb9ee912dc`
Scope	The scope is restricted to the changes between commit `b20e85ccc31295e0d5aed81a418100eb9ee912dc` and `274a8f025daadb6a6b592ade55ea990bda2f440b`.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The scope of the audit features the core protocol functionality and off-chain components implemented for Syndicate Withdrawals, Translator, and core Infrastructure as Code (IaC) artifacts.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Medium-High**	-
Code readability and clarity	**Medium-High**	Detailed inline comments regarding usages and expected functionalities are adequately documented.
Level of documentation	**High**	The client provided detailed documentation in the README files and https://docs.syndicate.io.
Test coverage	**Medium-High**	-

Summary of Findings

No	Description	Severity	Status
1	Silent loss via unvalidated zlib header in event parsing	Critical	Resolved
2	Static fee cap ignores chain units	Major	Resolved
3	Unsafe RPC buffer and message size	Major	Resolved
4	Incorrect log count logic causes `maxQty` validation to be ineffective	Major	Acknowledged
5	Missing bounds checks on block index may cause panic	Major	Acknowledged
6	Unequal block and receipt lengths may cause panic or skipped validation in `validate_receipts`	Major	Acknowledged
7	Private key stored as plain string in `submit_proof_to_chain` CLI handler	Major	Resolved
8	Shared `TransactOpts` instance may cause nonce reuse and transaction failures	Major	Resolved
9	`parse_address` accepts empty input	Minor	Resolved
10	Underlying decode error is silenced during transaction serialization	Minor	Resolved
11	Max quantity parameter is not enforced during log retrieval	Minor	Acknowledged
12	Invalid `enable_private_nodes` field in `cluster_autoscaling` block	Minor	Resolved
13	Incomplete denial of SMTP traffic in firewall rule	Minor	Resolved
14	Redis authentication disabled in testnet environment	Minor	Acknowledged
15	Containers running as root with full capabilities	Minor	Resolved
16	Disabled network policies allow unrestricted inter-namespace traffic	Minor	Acknowledged
17	GitHub PAT propagated cluster-wide via `ClusterExternalSecret`	Minor	Acknowledged

18	Service account tokens automatically mounted without RBAC restrictions	Minor	Resolved
19	Unencrypted `WebSocket` connections used for sequencing	Minor	Acknowledged
20	Incorrect HTTP status code returned	Informational	Resolved
21	Code quality improvement	Informational	Resolved
22	Incorrect inline documentation for `ChainIdMismatched`	Informational	Resolved
23	Hardcoded `log_timeout` field may limit flexibility	Informational	Resolved
24	Error message does not indicate all failure possibilities	Informational	Resolved
25	Missing descriptive error messages in `assert_eq!` macros	Informational	Resolved
26	Use of magic numbers decreases maintainability	Informational	Resolved
27	Timestamp truncated to `u32`	Informational	Acknowledged
28	Overly permissive internal network access rules	Informational	Resolved
29	Multiple unresolved TODOs	Informational	Acknowledged

Detailed Findings

1. Silent loss via unvalidated zlib header in event parsing

Severity: Critical

In appchain:synd-translator/crates/synd-block-builder/src/appchains/shared/rollup_adapter.rs, events are parsed with:

```
.iter()
.filter_map(|log| self.transaction_parser().get_event_transactions(log).ok())
```

This means that any Err from get_event_transactions is silently dropped.

Additionally, in SequencingTransactionParser.decode_event_data:

```
let compression_byte = data[0];
let compressed_data = &data[1..];
let compression_type = get_compression_type(compression_byte);
match compression_type {
    CompressionType::None => ...,
    CompressionType::Zlib => {

        let mut decompressed_data = decompress_transactions(data)
            .map_err(|e|
SequencingParserError::DecompressionError(e.to_string()))?;
        ...
    }
    CompressionType::Unknown => Err(UnknownCompressionType),
}
```

An attacker can send a tiny payload whose first byte passes is_valid_zlib_cm_bits but whose FLG or header checksum is invalid.

get_compression_type returns Zlib, so decompress_transactions(data) is called.

decompress_transactions reads the entire input into memory, decompresses with no size cap, then RLP decodes all transactions.

Therefore, the malicious data can fail decompression (due to an invalid FLG/checksum), resulting in an error.

`decode_event_data` returns `Err(DecompressionError)` → `get_event_transactions` logs the error and returns `Err` → `filter_map(... .ok())` swallows the log → eventually all transactions in that event are lost silently.

Recommendation

We recommend only accepting `CM=8` and validating both `CMF+FLG` checksums.

Status: Resolved

2. Static fee cap ignores chain units

Severity: Major

In `appchain:shared/src/tx_validation.rs:55-68`, the hardcoded fee cap of 1 ETH (1 × 10^{18} wei) applies uniformly to all chains.

On chains with different native tokens or unusual gas models (e.g., Polygon, BNB, etc.), this single constant may be too restrictive, rejecting valid high fee transactions.

Recommendation

We recommend making the fee cap configurable based on chain, e.g,. load `tx_fee_cap` from `Config.rpc_providers[chain_id]` and fallback to a sane default if unset, but allow operators to override per chain.

Status: Resolved

3. Unsafe RPC buffer and message size

Severity: Major

In `appchain:synd-chain-ingestor/src/client.rs:339-340`, inside `IngestorProvider::new`, the `WebSocket` client builder is configured as follows:

```
WsClientBuilder::new()
    .max_response_size(u32::MAX)
    .max_buffer_capacity_per_subscription(1024) // buffer up to 1024 messages
per subscription
    .request_timeout(timeout)
    // ...
    .build(url)
```

The `max_response_size(u32::MAX)` removes any cap on individual message size, allowing a single RPC response to be arbitrarily large and potentially exhaust memory.

Additionally, the `max_buffer_capacity_per_subscription(1024)` fixes the buffer at 1024 messages.

This is problematic because an attacker or misbehaving provider could flood the subscription with rapid or nested events, leading to large memory usage (1024 × average message size).

Recommendation

We recommend the following:

1. Limiting individual message sizes to a reasonable maximum (e.g., 1–5 MB) based on expected block/log payloads:

```
.max_response_size(5 * 1024 * 1024) // 5 MB
```

2. Making buffer capacity configurable (via CLI or config file) so operators can tune it:

```
 .max_buffer_capacity_per_subscription(config.ws_buffer_capacity)
```

Status: Resolved

4. Incorrect log count logic causes `maxQty` validation to be ineffective

Severity: Major

In `appchain:synd-withdrawals/scripts/tee-verifier/main.go:80-82`, the logic enforces that if `maxQty` parameter is specified (e.g., `maxQty > 0`) and if the length of logs is larger than the parameter (`uint64(len(logs)) >= maxQty`), all the logs are returned.

This is incorrect because the `maxQty` parameter is used to enforce the maximum amount of logs to retrieve. The current logic incorrectly returns the logs despite the number of logs having exceeded the `maxQty` limit.

Consequently, this may cause functions that validate the retrieved logs with the `maxQty` parameter to fail:

- `appchain:synd-withdrawals/scripts/tee-verifier/main.go:177-183`
- `appchain:synd-withdrawals/scripts/tee-verifier/main.go:205-211`

Recommendation

We recommend updating the condition such that the `maxQty` limit is enforced during log retrieval.

Status: Acknowledged

The client states that the code is only affecting a test script and does not affect operating code.

5. Missing bounds checks on block index may cause panic

Severity: Major

In `appchain:synd-verifier/src/verifier.rs:220-226`, the index `i` was used directly to access `settlement_chain_input.blocks[i]` after two `while` loops.

Without bounds checks, a small block array will cause an out of bounds panic.

Recommendation

We recommend adding explicit bounds checks in both loops and before using `i`.

Status: Acknowledged

The client states that `synd-verifier` has been deleted and logic has been migrated to `synd-withdrawals/synd-enclave` which is being reviewed in an upcoming audit (Stage 1C).

6. Unequal block and receipt lengths may cause panic or skipped validation in `validate_receipts`

Severity: Major

In `appchain:synd-verifier/src/verifier.rs:73-99`, the code iterates over `input.receipts` and directly indexes `input.blocks[i]` without verifying that `input.blocks.len() == input.receipts.len()`.

If `receipts.len() > blocks.len()`, `input.blocks[i]` will panic, crashing the verifier.

If `receipts.len() < blocks.len()`, some blocks will never be checked, potentially hiding mismatched roots.

Recommendation

We recommend enforcing equal lengths or iterating in lockstep at the top of `validate_receipts`.

Status: Acknowledged

The client states that synd-verifier has been deleted and logic has been migrated to synd-withdrawals/synd-enclave which is being reviewed in an upcoming audit (Stage 1C).

7. Private key stored as plain string in submit_proof_to_chain CLI handler

Severity: Major

In appchain:synd-withdrawals/synd-tee-attestation-zk-proofs/proof-submitter/src/main.rs:126, the CLI handler captures the private key as a plain String:

```
async fn submit_proof_to_chain(
    chain_rpc_url: String,
    contract_address: Address,
    private_key: String,
    proof: GenerateProofResult,
) -> Result<(), ProofSubmitterError> { ... }
```

Revealing secrets in a normal String means they live in memory unencrypted and may appear in backtraces, core dumps, or debug logs if the process panics or is misconfigured.

Future logging or error handling could inadvertently include private_key, exposing it to operators or logs.

Recommendation

We recommend using a zeroizing secret type so the key is wiped from memory after use.

Status: Resolved

8. Shared TransactOpts instance may cause nonce reuse and transaction failures

Severity: Major

In appchain:synd-withdrawals/synd-proposer/pkg/proposer.go:87-102, the methods closeChallengeLoop and pollingLoop share the same *bind.TransactOpts instance named SettlementAuth.

Since TransactOpts carries the transaction nonce internally, leaving opts.Nonce unset causes each call to invoke PendingNonceAt concurrently, allowing two nearly simultaneous calls to fetch the same nonce.

If the first transaction succeeds, the second will be rejected because the nonce is too low.

If `opts.Nonce` is ever written back into `TransactOpts`, every subsequent transaction keeps using that same stale nonce, so only the very first transaction ever goes through.

Recommendation

We recommend avoiding using the same `*bind.TransactOpts` object for every `send`. Instead, right before calling the contract, make a fresh shallow copy of the original `SettlementAuth`, clear its `Nonce` field, and use that copy.

Status: Resolved

9. `parse_address` accepts empty input

Severity: Minor

In `appchain:shared/src/parse.rs:26-28`, the `parse_address` function does not handle empty or whitespace-only strings as implemented in the `parse_addresses` function. This might create unintended results in the tests as the function is only called in these for the current version.

Recommendation

We recommend implementing the same validation as in `parse_addresses`.

Status: Resolved

10. Underlying decode error is silenced during transaction serialization

Severity: Minor

In `appchain:shared/src/tx_validation.rs:19-26`, the `decode_transaction` function replaces all decoding errors from `TxEnvelope::decode(&mut slice)` with a generic `InvalidInput(UnableToRLPDecode)` error.

This is problematic because it masks the actual reason for failure (e.g., unexpected EOFor invalid data format), hindering debugging and potentially obscuring the root cause of malformed transactions.

Recommendation

We recommend capturing the original decoding error and either propagating it or logging it explicitly.

Status: Resolved

11. Max quantity parameter is not enforced during log retrieval

Severity: Minor

In `appchain:synd-withdrawals/scripts/tee-verifier/main.go:58-89`, the `getLogs` function is intended to return no more than `maxQty` logs.

However, when the `FilterLogs` call succeeds (i.e., `err == nil`), the returned log count is not checked against `maxQty` in line `70`. This bypasses the intended limit, potentially returning more logs than requested.

Additionally, this issue also affects lines `83-87`, where the sum of old logs (`prevLogs` variable) and new logs (`logs` variable) may exceed max quantity limit.

Recommendation

We recommend modifying the implementation such that after the `FilterLogs` call, check if `maxQty > 0` (i.e., max quantity limit is specified) and truncate the returned logs accordingly.

Status: Acknowledged

The client states that this is a test directory.

12. Invalid `enable_private_nodes` field in `cluster_autoscaling` block

Severity: Minor

In `terraform:gcp/syndicate-io/syndicate-appchains-mainnet/us-central1/gke/main.tf:66`, the `cluster_autoscaling` block includes `enable_private_nodes`, which is not a recognized field according to the terraform-google-kubernetes-engine module.

Eventually, this field will be ignored by Terraform.

Recommendation

We recommend removing the `enable_private_nodes` field from the `cluster_autoscaling` block.

Status: Resolved

13. Incomplete denial of SMTP traffic in firewall rule

Severity: Minor

In `terraform:gcp/syndicate-io/syndicate-appchains-mainnet/us-central1/vpc/main.tf:65-73`, the `deny-smtp` firewall rule only blocks TCP port 25.

However, SMTP traffic can also occur over ports 465 (SMTPS), 587 (submission), and 2525 (alternative submission). The current configuration may allow SMTP traffic to bypass the intended restriction via alternate ports, reducing the effectiveness of the rule.

Recommendation

We recommend extending the deny rule to include ports 465, 587, and 2525.

Status: Resolved

14. Redis authentication disabled in testnet environment

Severity: Minor

In `terraform:gcp/syndicate-io/syndicate-appchains-testnet/us-central1/memorystore/main.tf`, the Redis instance is configured with `authorization_mode = "AUTH_DISABLED"`. While the configuration includes a comment acknowledging this is intentionally insecure for development convenience, authentication is completely disabled.

This is problematic because Redis can be used for file reading/writing operations beyond its primary caching function, making it a potential pivot point during post-exploitation activities. Any entity with VPC access can read or write data without authentication, potentially exposing sensitive cached information or allowing attackers to manipulate application state.

Recommendation

We recommend enabling authentication even in testnet environments to prevent Redis from becoming an attack vector during security incidents.

Status: Acknowledged

The client states the following:

“Memorystore is only accessible from k8s pods within our VPC (via internal-IPs only + firewall rules), which reduces the attack vector to "entities with VPC access".

The vulnerability is: “Any entity with VPC access has read/write access to Memorystore” The recommended solution is to add auth to Memorystore so that only Maestro/Sequencer pods can write/read from it. But any entity with VPC access has access to Maestro/Sequencer pods, so even if auth was enabled on Memorystore it would have no effect.

Furthermore, the data stored in Memorystore is all public info (wallet+chain+nonce), so reads are not an issue. An attacker could inject fake values to Memorystore, which potentially opens us up to a sandwich/MEV-attack, but again any entity with VPC access would have several ways of doing that.”

15. Containers running as root with full capabilities

Severity: Minor

In `helm:charts/rpc/values.yaml`, the container configuration allows pods to run as root user with full capabilities, as noted in the security comment referencing SEQ-1022. The principle of least privilege is not enforced at the container level.

This is problematic because compromised containers running as root can more easily escape containment, access sensitive host resources, or escalate privileges within the cluster. Root access provides unrestricted file system access and the ability to install packages or modify system configurations.

Recommendation

We recommend configuring containers to run as non-root users with minimal required capabilities.

Status: Resolved

16. Disabled network policies allow unrestricted inter-namespace traffic

Severity: Minor

In `helm:charts/apps/values/testnet/values.yaml`, network policies are explicitly disabled with `networkPolicies.enabled: false`. The configuration acknowledges that traffic is allowed between all namespaces, enabling lateral movement if any pod is compromised.

This is problematic because a breach in any single pod can lead to unrestricted access across all namespaces, potentially exposing sensitive services and data. Without network segmentation, attackers can freely communicate with critical infrastructure components from compromised workloads.

Recommendation

We recommend implementing network policies to restrict traffic between namespaces based on the principle of least privilege.

Status: Acknowledged

The client states the following:

“With our current microservice architecture in k8s, traffic must be allowed across nearly all namespaces. We accept the risk of leaving `networkPolicies` disabled because we do not have any critically sensitive services or data available to the k8s pods, and the development cost of maintaining `networkPolicies` at this stage would outweigh the minor security benefits”

17. GitHub PAT propagated cluster-wide via `ClusterExternalSecret`

Severity: Minor

In `helm:charts/apps/values/testnet/values.yaml`, a GitHub Personal Access Token (PAT) is distributed cluster-wide through a `ClusterExternalSecret` for GHCR image pulls. The configuration acknowledges this propagates the secret to all namespaces where any compromised pod can read it.

This is problematic because the PAT is user-bound and potentially over-privileged, creating a single point of compromise. If any pod in any namespace is breached, the attacker gains access to the GitHub credentials, potentially allowing unauthorized access to private repositories or image registries.

Recommendation

We recommend migrating to short-lived GitHub App installation tokens with minimal required permissions.

Status: Acknowledged

The client states the following:

“With our current microservice architecture in k8s, the Github PAT must be allowed across nearly all namespaces and its existence is temporary. We accept the risk of this secret existing across all namespaces because (1) the private repos it has access to will be public soon, so this secret is temporary (2) the private repos it has access to will not contain any critically sensitive data, (3) almost every namespace needs to use this secret, so using a cluster-wide secret is the most appropriate implementation.

We acknowledge that a ""Github App"" can be used instead of a PAT to prevent auth being tied to a single user / ""over-privileged"". We accept the risk of using a Github PAT temporarily and will switch to a Github App if needed, but will not be necessary once our repos/images are public”

18. Service account tokens automatically mounted without RBAC restrictions

Severity: Minor

In `helm:charts/rpc/values.yaml`, service accounts are configured with `automount: true`, automatically mounting Kubernetes API credentials into pods. No corresponding RBAC policies are mentioned in the configuration.

This is problematic because pods gain unnecessary access to the Kubernetes API even when not required for their operation. Compromised pods can use these tokens to query cluster information, potentially escalating attacks or gathering intelligence about the cluster topology.

Recommendation

We recommend disabling automatic service account mounting for pods that do not require Kubernetes API access and implementing restrictive RBAC policies for those that do.

Status: Resolved

19. Unencrypted `WebSocket` connections used for sequencing

Severity: Minor

In `charts/apps/values/testnet/values.yaml`, the sequencing WebSocket URL is configured as `ws://devnet-ingestor-risa-51014.ingestors.svc.cluster.local:8545` using unencrypted WebSocket protocol instead of secure WebSocket (`wss://`).

This is problematic because communication between services is transmitted in plaintext, allowing potential eavesdropping or man-in-the-middle attacks within the cluster network.

While the impact is limited in a devnet environment, this pattern may be replicated in production deployments.

Recommendation

We recommend using secure WebSocket connections (`wss://`) even in development environments to ensure secure communication patterns are maintained across all deployments.

Status: Acknowledged

The client states that they accept the risk of unencrypted inter-pod traffic, because (1) all pod communication takes place within the same k8s cluster in the same VPC which is not publicly accessible and (2) pod communication does not include any critically sensitive data (e.g. private keys, passwords).

20. Incorrect HTTP status code returned

Severity: Informational

In `appchain:synd-withdrawals/server/main.go:28`, the HTTP handler returns a 200 OK status (`http.StatusOK`) for non-POST requests. This response is incorrect and misleading because the actual request method sent (e.g., `GET`) is not the intended request type (`POST`).

Recommendation

We recommend returning `http.StatusMethodNotAllowed` in the response field.

Status: Resolved

21. Code quality improvement

Severity: Informational

In `appchain:synd-withdrawals/synd-tee-attestation-zk-proofs/aws-nitro/src/attestation_document.rs:111-132`, the following improvements can be implemented:

- The validation in line `111` can be removed because the `doc.timestamp` variable is unsigned, hence it will not be a negative value. Additionally, the zero value check is implemented in line `99`.
- The validation in line `115` that checks `doc.pcrs.is_empty()` can be removed because it is validated before in line `100`.
- The validation in line `120` that checks `if *key > 31` can be removed because it is validated before in line `115` (`doc.pcrs.len() > MAX_PCR_COUNT`).
- The validation in line `130` can be removed because it is validated before in line `102`.

Recommendation

We recommend applying the above-mentioned recommendations.

Status: Resolved

22. Incorrect inline documentation for `ChainIdMismatched`

Severity: Informational

In `appchain:shared/src/json_rpc.rs:204`, the inline documentation for the `ChainIdMismatched` error is “Chain ID is missing”, which is incorrect.

Recommendation

We recommend updating the documentation to be relevant to the error.

Status: Resolved

23. Hardcoded `log_timeout` field may limit flexibility

Severity: Informational

In `appchain:synd-chain-ingestor/src/config.rs:43-54`, the `log_timeout` value passed to `EthClient::new` is hardcoded as `Duration::from_secs(300)`, which is 300 seconds.

While this may be suitable for current use, making it configurable (e.g., via environment variable or config file) would improve flexibility for different deployment environments or future tuning.

Recommendation

We recommend exposing the `log_timeout` field as a configurable value in the `Config` struct.

Status: Resolved

24. Error message does not indicate all failure possibilities

Severity: Informational

In `appchain:synd-mchain/src/db.rs:263-279`, the `add_batch` function performs a few validations, which if an error occurred, the error will indicate which validations goes wrong with the "invalid batch: timestamp {} < {} or seq block {} != {} or set block {} < {}" statement.

However, this error message is not sufficient because it does not include the possibility of `mblock.slot.seq_block_number <= state.slot.seq_block_number` validation failed.

Recommendation

We recommend updating the error message to accurately reflect all checked conditions, including the case where `mblock.slot.seq_block_number <= state.slot.seq_block_number`.

Status: Resolved

25. Missing descriptive error messages in `assert_eq!` macros

Severity: Informational

Across the codebase, some `assert_eq!` macro (e.g., `appchain:synd-mchain/src/server.rs:76`) usages do not include descriptive error messages. When these assertions fail, they produce generic panic outputs without context about the assertion's intent or variables involved.

Recommendation

We recommend updating all `assert_eq!` usages to include a specific error message explaining the expectation.

Status: Resolved

26. Use of magic numbers decreases maintainability

Severity: Informational

Throughout the codebase, hard-coded number literals without context or a description are used. Using such "magic numbers" goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Instances of magic numbers are listed below:

- `appchain:synd-mchain/src/methods/eth_methods.rs:250`

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Resolved

27. Timestamp truncated to `u32`

Severity: Informational

In `appchain:synd-chain-ingestor/src/db.rs:83-86`, the function stores block timestamps as `u32`, then casts to `u64`:

```
pub fn get_block(&self, block: u64) -> BlockRef {
    // ...
    timestamp: u32::from_be_bytes(data[..4].try_into().unwrap()) as u64,
    // ...
}

pub fn update_block(&mut self, header: &Header, metrics: &ChainIngestorMetrics)
-> BlockUpdateResult {
    // ...
    self.add_block(header.timestamp as u32, header.hash);
    // ...
}
```

Ethereum block timestamps are `u64`, but truncating to `u32` limits valid timestamps to ≤ 2^{32}–1 seconds (≈ Feb 7, 2106).

Hence, casting `header.timestamp` as `u32` will silently wrap/truncate once it is exceeded.

Additionally, even before the year 2106, some testnets or custom chains that use non-Unix epochs or higher precision timestamps (e.g., milliseconds) could overflow sooner.

Recommendation

We recommend storing the timestamp as `u64` values.

Status: Acknowledged

The client states that using 8 bytes for the timestamp is not worth the increase in db size, they can change this later on in 75 years or so.

28. Overly permissive internal network access rules

Severity: Informational

In `terraform:gcp/syndicate-io/syndicate-appchains-testnet/us-central1/vpc/main.tf`, the firewall rule “default-allow-internal” permits all protocols on all ports for the internal network ranges `["10.10.0.0/16", "10.20.0.0/16", "10.30.0.0/20", "10.40.0.0/20"]` with `allow = [{ protocol = "all", ports = [] }]`.

While the internal network boundary provides a solid security foundation, the configuration allows unrestricted communication between all internal hosts. This broad permission set violates the principle of least privilege by not restricting traffic to only necessary ports and protocols.

Recommendation

We recommend implementing more granular firewall rules that restrict internal traffic to specific ports and protocols based on actual service requirements.

Status: Resolved

29. Multiple unresolved TODOs

Severity: Informational

In the terraform configuration files present in the terraform repo, there are multiple `TODO`’s and acknowledged security remarks. Since they seem to be acknowledged, they are not reported in order not to duplicate these items.

Recommendation

We recommend resolving all pending `TODO`s before production deployment.

Status: Acknowledged

The client states that a lot of the TODOs have been fixed since the audit, and many of them were informational for the auditors.

Security Model

The security model has been carefully crafted to delineate the various assets, actors, and underlying assumptions of withdrawal off-chain components for the Syndicate platform. These will then be analyzed in a threat model to outline high-level security risks and proposed mitigations.

The purpose of this security model is to recognize and assess potential threats, as well as the derivation of recommendations for mitigations and counter-measures.

There is a limit to which security risks can be identified by constructing a security/threat model. Some risks may remain undetected and may not be covered in the model described below (see disclaimer).

Assets

The following outlines assets that hold significant value to potential attackers or other stakeholders of the system.

Tokens

Tokens represent valuable assets that carry real-world value. Attackers are primarily motivated to steal or mint these tokens to gain financial profit.

Secret keys

Secret keys represent keys that carry privileged information, including but not limited to API tokens, JWT (JSON Web Token) signing secrets, cloud credentials, and enclave attestation keys. Attackers are motivated to obtain these keys to manipulate the system into performing unintended tasks.

For example, attackers are motivated to obtain enclave attestation keys to forge trusted enclave responses, thereby allowing them to authorize fraudulent withdrawals on behalf of users.

Withdrawal proposals

Structured JSON-RPC messages are used to authorize token withdrawals. If these messages are tampered with or forged, they may allow unauthorized asset transfers. Attackers may try to inject or modify proposals to steal funds from users.

TEE execution environment

The TEE execution environment in AWS Nitro Enclave is responsible for processing withdrawal logic in a cryptographically secure way. Attackers are motivated to compromise the TEE to forge trusted outputs, thereby stealing user funds.

Cloud service providers

The system depends on the correct functioning and security of services provided by GCP (Syndicate Proposer), AWS (Nitro Enclave), and Cloudflare (network-level filtering). These platforms are trusted to enforce network boundaries, isolate workloads, and securely store secrets.

Attackers may attempt to exploit vulnerabilities to abuse misconfigured permissions and compromise trust assumptions (e.g., by hijacking control planes or manipulating infrastructure behavior).

Cloud infrastructure configuration

The cloud providers (e.g., Google Cloud, Amazon Web Services) configuration represents a key factor in deciding the overall system architecture. These include, but are not limited to, firewall rules, IAM policies, and networking setups (e.g., IP whitelisting rules). An innocent misconfiguration may unintentionally expose internal services or allow unauthorized access.

VSOCK communication channel

The VSOCK communication channel is used for enclave-host communications. If the channel allows unauthenticated access or misconfigurations that lead to improper isolation, attackers could spoof or intercept messages to/from the enclave, thereby stealing user funds.

Third-party dependencies and libraries

The protocol relies on multiple third-party components. These include open-source libraries, container images, SDKs, and infrastructure modules (e.g., Terraform providers, RPC libraries, cryptographic primitives).

Attackers are motivated to compromise these dependencies to introduce malicious code, exploit known vulnerabilities, or poison builds, leading to a supply chain compromise. For example, compromising a dependency used in the enclave could undermine system integrity or leak sensitive data.

Privileged human access

Developers and operators possess credentials and knowledge that can grant access to sensitive infrastructure, signing keys, and deployment controls.

Attackers may attempt physical coercion (e.g., threats, bribery, or social pressure, commonly referred to as a “$5 wrench attack”) to extract secrets directly from individuals, bypassing technical protections entirely.

Stakeholders/Potential Threat Actors

The following outlines the various stakeholders or potential threat actors that interact with the system.

Users

Individuals or organizations that interact with the Syndicate platform to initiate withdrawal requests.

Users are assumed to act in their financial interest and may become opportunistic threat actors when incentives align. For example, in the Nomad Bridge hack, users copied a publicly visible exploit transaction to drain funds, causing further damage to the protocol.

Syndicate developers and operators

Actors responsible for governing the Syndicate platform, including maintaining the TEE infrastructure, proposer logic, and cloud deployments. If the developers become malicious, they may present as an insider threat.

Additionally, attackers may attempt to compromise their accounts (e.g., via phishing, malware, or coercion) to obtain privileged access to the system.

Cloud service providers

Cloud service providers are core components used by Syndicate as part of the off-chain withdrawal system. For example, the Syndicate Proposer is hosted in Google Cloud, the TEE runs inside an Amazon Web Services Nitro Enclave, and Cloudflare is used for network-level filtering and access control.

Third-party dependency maintainers

Authors or maintainers of external libraries, container images, or infrastructure modules used in the system. If the maintainer's account is compromised or becomes malicious, they can introduce backdoors, vulnerabilities, or undesired behavior into the software supply chain.

Appchain validators

Validators in Syndicate Appchains are responsible for signing and confirming on-chain withdrawal requests. Malicious or colluding validators may attempt to submit fraudulent withdrawal messages, delay legitimate transactions, or manipulate off-chain inputs to the proposer or enclave.

Off-chain system components

These are custom-built software components that handle withdrawal request processing, message translation, and interaction with the TEE or Appchains. Since they act as intermediaries between user input and trusted execution environments, a bug in any of these services can compromise the security of the entire withdrawal flow.

Attackers may exploit vulnerabilities such as logic flaws, unchecked inputs, or unsafe dependencies in these components to manipulate proposals, bypass verification, or trigger unauthorized fund transfers.

Bots

MEV (miner/maximal extractable value) bots represent autonomous actors that scan mempools to identify profitable transaction sequencing opportunities. They can repeat, copy, front-run, or back-run any transaction, which in return provides value extraction from the protocols. These bots operate independently from users and system components and may be controlled by malicious actors.

Assumptions

The following points outline the critical assumptions that form the foundation for the system's effective operation, ensuring its stability, security, and reliability.

Trusted TEE integrity and attestations

The AWS Nitro Enclave is assumed to operate as designed, with hardware-backed isolation and attestation mechanisms that cannot be bypassed or forged by unprivileged actors.

Cloud providers enforce isolation and access controls correctly

GCP, AWS, and Cloudflare are assumed to implement network boundaries, IAM policies, and service-level isolation correctly. No unauthorized access or lateral movement is expected within the assumed configuration boundaries.

Syndicate infrastructures are deployed from trusted sources

It is assumed that the core components (e.g., `synd-withdrawals`, `synd-translator`) are built and deployed from audited source code and not tampered with during the CI/CD or deployment process.

Secrets are properly stored and periodically rotated

Secrets such as API keys, enclave attestation keys, and cloud credentials are assumed to be securely stored using services like AWS Secrets Manager or GCP Secret Manager. These secrets are also expected to be rotated regularly to reduce exposure risk and limit the attacker's ability to move laterally within the system.

Enclave output should be trusted only if the attestation is valid

Consumers of the enclave's output result (e.g., proposers or relayers) are expected to verify attestation reports before accepting any signed data from the enclave to ensure they are trusted and valid. Invalid outputs should be rejected.

Timely human intervention is assumed in the event of anomalies

It is assumed that if a critical anomaly, compromise, or unexpected behavior is detected (e.g., enclave failure, suspicious withdrawals, or alert spikes), responsible operators will respond promptly to investigate, contain, and mitigate the issue. This is because the system relies on this human oversight to handle edge cases and failures that automated components may not fully address. For example, if an alert indicates unauthorized access, human intervention is expected to review and respond appropriately.

Monitoring and incident response

The system assumes that a qualified operational team is available 24/7 to monitor alerts and respond to incidents in real time. Additionally, monitoring infrastructure should be configured to distinguish high-priority security events from routine noise to ensure effective triage.

In the event of a compromise, unauthorized access, or enclave failure, operators are expected to intervene promptly and follow established procedures such as access revocation, key rotation, or enclave reinitialization. These response procedures should be documented and periodically tested to ensure readiness and practicality.

Append-only database integrity

The custom append-only file format used by `synd-chain-ingestor` (with its specific header structure and block items) is assumed to maintain integrity even under failure conditions. The exclusive file locking mechanism must prevent concurrent access corruption, and automatic truncation of corrupted entries is assumed to be reliable without data loss.

Terraform cloud automation security

The automated infrastructure deployment via Terraform Cloud's VCS integration is assumed to be secure from tampering. The threat model does not include possible zero-day security issues in the Terraform/Cloud layer (other than possible misconfigurations).

System capacity under normal operations

The system is assumed to have sufficient capacity to handle its expected operational workload without resource exhaustion. This includes processing withdrawal requests, ingesting blockchain data, and coordinating between components under normal usage patterns without degradation of service.

Threat Model

Process Applied

The process performed to analyze the system for potential threats and build a comprehensive model is based on the approach first pioneered by Microsoft in 1999 that has developed into the STRIDE model

(https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)).

Whilst STRIDE is aimed at traditional software systems, it is generic enough to provide a threat classification suitable for blockchain applications with little adaptation (see below).

The result of the STRIDE classification has then been applied to a risk management matrix with simple countermeasures and mitigations suitable for blockchain applications.

STRIDE Interpretation in the Blockchain Context

STRIDE was first designed for closed software applications in permissioned environments with limited network capabilities. However, the classification provided can be adapted to blockchain systems with small adaptations. The table below highlights a blockchain-centric interpretation of the STRIDE classification:

Spoofing	In a blockchain context, the authenticity of communications is built into the underlying cryptographic public key infrastructure. However, spoofing attack vectors can occur at the off-chain level and within a social engineering paradigm. An example of the former is a Sybil attack where an actor uses multiple cryptographic entities to manipulate a system (wash-trading, auction smart contract manipulation, etc.). The latter usually consists of attackers imitating well-known actors, for instance, the creation of an impersonation token smart contract with a malicious implementation.
Tampering	Similarly to spoofing, tampering of data is usually not directly relevant to blockchain data itself due to cryptographic integrity. It can still occur though, for example through compromised developers of the protocol that have access to deployment keys or through supply chain attacks that manages to inject malicious code or substitutes trusted software that interacts with

	the blockchain (node software, wallets, libraries).
Repudiation	Repudiation, i.e., the ability of an actor to deny that they have taken action, is usually not relevant at the transaction level of blockchains. However, it makes sense to maintain this category since it may apply to additional software used in blockchain applications, such as user-facing web services. An example is the claim of a loss of private keys and assets.
Information Disclosure	Information disclosure has to be treated differently at the blockchain layer and the off-chain layer. Since the blockchain state is inherently public in most systems, information leakage here relates to data that is discoverable on the blockchain, even if it should be protected. Predictable random number generation could be classified as such, in addition to simply storing private data on the blockchain. In some cases, information in the mempool (pending/unconfirmed transactions) can be exploited in front-running or sandwich attacks. At the off-chain layer, the leakage of private keys is a good example of operational threat vectors.
Denial of Service	Denial of service threat vectors translates directly to blockchain systems at the infrastructure level. At the smart contract or protocol layer, there are more subtle DoS threats, such as unbounded iterations over data structures that could be exploited to make certain transactions not executable.
Elevated Privileges	Elevated privilege attack vectors directly translate to blockchain services. Faulty authorization at the smart contract level is an example where users might obtain access to functionality that should not be accessible.

STRIDE Classification

The following threat vectors have been identified using the STRIDE classification, grouped by components of the system.

TEE (AWS Nitro Enclave)	
Spoofing	● Host VM forges messages or pretends to be the enclave to submit fraudulent approvals.
Tampering	● The attacker modifies the enclave input or output on the host before/after VSOCK communication.
Repudiation	● The host denies forwarding of messages or claims that the enclave never signed an output.
Information Disclosure	● Secrets or proposals leak from the enclave to an untrusted host due to misconfiguration or side channel attacks.
Denial of Service	● The host starves the enclave of resources or drops all VSOCK requests, halting withdrawals.
Elevated Privileges	● The host bypasses attestation validations to accept fake or invalid enclave outputs.

Syndicate proposer (GCP)	
Spoofing	● Unauthorized entities impersonate proposers and submit withdrawal proposals.
Tampering	● The proposer modifies the original withdrawal request (e.g., amount, destination address) before sending it to the enclave.
Repudiation	● The operator claims they did not submit fraudulent proposals.
Information Disclosure	● Logs or misconfigured storage leak sensitive information, such as user balances, secrets, or enclave responses.
Denial of Service	● Flooding the proposer with fake or invalid withdrawal requests exhausts memory or queue capacity.
Elevated Privileges	● The attacker gains control of the proposer and bypasses all withdrawal validations.

synd-withdrawals and synd-translator components	
Spoofing	● Malicious relayers or appchains pretend to be trusted middlewares or withdrawal senders.
Tampering	● The translator modifies messages between the source chain and the proposer to reroute or modify withdrawals.
Repudiation	● The malicious actor denies translating or forwarding malicious messages.
Information Disclosure	● Sensitive cross-chain message payloads or token metadata are logged.
Denial of Service	● Attackers spam cross-chain messages to crash the parser.
Elevated Privileges	● Attackers trigger code paths reserved for system-only actions (e.g., finalize without checks).

Infrastructure (CI/CD, secrets, networking)	
Spoofing	● Attackers impersonate infrastructure services (e.g., GitHub Action runner, Terraform provider).
Tampering	● The CI/CD pipeline is compromised to push malicious enclave/proposer builds.
Repudiation	● Team denies responsibility for leaked secrets or misconfigurations.
Information Disclosure	● Terraform state files leak secrets, logs, or metadata that expose internal IPs or credentials.
Denial of Service	● Infrastructure misconfiguration that causes outage (e.g., firewall blocks vsock or GCP triggers rate limits).
Elevated Privileges	● IAM misconfiguration that allows attackers to escalate into deploy permissions or read secrets.

Mitigation Matrix

The following mitigation matrix describes each of the threat vectors identified in the STRIDE classification above, assigning an impact and likelihood and suggesting countermeasures and mitigation strategies. Countermeasures can be taken to identify and react to a threat, while mitigation strategies prevent a threat or reduce its impact or likelihood.

TEE (AWS Nitro Enclave)

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Host VM forges messages or pretends to be the enclave to submit fraudulent approvals. An attacker controlling the host VM may spoof enclave-generated messages, tricking external systems into accepting unauthorized withdrawal authorizations.	High	Low	Require strict attestation checks before accepting enclave outputs.	Verify the signer's identity before trusting outputs.
The attacker modifies the enclave input or output on the host before/after VSOCK communication. An attacker with control over the host OS may alter data before it reaches the enclave or modify signed outputs to redirect funds or corrupt data flow.	High	Low	Encrypt messages between the host and the enclave with authenticated encryption	Consider using message authentication codes (MACs) or signatures for data integrity.
The host denies forwarding of messages or claims that the enclave never signed an	Medium	Low	Require enclaves to sign all responses with nonces to	Implement timeouts and watchdogs that alert when enclave responses are

output. A malicious host can drop enclave responses or falsely claim that no output was received, disrupting the withdrawal flow and removing auditability.			ensure accountability.	missing or delayed.
Secrets or proposals leak from the enclave to an untrusted host due to misconfiguration or side channel attacks. Sensitive enclave data, such as secrets, proposal logic, or decrypted inputs, may be exposed to the host through improper boundary controls or speculative execution leaks.	High	Low	Enforce strict enclave-host separation. Minimize and review enclave output.	Audit enclave code for side channels
The host starves the enclave of resources or drops all VSOCK requests, halting withdrawals. A malicious or misconfigured host may block VSOCK communication or restrict CPU/RAM usage, causing the enclave to become unresponsive or inoperable.	Medium	Medium	Monitor enclave health and enforce CPU/memory reservations.	Trigger alerts and automatic recovery if the enclave stops responding.
The host bypasses attestation validations to accept fake or invalid enclave outputs.	High	Low	Always verify attestation evidence, including its freshness and	Reject outputs that fail freshness, signature, or measurement

If attestation results are not correctly verified, a malicious host could spoof or reuse old outputs, tricking other components into accepting forged results.			the identity of the signer.	checks

Syndicate proposer (GCP)

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Unauthorized entities impersonate proposers and submit withdrawal proposals. Attackers may exploit vulnerabilities to submit fraudulent withdrawal proposals, potentially draining user funds by routing them to attacker-controlled addresses.	High	Low	Implement service account authentication with short-lived tokens. Use VPC Service Controls to create security perimeters. Enforce mTLS for all proposer communications.	Monitor for authentication failures and proposals from unexpected sources. Implement anomaly detection on proposal patterns.
The proposer modifies the original withdrawal request (e.g., amount, destination address) before sending it to the enclave. A compromised or malicious proposer could alter withdrawal details to steal funds or redirect them to unauthorized recipients.	High	Low	Sign withdrawal requests at the origin with user keys. Implement immutable audit trails. Use multiple proposer instances with consensus requirements.	Cross-verify proposals against on-chain events. Alert on any proposal modifications detected.
The operator claims they did not submit fraudulent proposals. Malicious operators could deny responsibility for fraudulent actions, making attribution and recovery difficult.	Medium	Medium	Implement comprehensive logging with operator identity tracking. Use hardware security keys	Regular audit reviews of operator actions. Implement multi-signature requirements for sensitive operations.

			for operator actions. Record all actions in immutable audit logs.	
Logs or misconfigured storage leak sensitive information, such as user balances, secrets, or enclave responses. Exposed logs could reveal API keys, JWT signing secrets, enclave attestation keys, or internal service endpoints that attackers could weaponize to forge requests or bypass authentication.	High	Medium	Encrypt logs at rest. Exclude secrets from logs entirely. Use GCP Secret Manager for all credentials. Implement log redaction for any accidentally logged secrets.	Scan logs for entropy patterns indicating leaked keys. Alert on any access to log storage from unexpected sources.
Flooding the proposer with fake or invalid withdrawal requests exhausts memory or queue capacity. Denial-of-service attacks prevent legitimate withdrawals from being processed, resulting in service disruption.	Medium	Medium	Implement rate limiting. Implement priority queues based on withdrawal value. Deploy auto-scaling with resource limits.	Monitor queue depth and processing latency. Implement circuit breakers for overload.
The attacker gains control of the proposer and bypasses all withdrawal validations. Proposer compromise exposes signing keys for	High	Low	Store signing keys in GCP HSM. Rotate keys on each	Monitor key usage patterns. Implement emergency key revocation

enclave communication, allowing forged attestations to be created.			deployment. Use workload identity federation.	procedures.

synd-withdrawals and synd-translator components

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Unauthorized entities impersonate proposers and submit withdrawal proposals. Malicious actors may spoof legitimate proposers to inject unauthorized proposals into the flow.	High	Medium	Authenticate proposers with signatures and origin verification.	Only accept proposals from an allowlist.
The proposer modifies the original withdrawal request (e.g., amount, destination address) before sending it to the enclave. A compromised or buggy proposer may alter user-supplied withdrawal data, resulting in redirected transfers.	High	Low	Enforce input validation before enclave submission.	Require hash commitments of original user inputs in the enclave validation logic.
The operator claims they did not submit fraudulent proposals. A malicious proposer operator could deny involvement in fraudulent actions.	Medium	Low	Require proposers to sign each proposal cryptographicall y.	Store signed logs with timestamps for enhanced traceability and audit purposes.
Logs or misconfigured storage leak sensitive information, such as user balances, secrets, or enclave responses. If logs or storage buckets	High	Medium	Avoid logging sensitive fields. Restrict log access to specific users.	Implement structured logging with redaction. Secure storage with fine-grained IAM

are public or misconfigured, sensitive proposal or enclave data may be exposed.				access control.
Flooding the proposer with fake or invalid withdrawal requests exhausts memory or queue capacity. Attackers could submit malformed or excessive requests to degrade proposer availability or stall withdrawal flow.	Medium	Medium	Implement input validation and rate limiting.	Throttle inputs per source and reject malformed messages early.
The attacker gains control of the proposer and bypasses all withdrawal validations. Full compromise of the proposer allows an attacker to submit arbitrary requests to the enclave, effectively bypassing the protocol's business logic.	High	Low	Enforce least privilege and hardened deployment practices.	Audit proposer activity. Alert on anomalous behavior. Rotate credentials regularly.

Infrastructure (CI/CD, secrets, networking)

Threat Vector	Impact	Likelihood	Mitigation	Countermeasures
Attackers impersonate infrastructure services (e.g., GitHub Action runner, Terraform provider). Supply chain attack injects key extraction code into builds.	High	Low	Pin provider checksums.	Monitor provider binary changes. Alert on new runner registrations for review.
The CI/CD pipeline is compromised to push malicious enclave/proposer builds. Malicious builds contain backdoors or modified attestation logic.	High	Low	Implement reproducible builds with multiple builders. Require artifact consensus.	Monitor behavioral changes. Use signed builds.
Leaked secrets or misconfigurations. Insider leaks HSM credentials or master signing keys.	Medium	Low	Use “M-of-N” key sharding. Require hardware tokens for access. Record sensitive operations.	Monitor key usage patterns. Implement PAM process/solution for sensitive operations.
Terraform state files leak secrets, logs, or metadata that expose internal IPs or credentials. The state contains HSM pins, service account	High	Medium	Exclude secrets from the state.	Scan for entropy patterns. Alert on state downloads.

keys, or enclave signing credentials.				
Infrastructure misconfiguration that causes outage (e.g., firewall blocks VSOCK or GCP triggers rate limits). Service disruption during critical withdrawal periods.	Medium	Medium	Implement change freezes during high activity. Use gradual rollouts. Test against production patterns.	Monitor availability metrics. Automate rollback procedures.
IAM misconfiguration that allows attackers to escalate into deploy permissions or read secrets. Excessive permissions allow key theft or logic modification.	High	Medium	Separate projects for key management. Use VPC Service Controls. Require organization policy approval for IAM changes.	Alert on key project access. Monitor escalation attempts.