Security Audit Report

Syndicate Stage 1C

Milestone 2

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 Audit of the Syndicate Appchain source code and infrastructure.

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, which 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 target:

Repository	https://github.com/SyndicateProtocol/syndicate-appchains
Commit	`0ad0f5e6e09607e070ec4b023d4c7541936ac994`
Scope	● synd-maestro module (all files in synd-maestro/) ● synd-batch-sequencer module (all files in synd-batch-sequencer/)
Fixes verified at commit	● https://github.com/SyndicateProtocol/syndicate-appchains/commit/92f4e322a5fbcddc7d90f5b9967222370614276a ● https://github.com/SyndicateProtocol/terraform/commit/8afd35723612a01341cf80b660c85f7937ba0aaf

	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
Commit	`35e4806b3dd25084286173e38025b5542a291af9`
Scope	● aws/ directory ● set of four readonly IAM accounts provided to auditors
Fixes verified at commit	– 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.

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 audit covers security assessment of the Syndicate Appchain infrastructure components including:

- synd-maestro: Orchestration module for managing Syndicate appchain operations
- synd-batch-sequencer: Batch sequencing and transaction aggregation module
- Terraform AWS infrastructure: Infrastructure as Code for AWS deployment
- AWS cloud configuration: Security review of IAM, database, and access policies

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**	-
Test coverage	**Medium-High**	-

Summary of Findings

No	Description	Severity	Status
1	Private Key may be leaked due to being logged in plaintext	Major	Resolved
2	Hardcoded `'0-0'` Stream ID triggers redundant batch resubmissions	Major	Resolved
3	IAM user with excessive privileges	Major	Resolved
4	Reverts might go unnoticed when `wait_for_receipt` is false	Minor	Acknowledged
5	Unrestricted HTTP outbound traffic in EC2 security group	Minor	Resolved
6	Missing validation of Chain Id may lead to transaction failures	Minor	Resolved
7	`XDEL` failures might bloat the memory	Informational	Resolved
8	`ReSubmit` shadows the `id` causing incorrect debug output	Informational	Resolved
9	Duplicate field name leading to incorrect logging	Informational	Resolved
10	Misleading compression ratio when the compression is disabled	Informational	Resolved
11	Inverted compression formula produces misleading results	Informational	Resolved

Detailed Findings

1. Private Key may be leaked due to being logged in plaintext

Severity: Major

In `synd-batch-sequencer/src/main.rs:19`, the code logs the entire configuration, including the private key:

```
info!("BatcherConfig: {:?}", config);
```

Because `Debug` on `BatcherConfig` includes the `private_key` field, the signing key is exposed in plaintext logs.

Recommendation

We recommend logging only nonsensitive fields explicitly.

Status: Resolved

2. Hardcoded '`0-0`' Stream ID triggers redundant batch resubmissions

Severity: Major

In `synd-batch-sequencer/src/batcher.rs:101`, on each startup, the batcher initializes its Redis stream consumer with the literal start ID `"0-0"`. This value becomes the in memory `last_id`, so every restart issues an `XREAD` beginning with `"0-0"`, replaying and resubmitting all historical messages.

This results in duplicate batches and in turn duplicate on-chain transactions as per the below code fragments:

```
    let stream_consumer = StreamConsumer::new(
         conn.clone(),
         config.chain_id,
         "0-0".to_string(),
         Arc::new(valkey_metrics),
     );
```

Further, in `consumer.rs`:

```
    pub fn new(
         conn: ConnectionManager,
         chain_id: ChainId,
         start_from_id: String,
         valkey_metrics: Arc<ValkeyMetrics>,
     ) -> Self {
         // NOTE maybe we don't need ConnectionManager here (unless we
want to have multiple
         // consumers per service)
         Self { conn, stream_key: tx_stream_key(chain_id), last_id:
start_from_id, valkey_metrics }
     }
```

and:

```
    pub async fn recv(
         &mut self,
         max_msg_count: usize,
         block_duration: Duration,
     ) -> eyre::Result<Vec<(Vec<u8>, String)>, StreamError> {
         let opts = StreamReadOptions::default()
             .block(block_duration.as_millis() as usize)
             .count(max_msg_count);

         let reply: Option<StreamReadReply> = with_cache_metrics!(
             self.valkey_metrics,
             self.conn.xread_options(&[self.stream_key.as_str()],
&[self.last_id.as_str()], &opts)
         )?;
 // .. //
         if let Some(last_id) = reply.keys[0].ids.last() {
             self.last_id = last_id.id.clone();
         }
```

Recommendation

We recommend persisting the last seen ID after each successful batch (e.g. in Redis, a database, or a file), and on startup read that stored ID instead of `"0-0"`.

Status: Resolved

3. IAM user with excessive privileges

Severity: Major

In the AWS profiles `088040914170_ReadOnlyAccess`, `137805458632_ReadOnlyAccess`, the IAM user `terraform-service-account` has been granted the AWS-managed `IAMFullAccess` policy along with multiple other full-access policies. This configuration provides unrestricted permissions to modify all IAM resources, including user creation, access key generation, and policy attachment capabilities. Compromise of this user's static credentials would enable immediate privilege escalation to full account administrator. Below listing shows the result of AWS CLI command `aws iam list-attached-user-policies --user-name terraform-service-account`

```
"AttachedPolicies": [ "AmazonSSMFullAccess", "AmazonEC2FullAccess",
"IAMFullAccess", "CloudWatchLogsFullAccess", "AmazonVPCFullAccess",
"AmazonS3FullAccess" ]
```

Recommendation

We recommend detaching the `IAMFullAccess` policy as well as other privileged policies and replacing it with an IAM role configured with granular permissions specific to Terraform's operational requirements following the Principle of Least Privilege.

Status: Resolved

4. Reverts might go unnoticed when `wait_for_receipt` is false

Severity: Minor

In `synd-batch-sequencer/src/batcher.rs:296`, when `wait_for_receipt` is set to `false`, the receipt check block is entirely skipped. This is problematic, because in such cases any on chain failure (`revert` or `status=false`) will not be detected.

Recommendation

We recommend always fetching and inspecting the receipt regardless of the flag, or at minimum log a warning when `wait_for_receipt` is disabled to highlight that failures may go unnoticed.

Status: Acknowledged

5. Unrestricted HTTP outbound traffic in EC2 security group

Severity: Minor

In `aws/syndicate/mainnet/us-east-2/ec2/main.tf`, the security group configuration allows unrestricted HTTP outbound traffic to all destinations (`0.0.0.0/0`).

This is problematic because such a broad egress rule permits instances to communicate with any external endpoint over HTTP, potentially enabling data exfiltration or command-and-control communications.

Recommendation

We recommend restricting egress rules to specific IP ranges or endpoints required for known business or technical operations.

Status: Resolved

6. Missing validation of Chain Id may lead to transaction failures

Severity: Minor

In `synd-batch-sequencer/src/config.rs:107-112`, the batcher's RPC ping logic connects and calls `eth_chainId`, but doesn't compare the returned chain ID to the configured `chain_id`.

Maestro's configuration already includes this validation. It fetches `get_chain_id` and compares it against each `chain_rpc_urls` key, erroring on mismatch. However, the batcher's `ping_sequencing_rpc_url` doesn't perform this check.

This allows misconfigured RPC URLs pointing to wrong networks to pass startup validation. The impact is potential silent failures during on-chain submissions when the batcher attempts to interact with the wrong network, causing transaction failures that could have been caught at initialization.

Recommendation

We recommend adding an explicit chain ID comparison in `ping_sequencing_rpc_url`.

Status: Resolved

7. XDEL failures might bloat the memory

Severity: Informational

In `synd-maestro/src/valkey/streams/producer.rs:212`, if the `XDEL` call fails in below code fragment, old entries won't be removed and they will accumulate in the Redis stream, leading to unbounded growth / memory bloat. After fetching all entries older than `finalization_ms`, the application tries to delete them:

```
 // Collect IDs for deletion and call XDEL on all of them
         let ids: Vec<String> = entries.iter().map(|(id, _)|
id.clone()).collect();
         if let Err(e) = with_cache_metrics!(
             valkey_metrics,
             producer_conn.xdel::<_, _, usize>(&stream_key, &ids) <<<
         ) {
             error!(%stream_key, %max_id, %e, "Failed to delete finalized
transaction entries");
             return Err(e);
         }
         trace!(%stream_key, %max_id, count = ids.len(), "Deleted
entries");
         Ok(entries)
```

Since the code immediately returns on error, those old entries stay in the stream. Over time, especially under load or persistent `XDEL` failures, the stream might grow without bound, risking Redis memory exhaustion. In the worst case this may cause Denial of Service.

Recommendation

We recommend implementing a simple retry or fallback (e.g. skipping deletion but marking for later) to prevent unbounded bloat.

Status: Resolved

8. `ReSubmit` shadows the `id` causing incorrect debug output

Severity: Informational

In `synd-maestro/src/valkey/streams/producer.rs:351-352`, in the `ReSubmit` branch, the debug linepresented below prints the new stream ID twice and doesn't show the original. That makes it impossible to correlate which old entry got resubmitted.

```
Ok(id) => {
    debug!(%stream_key, original_id = %id, new_id = %id, "Finalization
checker: Resubmitted transaction.");
}
```

As a result, this makes the debugging process more difficult.

Recommendation

We recommend avoiding shadowing the variable. For example:

```
match Self::add_to_stream(...).await {
    Ok(new_id) => {
        debug!(
            %stream_key,
            original_id = %id,   // outer id
            new_id      = %new_id, // fresh id from add_to_stream
            "Finalization checker: Resubmitted transaction."
        );
    }
    ...}
```

Status: Resolved

9. Duplicate field name leading to incorrect logging

Severity: Informational

In `synd-maestro/src/maestro.rs:792-795`, when the number of deleted keys doesn't match what was requested, it's logged as:

```
error!(
    %chain_id, %wallet_address,
    num_txns_requested = %txn_ids.len(),
    num_txns_requested = %result, // <- duplicate field name
    "Removed different number of waiting transactions from cache than
requested"
);
```

Because the `num_txns_requested` is used twice, it's not logged how many were actually removed.

Recommendation

We recommend giving the second field a distinct name, for example:

```
error!(
    %chain_id, %wallet_address,
    num_txns_requested = txn_ids.len(),
    num_txns_removed   = result,
    "Removed different number of waiting transactions from cache than
requested"
);
```

Status: Resolved

10. Misleading compression ratio when the compression is disabled

Severity: Informational

In `synd-batch-sequencer/src/batcher.rs:225-229`, when compression is `disabled`, the original batch size and compressed batch size are identical. The compression ratio metric computes `(original_size ÷ compressed_size) × 100`, so it always evaluates to 100.

This is problematic, because in such cases the dashboard will show a constant 100% compression ratio for uncompressed batches, which is misleading. This is shown in the below code:

- in `batcher.rs`:

```
let proposed_batch = match self.compression_enabled {
    true  => compress_batch(&txs, tx_bytes)?,
    false => uncompressed_batch(&txs, tx_bytes),
};

self.metrics.record_batch_transactions(txs.len());
self.metrics.record_compression_ratio(uncompressed_size, batch.len());
```

- in `metrics.rs`:

```
pub fn record_compression_ratio(&self, original_size: usize,
compressed_size: usize) {
    if original_size > 0 {
        let ratio = (original_size as f64 / compressed_size as f64 *
100.0) as i64;
        self.batch_compression_ratio.set(ratio);  // always 100 when
uncompressed
        self.record_batch_size(compressed_size);
    }
}
```

Recommendation

We recommend only calling `record_compression_ratio` when `compression_enabled == true`.

And in uncompressed mode, setting the metric to a neutral value (e.g. `0`) or skip recording entirely.

Status: Resolved

11. Inverted compression formula produces misleading results

Severity: Informational

In `synd-batch-sequencer/src/metrics.rs:87-93`, the compression ratio formula is inverted, using `original_size / compressed_size * 100` instead of the standard `compressed_size / original_size * 100`. This causes compressed data to incorrectly report values greater than `100%` which may lead to reporting incorrect compression values.

Recommendation

We recommend using the standard percent of original formula, for instance:

```
let ratio = (compressed_size as f64 / original_size as f64 * 100.0) as
i64;
```

Status: Resolved