Security Audit Report

Syndicate Stage 1C

Milestone 1

v1.0

September 9, 2025

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

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 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	`9d498cd48ac724751f8263f9d35ea585e3e70e3f`
Scope	The scope was restricted to the changes in the following files from commit `2ddcabf03944ff6003e690b8953b95811af7013f`: ● `synd-withdrawals/server/main.go` ● `synd-withdrawals/synd-enclave/cmd/enclave/main.go` ● `synd-withdrawals/synd-enclave/enclave/db.go` ● `synd-withdrawals/synd-enclave/enclave/server.go` ● `synd-withdrawals/synd-enclave/enclave/types.go` ● `synd-withdrawals/synd-enclave/enclave/verify.go` ● `synd-withdrawals/synd-enclave/enclave/wavmio/stub.go` ● `synd-withdrawals/synd-proposer/cmd/synd-proposer/main.go` ● `synd-withdrawals/synd-proposer/logger/logger.go` ● `synd-withdrawals/synd-proposer/metrics/metrics.go` ● `synd-withdrawals/synd-proposer/pkg/config/config.go` ● `synd-withdrawals/synd-proposer/pkg/helpers.go` ● `synd-withdrawals/synd-proposer/pkg/proposer.go` ● `synd-withdrawals/synd-proposer/pkg/tls/tls.go` ● `synd-withdrawals/synd-proposer/server/server.go` ● `synd-withdrawals/synd-tee-attestation-zk-proofs/aws-nitro/src/attestation_document.rs` ● `synd-withdrawals/synd-tee-attestation-zk-proofs/proof-submitter/src/lib.rs` ● `synd-withdrawals/synd-tee-attestation-zk-proofs/proof-submitter/src/main.rs` ● `synd-withdrawals/synd-tee-attestation-zk-proofs/sp1/program/src/main.rs` ● `synd-withdrawals/synd-tee-attestation-zk-proofs/sp1/script/src/bin/evm.rs` ● `synd-withdrawals/synd-tee-attestation-zk-proofs/sp1/script/src/bin/main.rs`
Fixes verified at commit	`23d981df25fedead460cd82084d5d4896e1b9637`

	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 scope of the audit features the core protocol functionality and off-chain components implemented for Syndicate Withdrawals.

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	Missing validation of the receipt status	Critical	Resolved
2	`CombineAppchainProofs` fails to check errors during signing inputs	Major	Resolved
3	Incorrect log count logic causes `maxQty` validation to be ineffective	Major	Acknowledged
4	Non-fatal errors on failures may lead to incorrect rollup detection	Major	Resolved
5	Malicious `EigeanDA` providers could exhaust memory	Major	Acknowledged
6	The private key is logged	Major	Resolved
7	`getLogs` will always revert if the error is in the lower half	Major	Acknowledged
8	`getLogs` not querying the lower half	Major	Resolved
9	Attestation document parser allows invalid PCR blob lengths for SHA-384	Major	Resolved
10	Max quantity parameter is not enforced during log retrieval	Minor	Acknowledged
11	Missing error return when no DAS reader is configured	Minor	Resolved
12	Missing error handling for `json.Marshal`	Minor	Resolved
13	Error masking in the signature method	Minor	Resolved
14	Uninitialized `PendingAssertion` may lead to panic	Informational	Acknowledged
15	Repeated assertion submissions	Informational	Acknowledged
16	Logging structures can be improved	Informational	Resolved
17	Blocks are not validated to increase monotonically	Informational	Resolved
18	Curve is not validated to be at the point	Informational	Acknowledged
19	The ticker is not stopped when the context is	Informational	Resolved

	reached		
20	Panic on RPC URL parsing failure	**Informational**	**Resolved**
21	Server listener settings may be overly permissive	**Informational**	**Resolved**
22	Leftover TODO comment	**Informational**	**Acknowledged**

Detailed Findings

1. Missing validation of the receipt status

Severity: Critical

In `synd-withdrawals/synd-tee-attestation-zk-proofs/proof-submitter/src/main.rs:313-336`, the status of the `receipt` is not validated in the `submit_proof_to_chain` method.

```
async fn submit_proof_to_chain<P: Provider>(
    contract: TeeKeyManagerInstance<P>,
    proof: GenerateProofResult,
) -> Result<(), ProofSubmitterError> {
    let tx = contract.addKey(proof.public_values.into(),
proof.proof.into());

    let receipt = tx
        .send()
        .await
        .map_err(|e| {
            info!("Error sending transaction: {e}");
            ProofSubmitterError::SubmitProofToChain(e.to_string())
        })?
        .get_receipt()
        .await
        .map_err(|e| {
            info!("Error getting receipt: {e}");
            ProofSubmitterError::SubmitProofToChain(e.to_string())
        })?;

    info!("Successfully submitted proof to chain. Receipt:
{receipt:?}");

    Ok(())
}
```

If the contract call reverts, for example, because the proof was malformed or gas was insufficient, the RPC node still returns a receipt with a status field set to `zero`.

Since the `get_receipt` function does not treat a revert as an error, the code logs success and returns `Ok(())` even though the on-chain operation failed.

Recommendation

We recommend verifying the status field on the `receipt` and returning an `error` if it is zero.

Status: Resolved

2. `CombineAppchainProofs` fails to check errors during signing inputs

Severity: Major

In `synd-withdrawals/synd-enclave/enclave/server.go:666`, errors that may be returned from the `sign` method are not handled.

```
input.Outputs[1].sign(&input.Inputs[0], s.signerKey)
```

Consequently, if a signing ever fails, the `CombineAppchainProofs` function will carry on as if nothing happened, leaving the `Signature` field blank or invalid.

Recommendation

We recommend handling the errors that may be returned from the `sign` method.

Status: Resolved

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

Severity: Major

In `synd-withdrawals/synd-proposer/pkg/helpers.go:71`, the logic enforces that if the `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 number 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:

- `synd-withdrawals/synd-proposer/pkg/helpers.go:239`
- `synd-withdrawals/synd-proposer/pkg/helpers.go:274`

Recommendation

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

Status: Acknowledged

The client states that `maxQty` is the minimum count at which the log search ends early. This variable can potentially be renamed to, eg, `targetQty` instead, but this is not a bug. The `maxQty` validation is just a best-effort sanity check.

4. Non-fatal errors on failures may lead to incorrect rollup detection

Severity: Major

In `synd-withdrawals/synd-proposer/pkg/proposer.go:163-170`, when querying the precompile address fails via `p.SettlementClient.CodeAt`, the `pollingLoop` function logs the error as a warning using `log.Warn().Stack().Err(wrappedErr).Msg(msg)`, which logs the error without terminating execution.

This is problematic because the execution will continue, causing the `code` variable to be used to derive the `settlesToArbitrumRollup` flag. The flag will be incorrectly evaluated as `false` since an error occurred (`len(code)` will become zero).

Consequently, the system may incorrectly assume it does not settle to an Arbitrum rollup, causing the proposer to take an incorrect path for proof generation or submission.

Recommendation

We recommend using `log.Fatal().Stack().Err(wrappedErr).Msg(msg)` so the execution will be terminated when an error occurs.

Status: Resolved

5. Malicious `EigeanDA` providers could exhaust memory

Severity: Major

In `synd-withdrawals/synd-enclave/enclave/verify.go:184`, the `QueryBlob` function accepts an `EigenDA` blob of any size and then runs inverse FFT decoding on it via `GenericDecodeBlog`. The decoding allocates a byte slice whose length matches the blob.

A malicious provider can supply a very large blob so that this allocation exhausts the enclave's memory, causing a panic and stopping block verification.

The unchecked size allocation happens in the `EigenDA` decoding stack, in the implementation of `GenericDecodeBlob`:

```
func decodeBlob(data []byte) ([]byte, error) {
      length := binary.BigEndian.Uint32(data[2:6])

      // decode raw data modulo bn254
      decodedData := codec.RemoveEmptyByteFromPaddedBytes(data[32:])

      // get non blob header data
      reader := bytes.NewReader(decodedData)
      rawData := make([]byte, length)//@audit no length check and
validation
      n, err := reader.Read(rawData)
      if err != nil {
            return nil, fmt.Errorf("failed to copy unpadded data into
final buffer, length: %d, bytes read: %d", length, n)
      }
      if uint32(n) != length {
            return nil, fmt.Errorf("data length does not match length
prefix")
      }
      return rawData, nil
}
```

Recommendation

We recommend checking the raw blob length against a safe maximum before decoding. If the blob exceeds that limit, reject it with an error rather than attempting decoding.

Status: Acknowledged

The client states that they may decide not to expose the enclave or tell users to run their own instance, and then size limitations should live at the server application level.

6. The private key is logged

Severity: Major

In `synd-withdrawals/synd-proposer/cmd/synd-proposer/main.go:34`, the configuration is logged:

```
log.Info().Msgf("Config: %+v", cfg)
```

This formats every field of the loaded `cfg` struct into the application log, including the private key. The private key could remain in logs/memory and be exposed.

Recommendation

We recommend masking or omitting sensitive fields from the log output.

Status: Resolved

7. `getLogs` will always revert if the error is in the lower half

Severity: Major

In `synd-withdrawals/synd-proposer/pkg/helpers.go:89-97`, the logs are retrieved from the upper half and then attempt the lower half:

```
logs, err = getLogs(ctx, c, mid+1, endBlock, addresses, topics, maxQty)
if err != nil {
    return nil, err
}
// ...
prevLogs, err := getLogs(ctx, c, startBlock, mid, addresses, topics,
maxQty)
if err != nil {
    return nil, err
}
prevLogs = append(prevLogs, logs...)
return prevLogs, nil
```

If the fetch for lower blocks fails, the function returns `nil, err`. After that, it drops the logs from the upper blocks that had succeeded. This can lead to an entire batch of events vanishing when the lower side of the split experiences an error.

Recommendation

We recommend retaining any retrieved logs before reporting an error. For example:

```
upperLogs, err1 := getLogs(ctx, c, mid+1, endBlock, addresses, topics,
maxQty)
if err1 != nil {
    return upperLogs, fmt.Errorf("error fetching blocks %d-%d: %w",
mid+1, endBlock, err1)
}
```

```
lowerLogs, err2 := getLogs(ctx, c, startBlock, mid, addresses, topics,
maxQty)
if err2 != nil {
    return lowerLogs, fmt.Errorf("error fetching blocks %d-%d: %w",
startBlock, mid, err2)
}

combined := append(lowerLogs, upperLogs...)
return combined, nil
```

Status: Acknowledged

The client states that they do not think partial results are useful in general if they are unable to fetch logs over the full range, so no need to return them.

8. `getLogs` not querying the lower half

Severity: Major

In `synd-withdrawals/synd-proposer/pkg/helpers.go:89-97`, the `getLogs` function splits the requested range and retrieves the upper half first:

```
mid := (startBlock + endBlock) / 2
logs, err = getLogs(ctx, c, mid+1, endBlock, addresses, topics, maxQty)
if err != nil {
    return nil, err
}
if maxQty > 0 && uint64(len(logs)) >= maxQty {
    return logs, nil
}
```

For example, if the call requests logs from block 1000 through block 1007 with `maxQty = 50`, the midpoint is computed as `(1000 + 1007) / 2 = 1003`. The function then retrieves logs from blocks 1004 to 1007.

If that subset yields fifty entries, the function returns immediately and never queries blocks 1000 to 1003. This causes the events in the first half to be ignored without any error or warning.

This pattern produces silent gaps in the event stream whenever the latter half of a range carries enough entries to meet the limit. Downstream systems that rely on a complete, ordered log sequence will observe missing entries in the first half of any such range.

Recommendation

We recommend applying the limit in both recursive calls:

```
mid := startBlock + (endBlock - startBlock) / 2

firstLogs, err := getLogs(ctx, c, startBlock, mid, addresses, topics,
maxQty)

if err != nil {
    return nil, err
}

secondLogs, err := getLogs(ctx, c, mid+1, endBlock, addresses, topics,
maxQty)

if err != nil {
    return nil, err
}

combined := append(firstLogs, secondLogs...)

if maxQty > 0 && uint64(len(combined)) > maxQty {
    return combined[:maxQty], nil
}

return combined, nil
```

Status: Resolved

9. Attestation document parser allows invalid PCR blob lengths for SHA-384

Severity: Major

In `synd-withdrawals/synd-tee-attestation-zk-proofs/aws-nitro/src/attestation_document.rs:123`, the function validates the values as;

```
if value.is_empty() || !(value.len() == 32 || value.len() == 48 ||
value.len() == 64) {
```

The Natspec comment in the Solidity binding makes clear that each PCR entry in the raw attestation document is a 48-byte SHA-384 output.

The parser first requires that `doc.digest == "SHA384"`, but then still allows PCR blobs of 32 or 64 bytes alongside the correct 48:

```
if value.is_empty() || !(value.len() == 32 || value.len() == 48 ||
value.len() == 64) {
    return Err(VerificationError::BadPCRValue);
}
```

Consequently, allowing any other length creates a risk that a malformed or truncated PCR value will be accepted.

Recommendation

We recommend demanding exactly 48 bytes when `digest == "SHA384"`, or drive length validation from a map of algorithm names to their fixed output sizes.

Status: Resolved

10. Max quantity parameter is not enforced during log retrieval

Severity: Minor

In `synd-withdrawals/synd-proposer/pkg/helpers.go:71`, 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 `84`. This bypasses the intended limit, potentially returning more logs than requested.

Additionally, this issue also affects lines `97-101`, where the sum of old logs (`prevLogs` variable) and new logs (`logs` variable) may exceed the max quantity limit.

Recommendation

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

Status: Acknowledged

The client states the same reasoning as issue 6: `maxQty` is the minimum count at which the log search ends early. This variable can potentially be renamed to, eg, `targetQty` instead, but this is not a bug. The `maxQty` validation is just a best-effort sanity check.

11. Missing error return when no DAS reader is configured

Severity: Minor

In `synd-withdrawals/synd-proposer/pkg/helpers.go:175-178`, when a DAS header byte is detected in `batch[40]`, but no valid `dapReader` is found, the `getBatchPreimageData` function logs an error message.

However, the function does not return an error or terminate the execution. This may lead to silent failure where downstream logic proceeds with incomplete or incorrect assumptions about preimage availability.

Recommendation

We recommend returning an error instead of logging it only.

Status: Resolved

12. Missing error handling for `json.Marshal`

Severity: Minor

In `synd-withdrawals/synd-proposer/pkg/proposer.go:536-549`, the `ToHexForLogsTrustedInput` and `ToHexForLogsPendingAssertion` functions call `json.Marshal(hexInput)` but ignores the returned error. This suppresses potential serialization failures and may result in returning an empty or invalid JSON string silently.

Recommendation

We recommend panicking if an error occurred.

Status: Resolved

13. Error masking in the signature method

Severity: Minor

In `synd-withdrawals/synd-enclave/enclave/types.go:152-155`, if the signing operation fails, it will mask the real error as it returns the generic "`signature must be 65 bytes`" message. This is problematic because the actual error is ignored, making debugging more challenging.

```go
func (output *VerifyAppchainOutput) sign(input *TrustedInput, priv
*ecdsa.PrivateKey) (err error) {
	output.Signature, err = crypto.Sign(output.hash(input), priv)
	if len(output.Signature) != 65 {
		return fmt.Errorf("signature must be 65 bytes")
```

```
      }
      output.Signature[64] += 27
      return
}
```

Recommendation

We recommend handling the error:

```
output.Signature, err = crypto.Sign(...)
if err != nil {
    return err
}
```

and only then verify `len(output.Signature)` before adjusting the recovery byte.

Status: Resolved

14. Uninitialized `PendingAssertion` may lead to panic

Severity: Informational

In `synd-withdrawals/synd-proposer/pkg/proposer.go:116`, the proposer constructor initializes `PendingAssertion` as `nil`.

On the very first iteration of the polling loop, it compares the new trusted input's hash against the stored zero hash. Since they match, it skips the proof step entirely and leaves `PendingAssertion` unchanged.

Immediately after, it calls `SubmitAssertion` and dereferences the `nil` pointer. It will panic and halt the service:

```
submissionTimer := metrics.NewTimer()
transaction, err := p.TeeModule.SubmitAssertion(p.SettlementAuth,
*p.PendingAssertion, p.PendingSignature,
crypto.PubkeyToAddress(p.Config.PrivateKey.PublicKey))
```

When the enclave has not yet produced any assertion, the trusted input hash is zero. Since the code did not populate `PendingAssertion`, the subsequent unconditional dereference causes a runtime crash and renders the proposer inoperable until it is restarted.

Recommendation

We recommend checking the nil `PendingAssertion` and continuing accordingly:

```
if p.PendingAssertion == nil {
    p.Logger.Warn("no pending assertion yet - deferring submit")
    continue
}
```

Status: Acknowledged

The client states that the trusted input hash is read from the tee module contract and should never be zero.

15. Repeated assertion submissions

Severity: Informational

In `synd-withdrawals/synd-proposer/pkg/proposer.go:208-216`, when the proposer service starts, the constructor assigns `PendingAssertion` to `nil`:

```
// in NewProposer
PendingAssertion:    nil,
PendingSignature:    nil,
PendingTeeInputHash: common.Hash{},
```

Calling `Run` spins up the `pollingLoop` goroutine. On each tick, it executes these key steps.

First, it fetches the current trusted input from the settlement contract:

```
trustedInput, err := p.getTrustedInput(ctx)  // calls
TeeModule.TeeTrustedInput
```

Next, it tests whether this input has changed:

```
if p.PendingTeeInputHash != trustedInput.Hash() {
    appOutput, err := p.Prove(ctx, trustedInput,
settlesToArbitrumRollup)
    p.PendingAssertion    = &appOutput.PendingAssertion
    p.PendingSignature    = appOutput.Signature
    p.PendingTeeInputHash = trustedInput.Hash()
}
```

The block runs exactly once when the chain state advances. Immediately after, regardless of whether the code entered that branch, the same assertion is sent on every tick:

```
transaction, err := p.TeeModule.SubmitAssertion(
    p.SettlementAuth,
    *p.PendingAssertion,
    p.PendingSignature,
    crypto.PubkeyToAddress(p.Config.PrivateKey.PublicKey),
)
```

Since `SubmitAssertion` lies outside the change detector, the service will replay the identical assertion until the process stops or crashes.

This repeated behavior will inject the same transaction into the mempool every cycle. Gas funds will erode rapidly, and the node’s RPC endpoint may reject further requests under rate limits.

Recommendation

We recommend guarding the submission call with a check that no assertion is currently in flight.

For example, store the returned transaction hash and skip `SubmitAssertion` until that transaction is mined or a timeout expires.

Clearing `PendingAssertion` only on confirmation also prevents the same data from circulating.

Status: Acknowledged

The client states that it would not inject the tx into the mempool or erode gas funds, since it always calls `estimateGas` before actually pushing the tx.

16. Logging structures can be improved

Severity: Informational

In `synd-withdrawals/server/main.go:19`, `38`, `53`, and `63`, `log.Printf` function is used for logging instances. This approach is inconsistent with the rest of the codebase, which utilizes structured logging via `log.Info().Msg(...)`.

Recommendation

We recommend replacing the instances with `log.Info().Msg(...)`.

Status: Resolved

17. Blocks are not validated to increase monotonically

Severity: Informational

In synd-withdrawals/synd-enclave/enclave/verify.go:377, the block is updated as header = block.Header() each iteration.

However, it never asserts that header.Number increments by exactly one.

Recommendation

We recommend validating that the new block is monotonically incremented.

Status: Resolved

18. Curve is not validated to be at the point

Severity: Informational

In synd-withdrawals/synd-enclave/enclave/wavmio/stub.go:28-40, the Init method does not verify whether the point is on the curve by using g1[i].IsOnCurve().

This issue also affects lines 129-130.

Recommendation

We recommend validating the points to be on the curve to prevent buggy commencements and possibly save time.

Status: Acknowledged

The client states that the g1 points can be considered trusted data and are all on the curve.

19. The ticker is not stopped when the context is reached

Severity: Informational

In synd-withdrawals/synd-proposer/pkg/proposer.go:174-176, in the pollingLoop function, a ticker is created but never stopped:

```go
func (p *Proposer) pollingLoop(ctx context.Context) {
    ticker := time.NewTicker(p.Config.PollingInterval)
    //@audit missing defer ticker.Stop()
    for {
        select {
        case <-ctx.Done():
            log.Info().Msg("Polling loop shutting down...")
            return
        case <-ticker.C:
            // work...
        }
    }
}
```

When `ctx.Done()` fires, the function returns without halting the ticker's internal goroutine. If the loop ever restarts, each old ticker remains in memory.

Over hours or days, dozens of orphaned goroutines and timers accumulate, slowly raising memory use and risking degraded performance

Recommendation

We recommend stopping the ticker on exit:

```
func (p *Proposer) pollingLoop(ctx context.Context) {
    ticker := time.NewTicker(p.Config.PollingInterval)
+   defer ticker.Stop()
    for {
        select {
        case <-ctx.Done():
            log.Info().Msg("Polling loop shutting down...")
            return
        case <-ticker.C:
            // work...
        }
    }
}
```

Status: Resolved

20. Panic on RPC URL parsing failure

Severity: Informational

In `synd-withdrawals/synd-tee-attestation-zk-proofs/proof-submitter/src/main.rs:148`, the line `let chain_rpc_url = args.chain_rpc_url.unwrap()` calls `unwrap` on the `chain_rpc_url`, which will panic in case of error instead of gracefully handling the error.

This is against best practices in error handling, as panic should happen only on unexpected, irrecoverable errors.

Recommendation

We recommend handling that case and returning an error in case the RPC is not processed correctly.

Status: Resolved

21. Server listener settings may be overly permissive

Severity: Informational

In `synd-withdrawals/server/main.go:73`, the `http.ListenAndServe` function uses a default empty argument for the function, which makes the server listen on all network interfaces.

While this could be intended in an internal network, this may potentially open up additional attack surfaces if not all networks where the server is located are trusted.

Recommendation

We recommend binding the server to a specific trusted network interface instead of all available ones.

Status: Resolved

22. Leftover TODO comment

Severity: Informational

In `synd-withdrawals/synd-enclave/enclave/types.go:76`, there is a leftover `TODO` comment that was already resolved in the code. This decreases code maintainability.

Recommendation

We recommend removing the comment that is no longer relevant.

Status: Acknowledged

The client states that the comment is still relevant, and they will be coming back to it before launch to remove it.