Syndicate

Staking & Emissions V2
SMART CONTRACT AUDIT

08.12.2025

Made in Germany by Softstack.io

hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

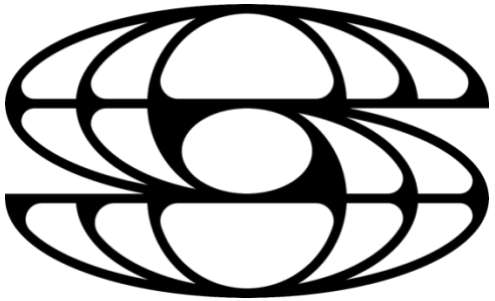
The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of SYNDICATE INC. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (20.10.2025)	Layout
0.4 (25.10.2025)	Automated & Manual Security Testing
0.5 (26.10.2025)	Verify Claims
0.9 (29.10.2025)	Summary and Recommendation
1.0 (01.11.2025)	Submission of findings
1.1 (06.11.2025)	Re-check
1.2 (08.12.2025)	Final document

2. About the Project and Company

Company address:

SYNDICATE INC.
1049 El Monte Avenue Ste C #560
Mountain View, CA 94040
USA



Website: https://syndicate.io

Twitter (X): https://x.com/syndicateio

Telegram: https://t.me/syndicateiocommunity

Farcaster: https://farcaster.xyz/syndicate

2.1 Project Overview

Syndicate is building infrastructure for appchains, blockchains tailored to the needs of specific applications and their communities. Instead of relying on general-purpose networks, Syndicate allows projects to own and govern their own chain, giving them sovereignty over performance, governance, and economics. At the heart of Syndicate's stack is the programmable on-chain sequencer, which lets developers define how transactions are ordered and processed.

This enables customization of throughput, latency, and security while ensuring communities have direct influence over how their networks operate. Developers can design their own fee models, rewards, and tokenomics, aligning incentives with their users instead of intermediaries. With the launch of Syndicate Mainnet, appchains can run in production with full community ownership.

Projects gain autonomy without losing interoperability, as Syndicate ensures that chains remain composable with the broader Web3 ecosystem. The broader vision is a community-owned internet: infrastructure where applications and their users control the stack, from consensus to economics. By combining flexibility, scalability, and open governance, Syndicate provides the foundation for Web3 projects to scale securely and sustainably while capturing the value they create.

3. Vulnerability & Risk Level

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert auditors and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to softstack to make sure we understand the size, scope, and functionality of the smart contract. ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to softstack describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Contract Files

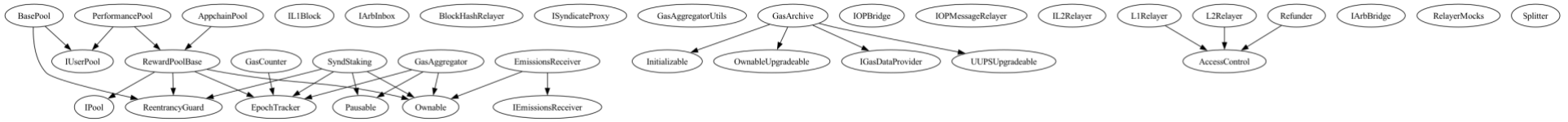
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Source: https://github.com/SyndicateProtocol/syndicate-appchains
Commit: 8911f49d540f35366d42926848a6b37fa82e6f06

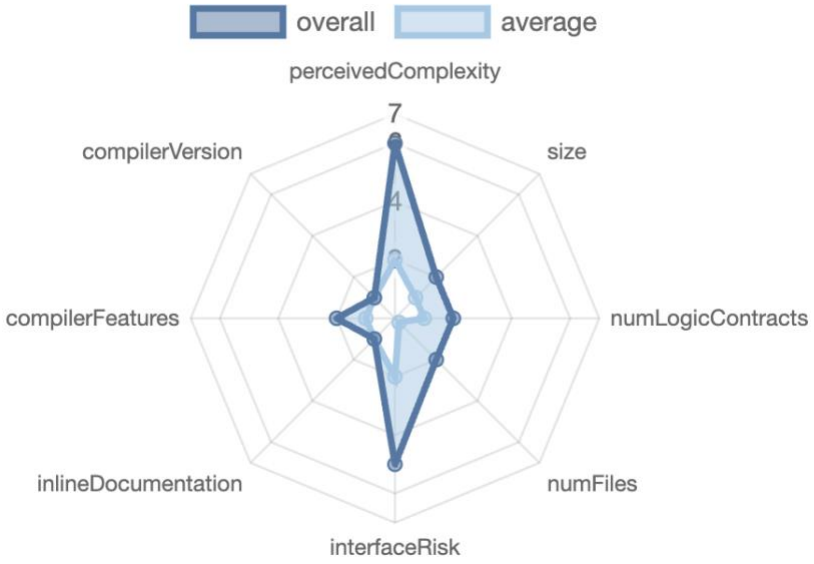
File	Fingerprint (MD5)
src/staking/interfaces/IGasDataProvider.sol	920465b93bdd50d699fcbf79b80867b5
src/staking/EpochTracker.sol	ae23e4a8411cffa7f0107e486b934aba
src/staking/GasAggregator.sol	93adf5fb153251252a68846a65c12429
src/staking/GasArchive.sol	ebfa3feae8fb273ed0504411a2a97c40
src/staking/BlockHashRelayer.sol	8cbc14bed8223c855dbd56130e9c3e43
src/staking/RewardPoolBase.sol	b0997d5f6c1ab3ab3f2e1ee5c880f059
src/staking/AppchainPool.sol	2c4d7e4d14ce6ac0a25d950539f6c278

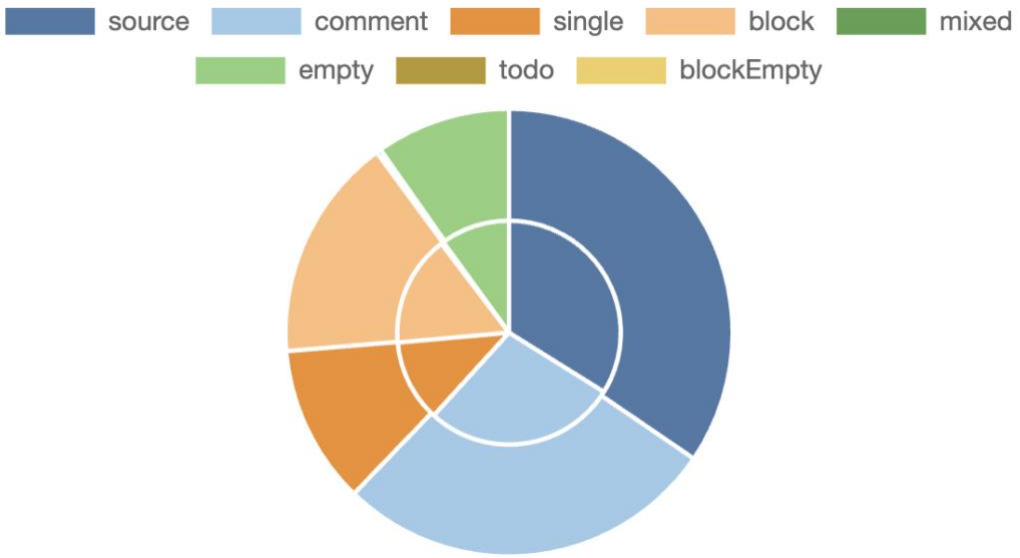
src/staking/PerformancePool.sol	990cf2c23fbf5c36c779cf28b3830b75
src/staking/Splitter.sol	4af10a7835a87f86ab34273661f3c427

5.2 Inheritance Graph



5.3 Source Lines & Risk





5.4 Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
0.8.28		yes	yes (10 asm blocks)	

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
			yes		

Exposed Functions

Public	Payable
62	5

External	Internal	Private	Pure	View
51	50	5	11	28

StateVariables

Total	Public
54	51

5.5 Dependencies / External Imports

Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/OwnableUpgradeable.sol
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/proxy/utils/Initializable.sol
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/proxy/utils/UUPSUpgradeable.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/utils/Address.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Address.sol
@openzeppelin/contracts/utils/Create2.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Create2.sol
@openzeppelin/contracts/utils/Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Pausable.sol
@openzeppelin/contracts/utils/ReentrancyGuard.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol
@openzeppelin/contracts/utils/structs/EnumerableSet.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/structs/EnumerableSet.sol
@prb/math/src/UD60x18.sol	https://github.com/PaulRBerg/prb-math/blob/main/src/UD60x18.sol

5.6 Source Unites in Scope

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
synd-contracts/src/staking/interfaces/IEpochTracker.sol		1	27	12	3	18	7
synd-contracts/src/staking/interfaces/IPool.sol		2	36	20	4	25	13
synd-contracts/src/staking/interfaces/IGasDataProvider.sol		1	38	14	3	22	11
synd-contracts/src/staking/AppchainPool.sol	1		156	156	57	77	42
synd-contracts/src/staking/BlockHashRelayer.sol	1	2	94	70	33	41	19
synd-contracts/src/staking/GasAggregator.sol	2	1	568	557	280	202	301
synd-contracts/src/staking/RewardPoolBase.sol	1		297	292	144	110	141
synd-contracts/src/staking/GasArchive.sol	1		462	429	234	131	246
synd-contracts/src/staking/PerformancePool.sol	1		141	137	52	68	51
synd-contracts/src/staking/Splitter.sol	1		105	105	40	50	30
Totals	**8**	**7**	**1924**	**1792**	**850**	**744**	**861**

6. Scope of Work

The audit will validate the security and correctness of the multi-chain gas tracking, cross-chain proof verification, reward distribution, and staking logic across Base (settlement), Arbitrum appchains (sequencing), and L3 (staking). The following critical areas must be addressed:

1. **Gas Tracking & Proof Verification Integrity**: The audit must verify that GasAggregator correctly collects gas usage from appchains using CREATE2 address derivation, enforces epoch sequentiality, and prevents manipulation. GasArchive must trustlessly validate Merkle Patricia storage proofs from Arbitrum Outbox contracts, with proper defense against known vulnerabilities (MPT silent fallthrough, RLP injection, forged storage roots) and replay attacks via block hash verification through BlockHashRelayer.

2. **Staking and Pro-Rata Accounting**: The staking system must accurately track user balances across global, per-appchain, and per-epoch dimensions, ensuring partial-epoch participants receive proportional rewards and that restaking, withdrawals, and delayed finalization cannot be exploited for excess rewards or double-claiming.

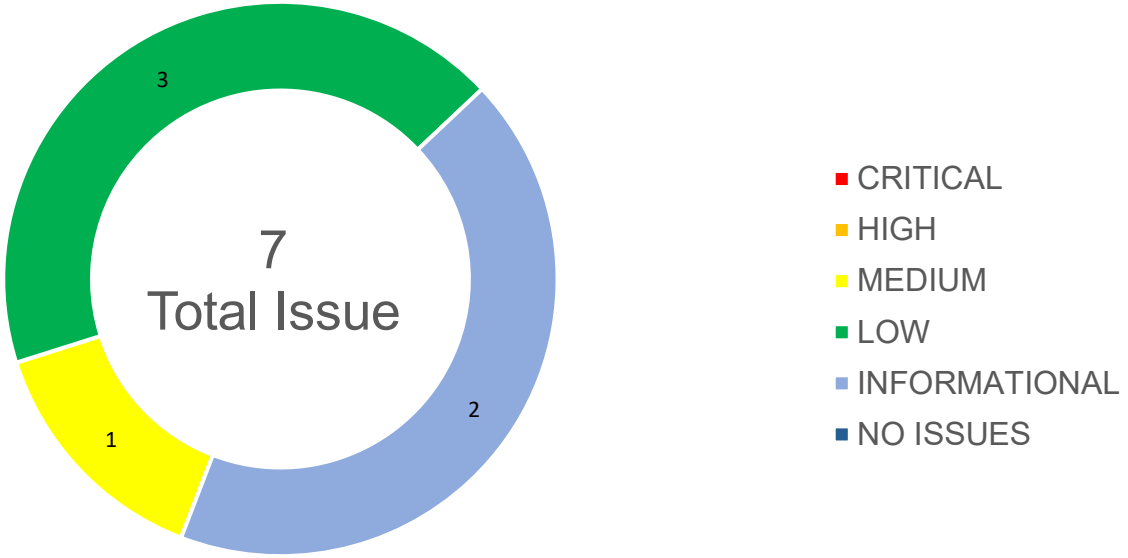
3. **Cross-Chain Messaging & Bridging Security:** All L1 → Base → L3 relay operations using Arbitrum retryable tickets must be secure, atomic, and protected against replay, spoofing, or mis-routing of funds. The Refunder contract must guarantee recovery of stuck SYND tokens from failed bridge transactions without loss.

4. **Reward Distribution & Diminishing Returns Algorithm**: The RewardPoolBase contracts must implement the diminishing returns formula (ln(1 + decayFactor * dominance)) correctly using PRB Math, preventing centralization through gas/stake manipulation. Splitter must distribute rewards accurately (30/40/30 split), AppchainPool must enforce 1-year linear vesting, and PerformancePool must prevent double-claims while allowing immediate user withdrawals.

5. **Access Control & Emergency Response**: Critical roles (Owner, Admin, blockHashSender) must be strictly enforced to prevent unauthorized minting, parameter changes, or upgrades. The system's pause mechanisms (Pausable), reentrancy guards (ReentrancyGuard), and state-then-interact patterns must effectively mitigate abuse, reentrancy, or governance capture risks.

The primary objective of this audit is to ensure that gas tracking, staking, and cross-chain distribution logic are secure, mathematically sound, and resistant to adversarial manipulation, providing a robust foundation for long-term protocol operation.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Admin Can Permanently Stall Epoch Advancement	MEDIUM	FIXED
6.2.2	Deterministic getNextChainId() enables frontrunning to brick auto-increment namespace	LOW	FIXED

6.2.3	ChainSubmitted event emitted after epoch increment	LOW	ACKNOWLEDGED
6.2.4	PerformancePool uses global stake for per-appchain rewards	LOW	FIXED
6.2.5	Unused imports	INFORMATIONAL	FIXED
6.2.6	Duplicate import of Initializable	INFORMATIONAL	FIXED
6.2.7	Unused import in Splitter	INFORMATIONAL	FIXED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, softstack‘s experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, softstack‘s experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, softstack‘s experts found **One Medium issues** in the code of the smart contract.

6.2.1 Admin Can Permanently Stall Epoch Advancement

Severity: MEDIUM
Status: FIXED
File(s) affected: GasArchive.sol
Update: https://github.com/SyndicateProtocol/syndicate-appchains/commit/a533ed922819e0a1345184128db1e7f6e9fa8721

Attack / Description	The removeSequencingChain() function contains a critical logic flaw that permanently prevents epoch advancement when the last sequencing chain is removed. The bug stems from checking the chain count after removal instead of before. Root Cause: The condition if (seqChainCount > 0 && epochRemainingChains == 0) evaluates to false when removing the last chain because seqChainCount is read from seqChains.length() AFTER the chain has been removed from the EnumerableSet. Failure Sequence: 1. seqChains.remove(chainID) → Set now empty 2. epochRemainingChains-- → Decrements to 0 3. seqChainCount = seqChains.length() → Returns 0 (empty set) 4. if (0 > 0 && 0 == 0) → FALSE 5. Epoch never increments → Permanently stuck

Proof of Concept

```
// SPDX-License-Identifier: UNLICENSED

pragma solidity 0.8.28;


import {Test, console2} from "forge-std/Test.sol";

import {GasArchive} from "src/staking/GasArchive.sol";

import {ERC1967Proxy} from

"@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";


/**


 * @notice Demonstrates permanent epoch stall when removing last

sequencing chain

 */

contract Minimal_PoC is Test {

    GasArchive public gasArchive;

    address public admin;
```

```
    uint256 constant START_TIMESTAMP = 1754089200;


    function setUp() public {
        admin = makeAddr("admin");
        vm.deal(admin, 100 ether);
    }


    function test__PermanentEpochStall() public {
        // Deploy GasArchive
        vm.warp(START_TIMESTAMP + 1 days);
        address blockHashSender = makeAddr("blockHashSender");
        uint256 settlementChainID = 42161;


        GasArchive implementation = new GasArchive(blockHashSender,
settlementChainID);
        bytes memory initData = abi.encodeCall(GasArchive.initialize,
(1));
```

```
        gasArchive = GasArchive(address(new
ERC1967Proxy(address(implementation), initData)));
        gasArchive.transferOwnership(admin);


        // Add 3 sequencing chains as admin
        vm.startPrank(admin);


        address aggr1 = makeAddr("aggregator1");
        address aggr2 = makeAddr("aggregator2");
        address aggr3 = makeAddr("aggregator3");
        address outbox1 = makeAddr("outbox1");
        address outbox2 = makeAddr("outbox2");
        address outbox3 = makeAddr("outbox3");


        gasArchive.addSequencingChain(1, aggr1, outbox1, false);
        gasArchive.addSequencingChain(2, aggr2, outbox2, false);
        gasArchive.addSequencingChain(3, aggr3, outbox3, false);
```

```
        uint256 initialEpoch = gasArchive.epoch();
        uint256 initialRemainingChains =
gasArchive.epochRemainingChains();
        uint256 initialSeqChainCount =
gasArchive.sequencingChainCount();

        assertEq(initialEpoch, 1, "Should start at epoch 1");
        assertEq(initialRemainingChains, 3, "Should have 3 remaining
chains");
        assertEq(initialSeqChainCount, 3, "Should have 3 sequencing
chains");

        // Remove all chains without submitting data
        gasArchive.removeSequencingChain(1);
        assertEq(gasArchive.epochRemainingChains(), 2);
        assertEq(gasArchive.sequencingChainCount(), 2);
```

```
gasArchive.removeSequencingChain(2);
assertEq(gasArchive.epochRemainingChains(), 1);
assertEq(gasArchive.sequencingChainCount(), 1);

// BUG TRIGGER: Remove last chain
gasArchive.removeSequencingChain(3);

uint256 finalRemaining = gasArchive.epochRemainingChains();
uint256 finalSeqCount = gasArchive.sequencingChainCount();
uint256 finalEpoch = gasArchive.epoch();

vm.stopPrank();

// VALIDATE BUG: Epoch should be 2 but stuck at 1
assertEq(finalRemaining, 0, "epochRemainingChains = 0");
assertEq(finalSeqCount, 0, "sequencingChainCount = 0");
```

```
        assertEq(finalEpoch, 1, "BUG: Epoch stuck at 1");


        console2.log("===  BUG VALIDATED ===");
        console2.log("Expected epoch: 2");
        console2.log("Actual epoch:", finalEpoch);
        console2.log("Bug confirmed: Epoch permanently stuck");
    }
}
```

Code	Line 440-459 (GasArchive.sol)

```
    function removeSequencingChain(uint256 chainID) external onlyOwner {
        require(seqChains.remove(chainID), SequencingChainDoesNotExist());
        seqChainGasAggregator[chainID] = address(0);
        if (chainID != settlementChainID) {
            seqChainOutbox[chainID] = address(0);
            seqChainSettlesToBase[chainID] = false;
        }
        if (!epochChainDataSubmitted[epoch][chainID]) {
            // clear the verified data hash in case it is set
            epochVerifiedDataHash[epoch][chainID] = bytes32(0);
            epochRemainingChains--;
            uint256 seqChainCount = seqChains.length();
```

```
            if (seqChainCount > 0 && epochRemainingChains == 0) {
                emit EpochCompleted(epoch);
                epoch++;
                epochRemainingChains = seqChainCount;
            }
        }
        emit ChainRemoved(epoch, chainID);
    }
```

Result/Recommendation

Remove Redundant Check

```
function removeSequencingChain(uint256 chainID) external onlyOwner {
    require(seqChains.remove(chainID), SequencingChainDoesNotExist());
    seqChainGasAggregator[chainID] = address(0);
    if (chainID != settlementChainID) {
        seqChainOutbox[chainID] = address(0);
```

```
            seqChainSettlesToBase[chainID] = false;
        }
        if (!epochChainDataSubmitted[epoch][chainID]) {
            epochVerifiedDataHash[epoch][chainID] = bytes32(0);
            epochRemainingChains--;


            // FIX: Remove seqChainCount > 0 check
            // Allow epoch to complete even when last chain removed
            if (epochRemainingChains == 0) {
                emit EpochCompleted(epoch);
                epoch++;
                uint256 seqChainCount = seqChains.length();
                epochRemainingChains = seqChainCount;  // May be 0 - valid
state
            }
```


hello@softstack.io
www.softstack.io
softstack GmbH
Schiffbrückstraße 8
24937 Flensburg
CRN: HRB 12635 FL
VAT: DE317625984

```
    }

    emit ChainRemoved(epoch, chainID);

}
```

Check Before Removal

```
function removeSequencingChain(uint256 chainID) external onlyOwner {
    uint256 seqChainCountBefore = seqChains.length();  // Read BEFORE removal


    require(seqChains.remove(chainID), SequencingChainDoesNotExist());
    seqChainGasAggregator[chainID] = address(0);
    if (chainID != settlementChainID) {
        seqChainOutbox[chainID] = address(0);
        seqChainSettlesToBase[chainID] = false;
    }
    if (!epochChainDataSubmitted[epoch][chainID]) {
```

```
            epochVerifiedDataHash[epoch][chainID] = bytes32(0);
            epochRemainingChains--;


            // Use pre-removal count
            if (seqChainCountBefore > 0 && epochRemainingChains == 0) {
                emit EpochCompleted(epoch);
                epoch++;
                uint256 seqChainCount = seqChains.length();
                epochRemainingChains = seqChainCount;
            }
        }
        emit ChainRemoved(epoch, chainID);
    }
```


hello@softstack.io
www.softstack.io
softstack GmbH
Schiffbrückstraße 8
24937 Flensburg
CRN: HRB 12635 FL
VAT: DE317625984

LOW ISSUES

During the audit, softstack‘s experts found **Three Low issues** in the code of the smart contract

6.2.2 Deterministic getNextChainId() enables frontrunning to brick auto-increment namespace

Severity: LOW
Status: ACKNOWLEDGED
File(s) affected: SyndicateFactory.sol
Update: We are rolling out UUPS upgrades to our contracts in the near future and this is resolved in this change as we change the underlying logic.

Attack / Description	The SyndicateFactory.createSyndicateSequencingChain() function supports both manual chain ID assignment and automatic ID allocation via appchainId = 0. However, the automatic ID is computed using getNextChainId() which is fully deterministic and publicly queryable. Since manual registrations (non-zero appchainId) do not increment nextAutoChainId, an attacker can: 1. Predict the next auto-ID by calling getNextChainId() 2. Frontrun the auto-deployment by calling createSyndicateSequencingChain(predictedId, ...) manually 3. Squat on the predicted ID, causing the auto-deployment to revert with ChainIdAlreadyExists 4. The nextAutoChainId counter never advances, permanently blocking all future auto-deployments This creates a permanent DoS on the factory's auto-increment feature with a single permissionless transaction. **Proof of Concept** `function test_bug_manualChainIdSnipingBricksAutoIncrement() public {`

```
// Setup: Factory starts with nextAutoChainId = 5

uint256 predictedId = factory.getNextChainId();

assertEq(predictedId, 5, "Next auto ID should be 5");

// EXPLOIT: Attacker frontruns by manually deploying to ID 5

vm.startPrank(attacker);

address attackerChain = factory.createSyndicateSequencingChain(

predictedId,  // Manual ID = 5 (the predicted auto-ID)

address(syndToken),

"Attacker Chain",

... // other params

);

assertEq(attackerChain != address(0), true, "Attacker chain deployed");

vm.stopPrank();

// Legitimate user tries to deploy with auto-ID (appchainId = 0)

vm.startPrank(legitimateUser);

// This call will compute appchainId = getNextChainId() = 5 internally

vm.expectRevert(abi.encodeWithSignature("ChainIdAlreadyExists(uint256)", 5));
```

```
        factory.createSyndicateSequencingChain(
        0,  // Auto-ID mode
        address(syndToken),
        "Legitimate Chain",
        ...
        );
        vm.stopPrank();
        // Verify the counter is STILL 5 (never advanced)
        uint256 nextIdAfterRevert = factory.getNextChainId();
        assertEq(nextIdAfterRevert, 5, "Counter stuck at 5");
        // ALL future auto-deployments will fail with same error
        vm.startPrank(anotherUser);
        vm.expectRevert(abi.encodeWithSignature("ChainIdAlreadyExists(uint256)", 5));
        factory.createSyndicateSequencingChain(0, ...);  // PERMANENTLY BLOCKED
        vm.stopPrank();
}
```

Code	Line 127-135 (src/factory/SyndicateFactory.sol)
	
	`function getNextChainId() public view returns (uint256) {`
	`  return nextAutoChainId;  //  Deterministic, publicly readable`
	`}`
	
	Line 66-105 (src/factory/SyndicateFactory.sol)
	
	`function createSyndicateSequencingChain(`
	`  uint256 appchainId,  // Can be 0 for auto, or manual ID`
	`  ...`
	`) external returns (address) {`
	`  if (appchainId == 0) {`
	`  appchainId = nextAutoChainId;`
	`  nextAutoChainId++;  // Only increments for auto-mode`
	`  }`
	`  require(!chainIdExists[appchainId], "ChainIdAlreadyExists");`
	`  // Deploy chain...`
	`  //  Manual deployments don't increment nextAutoChainId`

	`// If manual deployment squats on nextAutoChainId value, next auto-call reverts`
	
	`}`

Result/Recommendation	Always Increment Counter
	
	`function createSyndicateSequencingChain(`
	`uint256 appchainId,`
	`address admin,`
	`IRequirementModule permissionModule,`
	`bytes32 salt`
	`) external whenNotPaused returns (address sequencingChain, uint256 actualChainId) {`
	`// ... validation ...`
	
	`// Determine actual chain ID`
	`actualChainId = appchainId == 0 ? getNextChainId() : appchainId;`
	
	`// FIX: Increment BEFORE validation (only for auto-mode)`
	`if (appchainId == 0) {`

```
        nextAutoChainId++;
    }


    // Validate chain ID is not already used
    if (appchainContracts[actualChainId] != address(0)) {
        revert ChainIdAlreadyExists();
    }


    // ... deploy contract ...
}
```

6.2.3 ChainSubmitted event emitted after epoch increment
Severity: LOW
Status: FIXED
File(s) affected: GasArchive.sol
Update: https://github.com/SyndicateProtocol/syndicate-appchains/commit/ad7f3e7e7cdbdf47fe63c6474ae75e08397f6e06

Attack / Description	

	When the last chain submits, the contract emits EpochCompleted(epoch), then increments epoch, and then emits ChainSubmitted(epoch, seqChainID). This causes ChainSubmitted to reference the next epoch instead of the one just completed.

Code	Line 277-285 (GasArchive.sol)
	
	`epochChainDataSubmitted[epoch][seqChainID] = true;`
	`epochRemainingChains--;`
	`if (epochRemainingChains == 0) {`
	`  emit EpochCompleted(epoch);`
	`  epoch++;`
	`  epochRemainingChains = seqChains.length();`
	`}`
	`emit ChainSubmitted(epoch, seqChainID);`

Result/Recommendation	Emit ChainSubmitted before incrementing epoch, or store uint256 completedEpoch = epoch; and use it in the event after incrementing.

6.2.4 PerformancePool uses global stake for per-appchain rewards

Severity: LOW
Status: FIXED
File(s) affected: PerformancePool.sol

Attack / Description	getClaimableAmount calculates a user's share using their global stake (getUserStakeShare) divided by the appchain's total stake (getAppchainStake). This allows users with zero stake on a specific appchain to claim rewards from that appchain as long as they have any global stake. This might misallocate funds, if not intended by business logic.

Code	Line 127-139 (PerformancePool.sol)
	
	`function getClaimableAmount(uint256 epochIndex, address user, uint256 appchainId) public view returns (uint256) {`
	
	`uint256 appchainTotal = getAppchainTotalReward(epochIndex, appchainId);`
	
	`if (appchainTotal == 0) return 0;`
	
	`uint256 userStaked = ISyndStaking(address(stakingContract)).getUserStakeShare(epochIndex, user); // global`
	
	`if (userStaked == 0) return 0;`
	
	`uint256 appchainStaked = ISyndStaking(address(stakingContract)).getAppchainStake(epochIndex, appchainId);`
	
	`if (appchainStaked == 0) return 0;`
	
	`uint256 userShare = (appchainTotal * userStaked) / appchainStaked;`
	
	`...`
	
	`}`

Result/Recommendation	Use per-appchain user stake (e.g., getUserAppchainStake(epochIndex, user, appchainId)) in the numerator to ensure users can only claim from appchains where they have stake.

INFORMATIONAL ISSUES

During the audit, softstack‘s experts found **Three Informational issues** in the code of the smart contract

6.2.5 Unused imports
Severity: INFORMATIONAL
Status: FIXED
File(s) affected: GasAggregator.sol
Update: https://github.com/SyndicateProtocol/syndicate-appchains/commit/ad7f3e7e7cdbdf47fe63c6474ae75e08397f6e06

Attack / Description	ISyndicateSequencingChain and GasCounter are imported but not used.

Code	Line 7-10 (GasAggregator.sol)
	
	`import {ISyndicateSequencingChain} from "../interfaces/ISyndicateSequencingChain.sol";`
	`import {GasCounter} from "./GasCounter.sol";`

Result/Recommendation	Remove unused imports to reduce bytecode size and improve readability.

6.2.6 Duplicate import of Initializable

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: GasArchive.sol
Commit: https://github.com/SyndicateProtocol/syndicate-appchains/commit/ad7f3e7e7cdbdf47fe63c6474ae75e08397f6e06

Attack / Description	Initializable is imported twice from the same path, increasing bytecode size and reducing readability..

Code	Line 4.10 (GasArchive.sol)
	
	`import {Initializable} from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";`
	`...`
	`import {Initializable} from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";`

Result/Recommendation	Remove the duplicate import.

6.2.7 Unused import in Splitter

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: Splitter.sol
Commit: https://github.com/SyndicateProtocol/syndicate-appchains/commit/ad7f3e7e7cdbdf47fe63c6474ae75e08397f6e06

Attack / Description	Address is imported but not used.
Code	Line 4 (Splitter.sol) `import {Address} from "@openzeppelin/contracts/utils/Address.sol";`

Result/Recommendation	Remove the unused import.

6.3 Verify Claims

6.3.1 **Gas Tracking & Proof Verification Integrity**

The audit must verify that GasAggregator correctly collects gas usage from appchains using CREATE2 address derivation, enforces epoch sequentiality, and prevents manipulation. GasArchive must trustlessly validate Merkle Patricia storage proofs from Arbitrum Outbox contracts, with proper defense against known vulnerabilities (MPT silent fallthrough, RLP injection, forged storage roots) and replay attacks via block hash verification through BlockHashRelayer.

Status: tested and verified ☑

6.3.2 **Staking and Pro-Rata Accounting**

The staking system must accurately track user balances across global, per-appchain, and per-epoch dimensions, ensuring partial-epoch participants receive proportional rewards and that restaking, withdrawals, and delayed finalization cannot be exploited for excess rewards or double-claiming.

Status: tested and verified ☑

6.3.3 **Cross-Chain Messaging & Bridging Security**

All L1 → Base → L3 relay operations using Arbitrum retryable tickets must be secure, atomic, and protected against replay, spoofing, or mis-routing of funds. The Refunder contract must guarantee recovery of stuck SYND tokens from failed bridge transactions without loss.

Status: tested and verified ☑

6.3.4 **Reward Distribution & Diminishing Returns Algorithm**

The RewardPoolBase contracts must implement the diminishing returns formula (ln(1 + decayFactor * dominance)) correctly using PRB Math, preventing centralization through gas/stake manipulation. Splitter must distribute rewards accurately (30/40/30 split),

AppchainPool must enforce 1-year linear vesting, and PerformancePool must prevent double-claims while allowing immediate user withdrawals.
Status: tested and verified ☑

6.3.5 **Access Control & Emergency Response**
Critical roles (Owner, Admin, blockHashSender) must be strictly enforced to prevent unauthorized minting, parameter changes, or upgrades. The system's pause mechanisms (Pausable), reentrancy guards (ReentrancyGuard), and state-then-interact patterns must effectively mitigate abuse, reentrancy, or governance capture risks.
Status: tested and verified ☑

7. Executive Summary

Two independent softstack experts performed an unbiased and isolated audit of the smart contract provided by the Syndicate team. The main objective of the audit was to verify the security and functionality claims of the smart contract. The audit process involved a thorough manual code review and automated security testing.

Overall, the audit identified a total of 7 issues, classified as follows:
- Zero critical issue was found.
- Zero high severity issue was found.
- 1 medium severity issues were found.
- 3 low issues were found.
- 3 informational findings.

The audit report provides detailed descriptions of each identified issue, including severity levels, proof of concepts and recommendations for mitigation. It also includes code snippets, where applicable, to demonstrate the issues and suggest possible fixes. We recommend the Syndicate team to review the suggestions.

Update (29.11.2025): All previously identified findings have been successfully mitigated. The codebase was re-checked twice by softstack auditors to confirm the fixes and ensure no regressions were introduced.

8. About the Auditor

Established in 2017 under the name Chainsulting, and rebranded as softstack GmbH in 2023, softstack has been a trusted name in Web3 Security space. Within the rapidly growing Web3 industry, softstack provides a comprehensive range of offerings that include software development, cybersecurity, and consulting services. Softstack's competency extends across the security landscape of prominent blockchains like Solana, Tezos, TON, Ethereum and Polygon. The company is widely recognized for conducting thorough code audits aimed at mitigating risk and promoting transparency.

The firm's proficiency lies particularly in assessing and fortifying smart contracts of leading DeFi projects, a testament to their commitment to maintaining the integrity of these innovative financial platforms. To date, softstack plays a crucial role in safeguarding over $100 billion worth of user funds in various DeFi protocols.

Underpinned by a team of industry veterans possessing robust technical knowledge in the Web3 domain, softstack offers industryleading smart contract audit services. Committed to evolving with their clients' ever-changing business needs, softstack's approach is as dynamic and innovative as the industry it serves.

Check our website for further information: https://softstack.io