Syndicate

Token Staking & Emissions

SMART CONTRACT AUDIT

16.09.2025

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

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

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 (03.09.2025)	Layout
0.4 (05.09.2025)	Automated Security Testing Manual Security Testing
0.5 (09.09.2025)	Verify Claims
0.9 (10.09.2025)	Summary and Recommendation
1.0 (11.09.2025)	Submission of findings
1.1 (15.09.2025)	Re-check
1.2 (16.09.2025)	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

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

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

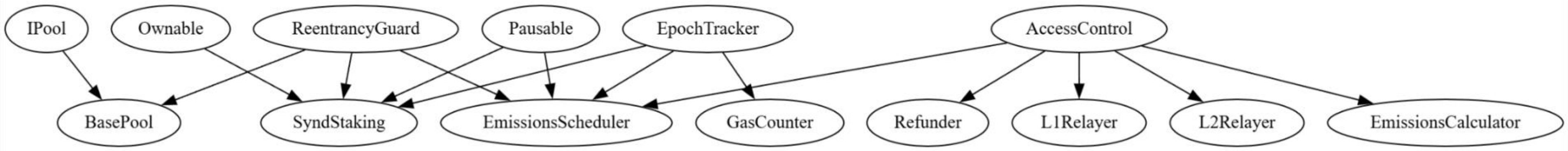
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

Source: https://github.com/SyndicateProtocol/syndicate-appchains
Commit: eae40c10d17b2f8ac705eaa308e4cb7917ca7cd0

File	Fingerprint (MD5)
./staking/BasePool.sol	8da8ac8d0d3a448b12982da85d22856a
./staking/EpochTracker.sol	5d2bba7f301ded57f3e60d62be0ee4f0
./staking/GasCounter.sol	ce6985b7291aa665d85f9663c550cd62
./staking/SyndStaking.sol	3b7db5d33b81c53bf85af8ee89c6b102
./staking/Refunder.sol	cf71479270a877e1bd1fa89f69cd3f08
./staking/L1Relayer.sol	617ceabab3b3da1ea929f63872942e04
./staking/L2Relayer.sol	ed89abea49044cb401671a7a4db9bbd6
./token/emissions/EmissionsCalculator.sol	5c7fc25e1fe2f0a0154119b24cbfc1a2
./token/emissions/EmissionsScheduler.sol	f2cee44fe73b1eeb1dcaec36665430ba

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

5.2 Inheritance Graph

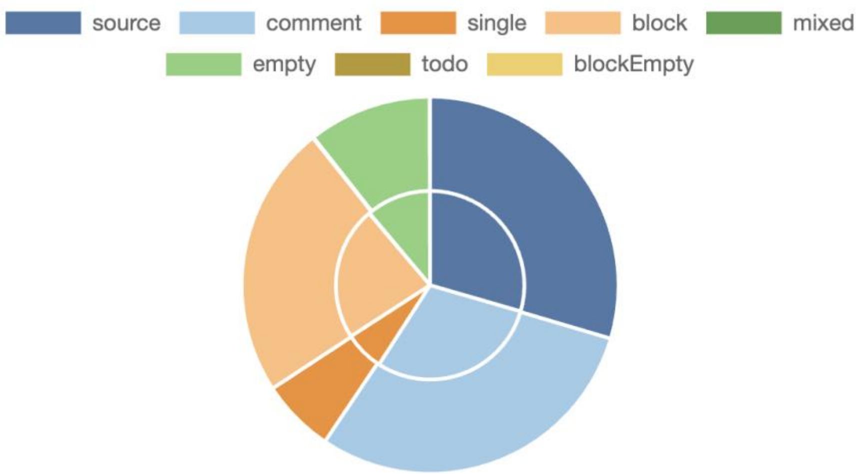


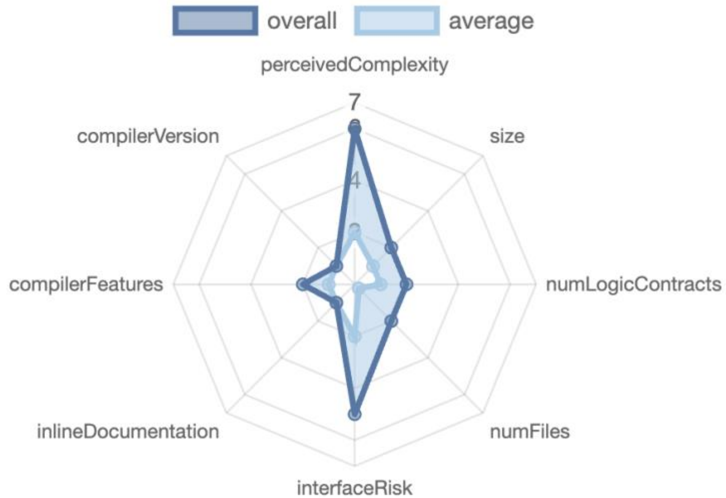
hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

5.3 Source Lines & Risk





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

5.4 Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
0.8.28		yes	No	

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
	Yes	No	yes		No

Exposed Functions

Public	Payable
28	4

External	Internal	Private	Pure	View
24	18	10	4	20

StateVariables

Total	Public
60	45

5.5 Dependencies / External Imports

Dependency / Import Path	Source
@openzeppelin/contracts/utils/Address.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.4.0/contracts/utils/Address.sol
@openzeppelin/contracts/utils/ReentrancyGuard.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.4.0/contracts/utils/ReentrancyGuard.sol
@openzeppelin/contracts/utils/Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.4.0/contracts/utils/Pausable.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.4.0/contracts/access/Ownable.sol
@openzeppelin/contracts/access/AccessControl.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.4.0/contracts/access/AccessControl.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.4.0/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v5.4.0/contracts/token/ERC20/utils/SafeERC20.sol

hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

5.6 Source Unites in Scope

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines
synd-contracts/src/staking/BasePool.sol	1		155	151	62	67
synd-contracts/src/staking/EpochTracker.sol	1		55	55	21	29
synd-contracts/src/staking/L2Relayer.sol	1	1	146	120	49	66
synd-contracts/src/staking/L1Relayer.sol	1	2	149	123	48	70
synd-contracts/src/staking/GasCounter.sol	1		100	100	41	40
synd-contracts/src/staking/SyndStaking.sol	1		693	684	321	271
synd-contracts/src/staking/Refunder.sol	1		58	58	21	29
synd-contracts/src/token/emissions/EmissionsCalculator.sol	1	1	342	309	117	145
synd-contracts/src/token/emissions/EmissionsScheduler.sol	1	1	223	207	79	103
Totals	**9**	**5**	**1921**	**1807**	**759**	**820**

Legend:

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments

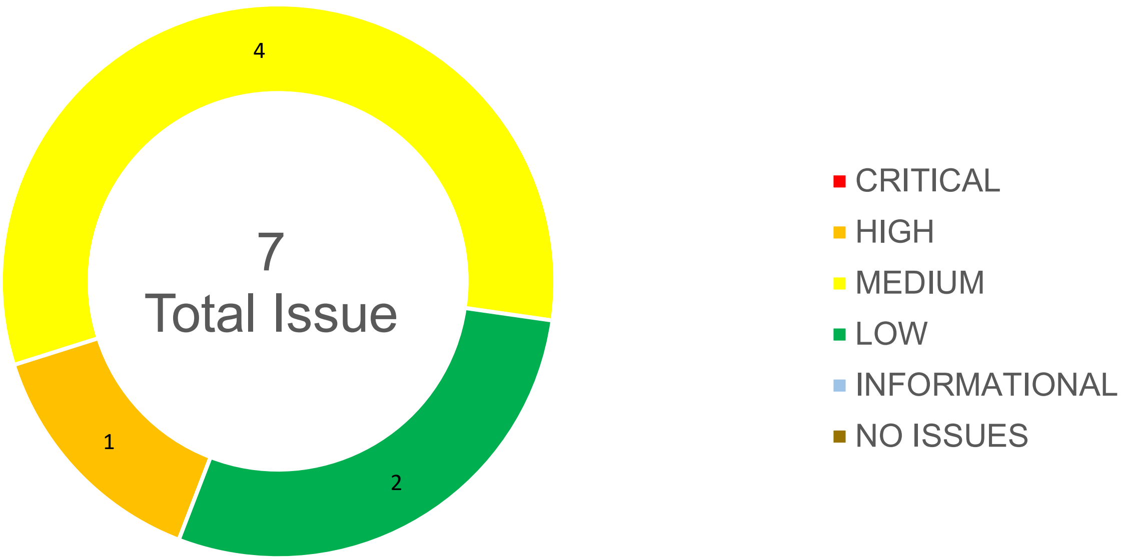
6. Scope of Work

The Syndicate team has developed a multi-chain staking and emissions system spanning Ethereum Mainnet (L1), Optimism (L2), Arbitrum L3 (Commons Chain), and individual appchains. The audit will focus on validating the correctness, security, and robustness of the emission schedule, staking logic, reward distribution, and cross-chain operations. The following critical areas and assumptions must be addressed during the audit:

1. **Emission Schedule Integrity**
   The audit must verify that the geometric decay formula for distributing 80M SYND over 48 epochs is implemented correctly, prevents double-minting, enforces sequential epoch processing, and cannot be manipulated through timing attacks or governance misconfiguration.
2. **Staking and Pro-Rata Accounting**
   The staking system must accurately track user balances across global, per-appchain, and per-epoch dimensions, ensuring that partial-epoch participants receive correct rewards and that restaking, withdrawals, and delayed finalization cannot be exploited for excess rewards.
3. **Cross-Chain Messaging & Bridging Security**
   All L1 → L2 → L3 relay operations must be secure, atomic, and protected against replay, spoofing, or mis-routing of funds. Retryable ticket logic and fund recovery (via Refunder) must guarantee liveness and correctness under failed bridge scenarios.
4. **Reward Distribution & Gas-Based Incentives**
   The BasePool and GasCounter contracts must ensure that sequencer and staking rewards are distributed proportionally to actual stake and gas consumption, without allowing manipulation of gas tracking, double-claims, or misallocation across appchains.
5. **Access Control & Emergency Response**
   Critical roles (Admin, Decay Manager, Emissions, Pauser) must be strictly enforced to prevent unauthorized minting or parameter changes. The system's pause mechanisms, reentrancy guards, and state-then-interact patterns must effectively mitigate abuse, reentrancy, or governance capture risks.

The primary objective of this audit is to ensure that emissions, 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	No Destination Zero Check in claimAllRewards	HIGH	FIXED
6.2.2	Unnecessary Payable Modifier in stageStakeTransfer	MEDIUM	FIXED
6.2.3	Function Selector Tight Coupling in L1Relayer Cross-Chain Messaging	MEDIUM	FIXED
6.2.4	Epoch Underflow Before Start Time in EpochTracker	MEDIUM	ACKNOWLEDGED
6.2.5	Double-spend/Incorrect Fee Accounting in L2Relayer	MEDIUM	ACKNOWLEDGED
6.2.6	Division Precision Loss in Reward Calculation	LOW	FIXED
6.2.7	Gas Price Manipulation in GasCounter	LOW	ACKNOWLEDGED

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 **one High issue** in the code of the smart contract.

6.2.1 No Destination Zero Check in claimAllRewards

Severity: HIGH
Status: FIXED
File(s) affected: synd-contracts/src/staking/SyndStaking.sol
Update: https://github.com/SyndicateProtocol/syndicate-appchains/commit/ec648b289d5d7ca68816205985854c9709e021ab

Attack / Description	The `SyndStaking.claimAllRewards` iterates over provided pools and calls `IPool(pool).claimFor(epochIndex, user, destination)` with the caller-supplied `destination`, but performs no zero-address check. See [`claimAllRewards`] By contrast, withdrawals validate `destination != address(0)`: - Single withdraw: [`withdraw`] - Bulk withdraw: [`withdrawBulk`] On the pool side, the implementation unconditionally transfers ETH to the provided `destination` with no zero-address validation in [`BasePool._claim`], specifically the send at [L99] `Address.sendValue(payable(destination), claimAmount);`.

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

This means specifying `destination = address(0)` causes ETH to be transferred to the zero address, permanently burning funds.

Additional exposure: Users can also call pools directly and burn rewards via `BasePool.claim` which similarly lacks a zero-address check. See [`claim`] and forwarding path [`claimFor`]

Proof of Concept

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

pragma solidity 0.8.28;

import {Test} from "forge-std/Test.sol";
import {SyndStaking} from "../../src/staking/SyndStaking.sol";
import {BasePool} from "../../src/staking/BasePool.sol";

contract ClaimAllRewardsZeroDestinationTest is Test {

    SyndStaking staking;

    BasePool pool;

    address user = address(0xBEEF);

    function setUp() public {

        staking = new SyndStaking(address(this));

        pool = new BasePool(address(staking));

        vm.deal(address(this), 100 ether);

        vm.deal(user, 10 ether);
```

```
    }

    function test_claimAllRewards_toZeroDestination_burnsFunds() public {

        uint256 epoch1Start = staking.getEpochStart(1);

        vm.warp(epoch1Start + 1);

        vm.prank(user);

        staking.stakeSynd{value: 1 ether}(1);

        pool.deposit{value: 1 ether}(1);

        uint256 epoch1End = staking.getEpochEnd(1);

        vm.warp(epoch1End + 1);

        uint256 claimable = pool.getClaimableAmount(1, user);

        assertGt(claimable, 0);

        SyndStaking.ClaimRequest[] memory claims = new
SyndStaking.ClaimRequest[](1);

        claims[0] = SyndStaking.ClaimRequest({epochIndex: 1, poolAddress:
address(pool)});



        uint256 zeroBefore = address(0).balance;

        uint256 poolBefore = address(pool).balance;
```

	`` uint256 userBefore = user.balance; vm.prank(user); staking.claimAllRewards(claims, address(0)); uint256 zeroAfter = address(0).balance; uint256 poolAfter = address(pool).balance; uint256 userAfter = user.balance; assertEq(zeroAfter - zeroBefore, claimable); assertEq(poolBefore - poolAfter, claimable); assertEq(userAfter, userBefore); } } ```
Code	Line 516-523 (SyndStaking.sol): ``` function claimAllRewards(ClaimRequest[] calldata claims, address destination) external nonReentrant { if (claims.length == 0) { revert NoClaimsProvided(); } ```

hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

	`` for (uint256 i = 0; i < claims.length; i++) { IPool(claims[i].poolAddress).claimFor(claims[i].epochIndex, msg.sender, destination); } ```
Result/Recommendation	Add a zero-address validation to `claimAllRewards`: function claimAllRewards(ClaimRequest[] calldata claims, address destination) external nonReentrant { if (claims.length == 0) revert NoClaimsProvided(); if (destination == address(0)) revert InvalidDestination(); for (uint256 i = 0; i < claims.length; i++) { IPool(claims[i].poolAddress).claimFor(claims[i].epochIndex, msg.sender, destination); } } Also harden the pool contract by validating `destination` in `BasePool._claim` (or in both `claim` and `claimFor`) to close the burn vector even if future integrations omit validation.

MEDIUM ISSUES

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

6.2.2 Unnecessary Payable Modifier in stageStakeTransfer

Severity: MEDIUM
Status: FIXED
File(s) affected: synd-contracts/src/staking/SyndStaking.sol
Update: https://github.com/SyndicateProtocol/syndicate-appchains/commit/86ce3133dcc402f1f843c4b7b52dc0399d86c2a6

Attack / Description	The function performs purely accounting updates for moving stake between appchains. Despite being \`payable\`, it neither reads nor uses \`msg.value\`, and no ETH is expected during a stake transfer. Because the contract lacks a refund path for unsolicited ETH and withdrawals depend strictly on prior \`initializeWithdrawal\` bookkeeping, any ETH sent to this function increases the contract balance without corresponding state, making it stuck. This breaks the expectation that value-bearing functions either consume or reject ETH and violates fund-safety guarantees by allowing accidental value loss. **Proof of Concept** `// SPDX-License-Identifier: UNLICENSED` `pragma solidity 0.8.28;` `import "forge-std/Test.sol";` `import {SyndStaking} from "../../src/staking/SyndStaking.sol";` `contract H03_SyndStaking_StageStakeTransfer_UnnecessaryPayable_PoC is Test {` `    SyndStaking staking;`

```
    address user = address(0xBEEF);

    function setUp() public {

        staking = new SyndStaking();

        // Warp into epoch 1 right after start

        uint256 start = staking.getEpochStart(1);

        vm.warp(start + 1);

        // Fund user and stake 5 ether into appchain 1

        vm.deal(user, 10 ether);

        vm.prank(user);

        staking.stakeSynd{value: 5 ether}(1);

        // Precondition: contract holds exactly 5 ether from staking

        assertEq(address(staking).balance, 5 ether, "pre: staking balance");

    }

    function test_unnecessary_payable_locks_eth() public {

        // User performs a stake transfer of 1 ether from appchain 1 -> 2

        // BUT mistakenly sends 1 ether along with the call. Function is payable
yet ignores msg.value.
```

```
        vm.prank(user);

        staking.stageStakeTransfer{value: 1 ether}(1, 2, 1 ether);

        // Contract balance increased by 1 ether, even though transfer is purely
accounting and should not accept ETH

        assertEq(address(staking).balance, 6 ether, "contract balance increased
unexpectedly");

        // The 1 ether is not reflected in stake accounting:

        // After transferring 1 from 1->2, user has 4 on appchain 1 and 1 on
appchain 2

        assertEq(staking.userAppchainTotal(user, 1), 4 ether, "appchain 1 total
incorrect");

        assertEq(staking.userAppchainTotal(user, 2), 1 ether, "appchain 2 total
incorrect");

        // No stake accounting changed except appchain redistribution; user still
has 5 total.

        assertEq(staking.userTotal(user), 5 ether, "userTotal should remain 5");

        assertEq(staking.userAppchainTotal(user, 1), 4 ether, "appchain 1 total
should be 4");

        assertEq(staking.userAppchainTotal(user, 2), 1 ether, "appchain 2 total
should be 1");
```

	<pre> // Since stageStakeTransfer is payable but ignores msg.value, the extra 1 ether is accepted // and there is no path in stageStakeTransfer to send it back. This demonstrates unintended ETH acceptance. } }</pre>
Code	Line 308-349 (SyndicateFactory.sol): <pre> function stageStakeTransfer(uint256 fromAppchainId, uint256 toAppchainId, uint256 amount) external payable nonReentrant whenNotPaused { if (amount == 0) { revert InvalidAmount(); } if (fromAppchainId == 0 || toAppchainId == 0) {</pre>

	`` revert InvalidAppchainId(); } if (fromAppchainId == toAppchainId) { revert SameAppchainTransfer(); } if (userAppchainTotal[msg.sender][fromAppchainId] < amount) { revert InsufficientStake(); } uint256 epochIndex = getCurrentEpoch(); if (userFinalizedEpochCount[msg.sender] < epochIndex) { finalizeUserEpochs(msg.sender); } if (appchainFinalizedEpochCount[fromAppchainId] < epochIndex) { finalizeAppchainEpochs(fromAppchainId); } ```

	`` if (userAppchainFinalizedEpochCount[msg.sender][fromAppchainId] < epochIndex) { finalizeUserAppchainEpochs(msg.sender, fromAppchainId); } epochUserAppchainTotal[epochIndex][msg.sender][fromAppchainId] += amount; epochAppchainTotal[epochIndex][fromAppchainId] += amount; userAppchainTotal[msg.sender][fromAppchainId] -= amount; appchainTotal[fromAppchainId] -= amount; epochUserAppchainAdditions[epochIndex][msg.sender][toAppchainId] += amount; epochAppchainAdditions[epochIndex][toAppchainId] += amount; userAppchainTotal[msg.sender][toAppchainId] += amount; appchainTotal[toAppchainId] += amount; emit StakeTransfer(epochIndex, msg.sender, amount, fromAppchainId, toAppchainId); } ```
Result/Recommendation	Remove `payable` from `stageStakeTransfer` to reject unexpected ETH: ``` function stageStakeTransfer(uint256 fromAppchainId, uint256 toAppchainId, uint256 amount) external ```

	`` nonReentrant whenNotPaused { // ... existing logic } ``` Alternatively, explicitly guard value: \`require(msg.value == 0, "no ETH expected");\` or revert if \`msg.value != 0\`.

6.2.3 Function Selector Tight Coupling in L1Relayer Cross-Chain Messaging

Severity: MEDIUM
Status: FIXED
File(s) affected: synd-contracts/src/staking/L1Relayer.sol
Update: https://github.com/SyndicateProtocol/syndicate-appchains/commit/4e35479ee6d9104bb0172b950e8bcb26a42879e3

Attack / Description	The L1Relayer contract's `_relay` function uses `this.relay.selector` when encoding the message payload for the L2Relayer. This creates an implicit assumption that L1 and L2 relay functions have identical signatures, which is a design coupling between contracts that creates maintenance risk. Currently, both contracts have matching signatures (`relay(address,uint256)` with selector `0xeeec0e24`), so the system functions correctly. However, this design breaks the separation of concerns principle. If either contract's function signature changes independently (e.g., during refactoring, adding parameters, or upgrades), the encoded selector would no longer match, causing all cross-chain operations to fail.

This creates technical debt by making an implicit dependency that isn't enforced by interfaces or compilation, which could lead to operational issues during future development.

Proof of Concept

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

pragma solidity 0.8.28;

import "forge-std/Test.sol";

import {L1Relayer} from "src/staking/L1Relayer.sol";

import {L2Relayer} from "src/staking/L2Relayer.sol";

contract MockToken {

    mapping(address => uint256) public balances;

    mapping(address => mapping(address => uint256)) public allowances;

    function balanceOf(address account) external view returns (uint256) {

        return balances[account];

    }

    function approve(address spender, uint256 amount) external returns (bool) {

        allowances[msg.sender][spender] = amount;

        return true;
```


hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

```
    }

    function mint(address to, uint256 amount) external {

        balances[to] = amount;

    }

}

contract MockOPBridge {

    function depositERC20To(address, address, address, uint256, uint32, bytes
calldata) external {}

}

contract MockOPMessageRelayer {

    address public lastTarget;

    bytes public lastMessage;

    uint32 public lastMinGasLimit;

    function sendMessage(address target, bytes memory message, uint32 minGasLimit)
external {

        lastTarget = target;

        lastMessage = message;

        lastMinGasLimit = minGasLimit;
```


hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

```
    }
}
contract L1RelayerSelectorMismatchTest is Test {
    L1Relayer l1Relayer;
    MockToken l1Token;
    MockOPBridge bridge;
    MockOPMessageRelayer messageRelayer;
    address l2Token = address(0x2);
    address l2RelayerAddr = address(0x3);
    address admin = address(this);
    function setUp() public {
        l1Token = new MockToken();
        bridge = new MockOPBridge();
        messageRelayer = new MockOPMessageRelayer();
        l1Relayer = new L1Relayer(
            address(bridge),
            address(messageRelayer),
```

```
            address(l1Token),

            l2Token,

            l2RelayerAddr,

            admin

        );

        // Give some tokens to L1Relayer

        l1Token.mint(address(l1Relayer), 1000);

    }

    function testSelectorMismatch() public {

        // Get the selector for L1Relayer's relay function

        bytes4 l1RelaySelector = l1Relayer.relay.selector;

        // For comparison, let's also get the selector for the expected L2
interface

        // relay(address,uint256) signature

        bytes4 expectedSelector = bytes4(keccak256("relay(address,uint256)"));

        // Call relay which will trigger _relay and sendMessage

        l1Relayer.relay(address(0xBEEF), 42);
```

```
        // Verify the message was sent
        assertEq(messageRelayer.lastTarget(), l2RelayerAddr);
        // Extract the selector from the message
        bytes4 selectorInMessage = bytes4(messageRelayer.lastMessage());
        // Extract parameters from the message (skip first 4 bytes for selector)
        bytes memory messageData = messageRelayer.lastMessage();
        bytes memory params = new bytes(messageData.length - 4);
        for (uint i = 0; i < params.length; i++) {
            params[i] = messageData[i + 4];
        }
        (address dest, uint256 epoch) = abi.decode(params, (address, uint256));
        assertEq(dest, address(0xBEEF));
        assertEq(epoch, 42);
        // Verify both selectors are currently the same (system works correctly)
        assertEq(selectorInMessage, l1RelaySelector, "Message contains L1Relayer
selector");
```


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

	`` assertEq(selectorInMessage, expectedSelector, "Selectors currently match - system works"); // The design issue: L1Relayer uses this.relay.selector // This creates tight coupling - if either contract changes independently, the bridge breaks console.log("L1Relayer selector:", vm.toString(l1RelaySelector)); console.log("Expected selector: ", vm.toString(expectedSelector)); console.log("Message selector: ", vm.toString(selectorInMessage)); console.log("STATUS: System works correctly, but uses `this.relay.selector` creating coupling"); } } ```
Code	Line 144-148 (L1Relayer.sol) ``` function _relay(address destination, uint256 epochIndex) internal { IOPMessageRelayer(opMessageRelayer).sendMessage(l2Relayer, abi.encodeWithSelector(this.relay.selector, destination, epochIndex), minGasLimit); } ```

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

Result/Recommendation	Replace: `abi.encodeWithSelector(this.relay.selector, destination, epochIndex)` with: `abi.encodeWithSelector(IL2Relayer.relay.selector, destination, epochIndex)` where \`IL2Relayer\` is an explicit interface defining the L2 contract's expected function signature: `interface IL2Relayer {` `    function relay(address destination, uint256 epochIndex) external;` `}`

6.2.4 Epoch Underflow Before Start Time in EpochTracker

Severity: MEDIUM
Status: ACKNOWLEDGED
File(s) affected: synd-contracts/src/staking/EpochTracker.sol
Update: We acknowledge this edge case and have decided not to make any changes for a few reasons: No contract should/will have logic around an epoch before 1, The hardcoded START TIME is already in the past and will not change, this effectively makes this edge case impossible to hit.

Attack / Description	The \`EpochTracker\` contract calculates the current epoch as: `return ((block.timestamp - START_TIMESTAMP) / EPOCH_DURATION) + 1;` If \`block.timestamp < START_TIMESTAMP\`, this subtraction underflows and reverts in Solidity ≥0.8. Any contract inheriting from \`EpochTracker\` (including \`BasePool\`, \`SyndStaking\`, \`GasCounter\`, \`Refunder\`, \`L1Relayer\`, \`L2Relayer\`, \`EmissionsCalculator\`, \`EmissionsScheduler\`)

will have all epoch-dependent functions bricked before the start time. This breaks the liveness and availability guarantees of the staking and emissions system.

Proof of Concept

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.28;

import {Test} from "forge-std/Test.sol";
import {EpochTracker} from "src/staking/EpochTracker.sol";

contract TrackerImpl is EpochTracker {}

contract EpochTrackerUnderflowPoC is Test {
    EpochTracker public tracker;

    function setUp() public {
        tracker = new TrackerImpl();
    }
```


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

	`` function test_getCurrentEpoch_underflow_reverts() public { // Warp to before the START_TIMESTAMP uint256 beforeStart = tracker.START_TIMESTAMP() - 1; vm.warp(beforeStart); // Should revert due to underflow vm.expectRevert(); tracker.getCurrentEpoch(); } } ```
Code	Line 26-30 (EpochTracker.sol) ``` function getCurrentEpoch() public view returns (uint256) { // Since all the epoch finalization counts are initialized to 0, // we start the epochs at 1 to make sure we will finalize the first epoch. return ((block.timestamp - START_TIMESTAMP) / EPOCH_DURATION) + 1; } ```

Result/Recommendation	Add a precondition check in \`getCurrentEpoch()\` to return 0 (or 1, depending on design) if \`block.timestamp < START_TIMESTAMP\`: `function getCurrentEpoch() public view returns (uint256) {` `    if (block.timestamp < START_TIMESTAMP) {` `        return 0; // or 1, depending on design` `    }` `    return ((block.timestamp - START_TIMESTAMP) / EPOCH_DURATION) + 1;` `}`

6.2.5 Double-spend/Incorrect Fee Accounting in L2Relayer

Severity: MEDIUM
Status: ACKNOWLEDGED
File(s) affected: synd-contracts\src\staking\L2Relayer.sol
Update: We are ok with using some of the funds for gas purposes as its a very specific bridging usecase that Syndicate is ultimately paying for.

Attack / Description	The \`_relay\` function in L2Relayer sets \`tokenTotalFeeAmount = amount\` and computes \`l2CallValue = amount - gasCost\`. When relaying, the bridge is authorized to pull the full \`amount\` as fees, but only \`amount - gasCost\` is used as the call value for the destination. This breaks value conservation and can lead to over-collection or mis-accounting, especially if the bridge does not refund the difference. The bug is triggered by normal use of the relay function with nonzero gas settings. **Proof of Concept**

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

pragma solidity 0.8.28;



import "forge-std/Test.sol";

import {L2Relayer} from "src/staking/L2Relayer.sol";

import {IArbBridge} from "src/staking/L2Relayer.sol";



contract TestToken {

    mapping(address => uint256) public balanceOf;

    mapping(address => mapping(address => uint256)) public allowance;

    function approve(address s, uint256 a) external returns (bool) {
allowance[msg.sender][s] = a; return true; }

    function transferFrom(address f, address t, uint256 a) external returns (bool)
{

        require(allowance[f][msg.sender] >= a && balanceOf[f] >= a);

        allowance[f][msg.sender] -= a; balanceOf[f] -= a; balanceOf[t] += a; return
true;

    }

    function mint(address to, uint256 a) external { balanceOf[to] += a; }
```


hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

```
}



contract MockArbBridge is IArbBridge {

    address public immutable token;

    uint256 public lastL2CallValue;

    uint256 public lastTokenTotalFeeAmount;

    constructor(address _token) { token = _token; }

    function unsafeCreateRetryableTicket(

        address, uint256 l2CallValue, uint256, address, address, uint256, uint256,
uint256 tokenTotalFeeAmount, bytes calldata

    ) external returns (uint256) {

        lastL2CallValue = l2CallValue; lastTokenTotalFeeAmount =
tokenTotalFeeAmount;

        TestToken(token).transferFrom(msg.sender, address(this),
tokenTotalFeeAmount);

        return 1;

    }

}
```

	`` contract L2Relayer_DoubleSpendFeeAccounting_Test is Test { function test_DoubleSpendFeeAccounting() public { TestToken token = new TestToken(); MockArbBridge bridge = new MockArbBridge(address(token)); L2Relayer relayer = new L2Relayer(address(bridge), address(token), address(0xBEEF), address(this)); relayer.setGasSettings(10, 1); // gasCost = 10 token.mint(address(relayer), 1000); relayer.relay(address(0xD1E5), 1); assertEq(bridge.lastTokenTotalFeeAmount(), 1000, "Bridge pulled full amount as fee"); assertEq(bridge.lastL2CallValue(), 990, "Call value excludes gasCost"); } } ```
Code	Line 121-145 (L2Relayer.sol) ``` function _relay(uint256 amount, address destination, uint256 epochIndex) internal { ```

hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

	`` uint256 gasCost = gasLimit * maxFeePerGas; // Validate that the gas cost is less than the amount being relayed // If the gas cost is greater than the amount, set the gas cost to 0 if (gasCost >= amount) { gasCost = 0; } // Calculate the call value to be sent to the destination contract uint256 callValue = amount - gasCost; // We use unsafe so the refunder address doesnt get aliased IArbBridge(arbBridge).unsafeCreateRetryableTicket(destination, callValue, 0, // Always 0 for custom gas token chains refunder, ```

hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

	`` refunder, gasLimit, maxFeePerGas, amount, abi.encodeWithSelector(IPool.deposit.selector, epochIndex)); } ```
Result/Recommendation	calculate total fees as `submissionCost + (gasLimit * maxFeePerGas)`, set `callValue = amount - totalFees`, and use `totalFees` for `tokenTotalFeeAmount`: `uint256 totalFees = submissionCost + (gasLimit * maxFeePerGas);` `uint256 callValue = amount - totalFees;` `require(callValue > 0, "Insufficient funds for operation");` `// Use totalFees for tokenTotalFeeAmount, not amount`

LOW ISSUES

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

6.2.6 Division Precision Loss in Reward Calculation

Severity: LOW

Status: FIXED
File(s) affected: synd-contracts/src/staking/BasePool.sol
Update: https://github.com/SyndicateProtocol/syndicate-appchains/commit/50b4a483c5f25b6400e89571fe9426d4e2954fe0

<table>
<tr>
<td>Attack / Description</td>
<td>
In `BasePool.getClaimableAmount()` at line 137, the reward calculation uses integer division:
<pre><code>uint256 userRewardShare = (rewardTotal * userStakedAmount) / totalStakedAmount;</code></pre>
This calculation truncates any remainder from the division operation. When rewards are small relative to the number of stakers, or when the reward amount is not perfectly divisible by the total stake, the division results in zero for individual users while leaving unclaimable dust in the contract.
The security guarantee broken is **reward completeness** - all deposited rewards should be claimable by users proportional to their stakes. This bug violates that guarantee by permanently locking remainder amounts that cannot be recovered.
For example, if 1 wei reward is deposited and there are 2 equal stakers, each user gets `(1 * 1) / 2 = 0` wei due to integer division truncation, leaving 1 wei permanently locked in the contract.
Proof of Concept
<pre><code>// SPDX-License-Identifier: UNLICENSED

pragma solidity 0.8.28;

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

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

import {Test} from "forge-std/Test.sol";</code></pre>
</td>
</tr>
</table>

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

```
contract M01_DivisionPrecisionLoss_PoC is Test {
    SyndStaking public staking;
    BasePool public basePool;
    address public user1;
    address public user2;

    function setUp() public {
        staking = new SyndStaking(address(this));
        basePool = new BasePool(address(staking));
        user1 = makeAddr("user1");
        user2 = makeAddr("user2");
        vm.deal(user1, 1 ether);
        vm.deal(user2, 1 ether);
        vm.warp(staking.START_TIMESTAMP());
        // Both users stake 1 wei (minimal, to maximize dust effect)
        vm.startPrank(user1);
```

```
    staking.stakeSynd{value: 1}(111);

    vm.stopPrank();

    vm.startPrank(user2);

    staking.stakeSynd{value: 1}(111);

    vm.stopPrank();

    // Move to next epoch

    vm.warp(block.timestamp + staking.EPOCH_DURATION());

}

function test_precision_loss_leaves_dust() public {

    // Deposit 1 wei as reward for epoch 2

    basePool.deposit{value: 1}(2);

    // Each user should get 0 wei due to integer division truncation

    assertEq(basePool.getClaimableAmount(2, user1), 0);

    assertEq(basePool.getClaimableAmount(2, user2), 0);

    // The 1 wei is locked in the contract, unclaimable

    assertEq(address(basePool).balance, 1);

}
```

```
function test_realistic_scenario_minimal_impact() public {
    // Reset with realistic stakes
    vm.startPrank(user1);
    staking.stakeSynd{value: 1 ether}(111);
    vm.stopPrank();
    vm.startPrank(user2);
    staking.stakeSynd{value: 1 ether}(111);
    vm.stopPrank();
    // Realistic reward amount
    uint256 rewardAmount = 0.001 ether; // 1000000000000000 wei
    basePool.deposit{value: rewardAmount}(2);

    uint256 claimable1 = basePool.getClaimableAmount(2, user1);
    uint256 claimable2 = basePool.getClaimableAmount(2, user2);
    uint256 totalClaimable = claimable1 + claimable2;
    uint256 dust = rewardAmount - totalClaimable;
```

	`` // Each user gets 500000000000000 wei, leaving 0 wei dust // With equal large stakes, precision loss is often 0 assertLe(dust, 2); // At most 2 wei dust assertLt(dust * 10000 / rewardAmount, 1); // Less than 0.01% loss } } ```
Code	Line 135-154 (BasePool.sol): ``` function getClaimableAmount(uint256 epochIndex, address user) public view returns (uint256) { if (epochRewardTotal[epochIndex] == 0) { return 0; } uint256 userStakedAmount = stakingContract.getUserStakeShare(epochIndex, user); if (userStakedAmount == 0) { ```

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

	`` return 0; } uint256 totalStakedAmount = stakingContract.getTotalStakeShare(epochIndex); if (totalStakedAmount == 0) { return 0; } uint256 rewardTotal = epochRewardTotal[epochIndex]; uint256 userRewardShare = (rewardTotal * userStakedAmount) / totalStakedAmount; // Subtract the amount the user has already claimed for this epoch return userRewardShare - claimed[epochIndex][user]; } ```
Result/Recommendation	Option 1: Accept as Expected Behavior (Recommended) Add documentation acknowledging the precision loss as expected behavior in edge cases: ``` /** * @notice Calculates the claimable reward amount for a user in a specific epoch * @dev Uses integer division which may result in small precision loss (dust) when ```

```
 *      reward amounts are not evenly divisible. This is expected behavior to
maintain
 *      gas efficiency. Dust amounts are typically negligible in normal operations.
 */
```

Option 2: Gas-Efficient Dust Handling

Implement a simple dust collection mechanism:

```
function getClaimableAmount(uint256 epochIndex, address user) public view returns
(uint256) {
    // ... existing logic ...

    uint256 userRewardShare = (rewardTotal * userStakedAmount) / totalStakedAmount;

    // Give any remaining dust to the last claimant
    uint256 totalClaimed = /* calculate total already claimed */;
    uint256 remaining = rewardTotal - totalClaimed;
    if (userRewardShare > 0 && remaining < userRewardShare) {
        userRewardShare = remaining; // Give all remaining to this user
    }
```


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

	`return userRewardShare - claimed[epochIndex][user];` `}` Option 3: Scaled Arithmetic (Higher Gas Cost) Only implement if perfect precision is absolutely required: `function getClaimableAmount(uint256 epochIndex, address user) public view returns (uint256) {` `// ... existing checks ...` `// Use scaled arithmetic to preserve precision` `uint256 PRECISION = 1e18;` `uint256 userRewardShare = (rewardTotal * userStakedAmount * PRECISION) / totalStakedAmount / PRECISION;` `return userRewardShare - claimed[epochIndex][user];` `}`

6.2.7 Gas Price Manipulation in GasCounter

Severity: LOW
Status: ACKNOWLEDGED

File(s) affected: synd-contracts/src/staking/GasCounter.sol

Attack / Description	The \`GasCounter._trackGas()\` function calculates gas cost as \`gasUsed * tx.gasprice\` (line 62 in GasCounter.sol). Since \`tx.gasprice\` is user-controlled, a user can call tracked functions with an extremely high gas price, inflating their recorded gas usage metrics. The gas tracking data is aggregated through the \`GasAggregator\` contract across multiple app chains. This breaks the security guarantee of accurate gas cost tracking. The vulnerability exists because the contract trusts user-provided gas price data without validation or capping, allowing manipulation of the gas accounting system. Economic Constraint: Critically, users must actually pay the high gas prices they set, creating a significant economic barrier to exploitation. The attack is only viable if the benefits from inflated gas metrics exceed the cost of paying inflated gas fees. The affected code in \`src/staking/GasCounter.sol\` at lines 58-65: `function _trackGas(uint256 gasUsed) internal {` `    uint256 currentEpoch = getCurrentEpoch();` `    // Calculate gas cost using current transaction gas price` `    uint256 gasPrice = tx.gasprice;` `    // WORKAROUND: estimate gas will give a wrong value when called with tx.gasprice 0`

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

```
    if (gasPrice == 0) {

        gasPrice = 1;

    }

    // Add gas and cost to current epoch

    tokensUsedPerEpoch[currentEpoch] += gasUsed * gasPrice;

}
```

The vulnerability propagates through any contract that inherits from `GasCounter` and uses the `trackGasUsage` modifier, such as `SyndicateSequencingChain`. The tracked gas data is collected by the `GasAggregator` contract, which aggregates gas usage across multiple app chains for accounting purposes.

Proof of Concept

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

pragma solidity 0.8.28;



import {GasCounter} from "../synd-contracts/src/staking/GasCounter.sol";

import {Test} from "../synd-contracts/lib/forge-std/src/Test.sol";



/// @title M-02_GasPriceManipulation_PoC.t.sol
```

```
/// @notice PoC for M-02: Gas Price Manipulation in GasCounter

contract M02_GasPriceManipulation_PoC is Test {

    GasCounterTestHarness public gasCounter;

    function setUp() public {

        gasCounter = new GasCounterTestHarness();

        // Warp to after START_TIMESTAMP to enable epoch tracking

        vm.warp(gasCounter.START_TIMESTAMP() + 1);

    }

    function test_gas_price_manipulation() public {

        // Get initial state

        uint256 epoch = gasCounter.getCurrentEpoch();

        uint256 initialTracked = gasCounter.tokensUsedPerEpoch(epoch);


        // Simulate a user calling a tracked function with a high gas price

        uint256 fakeGasPrice = 1_000_000_000_000; // 1,000 gwei (artificially high)

        vm.txGasPrice(fakeGasPrice);

        gasCounter.trackedFunction();
```

	`` // The tokensUsedPerEpoch should reflect the manipulated gas price uint256 trackedAfterHighGas = gasCounter.tokensUsedPerEpoch(epoch); uint256 highGasDelta = trackedAfterHighGas - initialTracked; assertGt(highGasDelta, 0, "Gas usage should be tracked with high gas price"); // Now call with a normal gas price uint256 normalGasPrice = 1 gwei; // 1000x lower vm.txGasPrice(normalGasPrice); gasCounter.trackedFunction(); uint256 trackedAfterNormalGas = gasCounter.tokensUsedPerEpoch(epoch); uint256 normalGasDelta = trackedAfterNormalGas - trackedAfterHighGas; // The high gas price transaction should have recorded ~1000x more "cost" // despite using similar actual gas assertGt(highGasDelta, normalGasDelta * 500, "High gas price should inflate recorded cost significantly"); // Demonstrate economic constraint: user actually pays the high gas price // In a real scenario, the user would pay fakeGasPrice * actualGasUsed ```

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

	`` emit log_named_uint("High gas price set (gwei)", fakeGasPrice / 1e9); emit log_named_uint("Normal gas price set (gwei)", normalGasPrice / 1e9); emit log_named_uint("High gas recorded cost", highGasDelta); emit log_named_uint("Normal gas recorded cost", normalGasDelta); } } /// @dev Minimal harness to expose trackGasUsage modifier contract GasCounterTestHarness is GasCounter { function trackedFunction() public trackGasUsage { // Do nothing, just consume minimal gas } } ```
Code	Line 53-66 (GasCounter.sol): ``` function _trackGas(uint256 gasUsed) internal { uint256 currentEpoch = getCurrentEpoch(); // Calculate gas cost using current transaction gas price ```

	`uint256 gasPrice = tx.gasprice;` `// WORKAROUND: estimate gas will give a wrong value when called with tx.gasprice 0` `if (gasPrice == 0) {` `gasPrice = 1;` `}` `// Add gas and cost to current epoch` `tokensUsedPerEpoch[currentEpoch] += gasUsed * gasPrice;` `}`
Result/Recommendation	Primary Solution: Replace `tx.gasprice` with `block.basefee` for more reliable and manipulation-resistant gas pricing: `function _trackGas(uint256 gasUsed) internal {` `uint256 currentEpoch = getCurrentEpoch();` `// Use block.basefee instead of tx.gasprice to prevent manipulation` `uint256 gasPrice = block.basefee;` `// WORKAROUND: estimate gas will give a wrong value when called with gasPrice 0` `if (gasPrice == 0) {` `gasPrice = 1;`

```
    }

    // Add gas and cost to current epoch

    tokensUsedPerEpoch[currentEpoch] += gasUsed * gasPrice;

}
```

Alternative Solution: Implement a maximum gas price cap:

```
uint256 gasPrice = tx.gasprice;

uint256 MAX_GAS_PRICE = 100 gwei; // Set reasonable maximum based on network conditions

if (gasPrice > MAX_GAS_PRICE) {

    gasPrice = MAX_GAS_PRICE;

}
```

Hybrid Approach: For more sophisticated tracking, consider using a combination:

```
uint256 gasPrice = tx.gasprice > block.basefee * 2 ? block.basefee : tx.gasprice;
```

The `block.basefee` approach is preferred as it provides protocol-consistent pricing that reflects actual network conditions and cannot be manipulated by individual users, while still being economically meaningful.

INFORMATIONAL ISSUES

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

6.3 Verify Claims

6.3.1 **Emission Schedule Integrity**

The audit must verify that the geometric decay formula for distributing 80M SYND over 48 epochs is implemented correctly, prevents double-minting, enforces sequential epoch processing, and cannot be manipulated through timing attacks or governance misconfiguration.

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 that partial-epoch participants receive correct rewards and that restaking, withdrawals, and delayed finalization cannot be exploited for excess rewards.

Status: tested and verified ✅

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

All L1 → L2 → L3 relay operations must be secure, atomic, and protected against replay, spoofing, or mis-routing of funds. Retryable ticket logic and fund recovery (via Refunder) must guarantee liveness and correctness under failed bridge scenarios.

Status: tested and verified ✅

6.3.4 **Reward Distribution & Gas-Based Incentives**

The BasePool and GasCounter contracts must ensure that sequencer and staking rewards are distributed proportionally to actual stake and gas consumption, without allowing manipulation of gas tracking, double-claims, or misallocation across appchains.

Status: tested and verified ✅

6.3.5 **Access Control & Emergency Response**

Critical roles (Admin, Decay Manager, Emissions, Pauser) must be strictly enforced to prevent unauthorized minting or parameter changes. The system’s pause mechanisms, reentrancy guards, 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 issue, classified as follows:
- zero critical issue was found.
- 1 high severity issue was found.
- 4 medium severity issues were found.
- 2 low severity issues were discovered

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 (16.09.2025): The Syndicate team has successfully addressed all identified issues from the audit. All high, medium, low severities have been mitigated based on the recommendations provided in the report. A follow-up review confirms that the fixes have been implemented effectively, ensuring the security and functionality of the smart contract. The updated codebase reflects the necessary improvements, and no further concerns remain.

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 industry-leading 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