



Syndicate
Crosschain Token (SYND)
SMART CONTRACT AUDIT
09.07.2025

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.....	4
2. About the Project and Company	5
2.1 Project Overview.....	6
3. Vulnerability & Risk Level	7
4. Auditing Strategy and Techniques Applied.....	8
4.1 Methodology	8
5. Metrics	9
5.1 Tested Contract Files	9
5.2 Inheritance Graph.....	10
5.3 Source Lines & Risk.....	11
5.4 Capabilities	12
5.5 Dependencies / External Imports.....	13
5.6 Source Unites in Scope	15
6. Scope of Work.....	18
6.1 Findings Overview	19
6.2 Manual and Automated Vulnerability Test.....	20
6.2.1 Unlimited Token Minting via Crosschain Bridge.....	20
6.2.2 Bridge Manager Privilege Escalation	24
6.2.3 Emission Schedule Bypass	27
6.2.4 Predictable Rate Limiting Allows Burst Minting	31
6.2.5 Gas Griefing via Bridge Removal Function.....	34
6.2.6 Transfer Lock Period Concerns	38



6.2.7 Bridge Limit Bypass via Reconfiguration	41
6.2.8 Missing Input Validation on Bridge Addresses	44
6.2.9 Divergence from Documentation: Bridge Config & Factory Removal.....	48
6.3 Verify Claims	50
7. Executive Summary.....	51
8. About the Auditor	52



1. Disclaimer

The audit makes no statements or warranties 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.

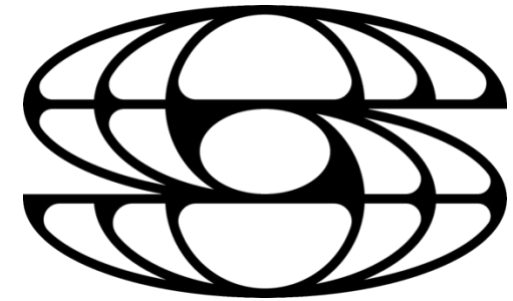
Major Versions / Date	Description
0.1 (25.06.2025)	Layout
0.4 (28.06.2025)	Automated Security Testing Manual Security Testing
0.5 (29.06.2025)	Verify Claims
0.9 (30.06.2025)	Summary and Recommendation
1.0 (04.07.2025)	Submission of findings
1.1 (07.07.2025)	Re-check
1.2 (09.07.2025)	Final document



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>



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

2.1 Project Overview

Syndicate is a modular, programmable infrastructure platform enabling developers and enterprises to launch scalable, production-ready on-chain applications across any EVM-compatible network. With a transaction-first architecture, Syndicate abstracts complex wallet management, signing flows, and multi-chain bridging into a single developer-friendly API stack.

At its foundation, Syndicate provides compliant, secure, and performant services for broadcasting transactions, managing attestations, and integrating on-chain logic with off-chain workflows. Its transaction cloud is built to support thousands of operations per second and offers compatibility with Ethereum, Layer 2s like Arbitrum and Optimism, and emerging Superchain networks.

Syndicate's infrastructure includes deterministic CREATE3-based deployment, ERC-7802-compatible crosschain token tooling, rate-limited bridge proxies, and automated emission schedulers, ideal for token launches, DePIN deployments, and DAO tooling. With native support for ERC20Votes, ERC20Permit, and gasless transactions, Syndicate empowers developers to build governance-ready applications with flexible emission and distribution models.

Use cases span from crosschain token issuance and NFT minting to embedded DeFi rails and programmable user wallets. Syndicate also offers REST APIs, SDKs, and webhook integrations to streamline deployment pipelines and orchestrate high-throughput, multi-chain interactions. Future development includes zk-powered privacy layers, intent-based execution, and integration with emerging L3 ecosystems.



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

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.



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

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/tree/main/synd-contracts/src/token>

Commit: 67515fe08a0b2c13c96dc03d9b6bd16f661ae621

File	Fingerprint (MD5)
./SyndicateToken.sol	5f1d9114ab83d396019c8c90d998199a
./SyndicateTokenCrosschain.sol	81b0c9cdaa5a3ea2a61e0ff5ec84291c
./crosschain/libraries/CREATE3.sol	182ba0ca2fd9612efce6d6c4bcf6f468
./crosschain/libraries/Bytes32AddressLib.sol	7526857abd0188726d1e495f8597167c
./crosschain/interfaces/IERC7802.sol	18cfbe1b9bd060a6ed76fb51af94a2bc
./crosschain/interfaces/IBridgeRateLimiter.sol	4fd390cdcf552522933870f42a26216
./bridges/BaseBridgeProxy.sol	cce5bf9f550a419ab1c45886b37bb3be
./bridges/OptimismBridgeProxy.sol	01dd62944d0e4e1e16c9f4b3f53b9b98
./bridges/ArbitrumBridgeProxy.sol	7a571a7d8dce4404d2b85482df064310
./TestnetSyndTokenCrosschain.sol	f94e6593d2656804d24757af74664ede
./SyndicateTokenEmissionScheduler.sol	788fb6a5d0f39d40f0d227b8babea389
./interfaces/IBridgeProxy.sol	5e69c868755ded4abeb5416e21daed5c

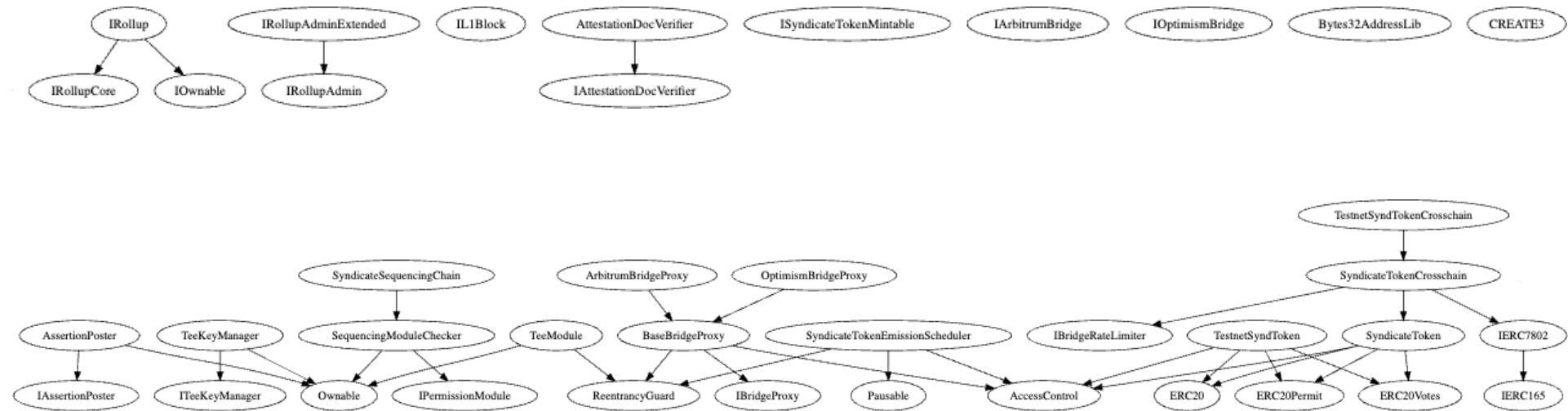


hello@softstack.io
www.softstack.io

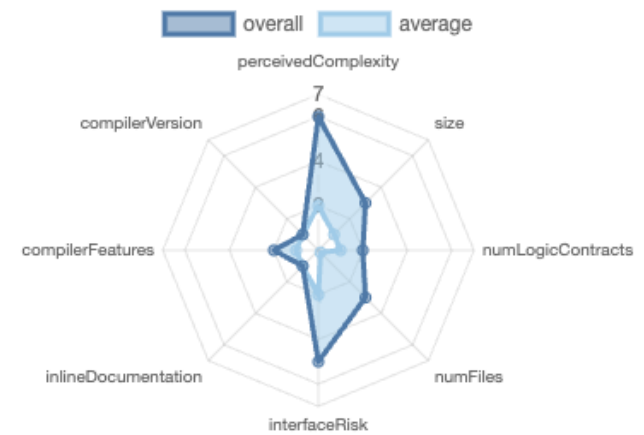
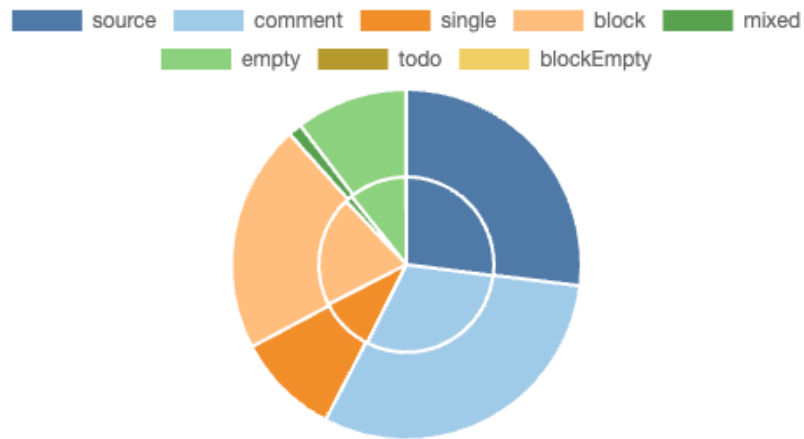
softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

5.2 Inheritance Graph



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 ^0.8.20 >=0.8.0				yes		yes (1 asm blocks)					
 Transfers ETH		 Low-Level Calls		 DelegateCall		 Uses Hash Functions		 ECRrecover		 New/Create/Create2	
						yes				yes → NewContract:NotInitializedModule → AssemblyCall:Name:create2	

Exposed Functions

 Public	 Payable			
109	3			
External	Internal	Private	Pure	View
93	98	7	9	49

StateVariables

Total	 Public
81	65



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

5.5 Dependencies / External Imports

Dependency / Import Path	Source
@arbitrum/nitro-contracts/src/bridge/IBridge.sol	https://github.com/OffchainLabs/nitro-contracts/src/bridge/IBridge.sol
@arbitrum/nitro-contracts/src/bridge/IOwnable.sol	https://github.com/OffchainLabs/nitro-contracts/src/bridge/IOwnable.sol
@arbitrum/nitro-contracts/src/libraries/IGasRefunder.sol	https://github.com/OffchainLabs/nitro-contracts/src/libraries/IGasRefunder.sol
@arbitrum/nitro-contracts/src/rollup/IRollupAdmin.sol	https://github.com/OffchainLabs/nitro-contracts/src/rollup/IRollupAdmin.sol
@arbitrum/nitro-contracts/src/rollup/IRollupCore.sol	https://github.com/OffchainLabs/nitro-contracts/src/rollup/IRollupCore.sol
@arbitrum/nitro-contracts/src/state/GlobalState.sol	https://github.com/OffchainLabs/nitro-contracts/src/state/GlobalState.sol
@offchainlabs/upgrade-executor/src/IUpgradeExecutor.sol	https://github.com/OffchainLabs/upgrade-executor/src/IUpgradeExecutor.sol
@openzeppelin/contracts/access/AccessControl.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/AccessControl.sol
@openzeppelin/contracts/access/IAccessControl.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/IAccessControl.sol



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Permit.sol
@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Votes.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.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/cryptography/ECDSA.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

Dependency / Import Path	Source
@openzeppelin/contracts/utils/introspection/IERC165.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/introspection/IERC165.sol
@openzeppelin/contracts/utils/structs/EnumerableSet.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/structs/EnumerableSet.sol
@sp1-contracts/ISP1Verifier.sol	https://github.com/succinctlabs/sp1-contracts/blob/main/contracts/src/ISP1Verifier.sol

5.6 Source Unites in Scope

File	Logic Contracts	Interfaces	Lines	nSLOC	Comment Lines
synd-contracts/src/SyndicateSequencingChain.sol	1		75	29	35
synd-contracts/src/withdrawal/AssertionPoster.sol	1	2	342	157	116
synd-contracts/src/withdrawal/IAssertionPoster.sol		1	11	3	6
synd-contracts/src/withdrawal/ITeeKeyManager.sol		1	11	3	6
synd-contracts/src/withdrawal/TeeModule.sol	1	1	264	164	61
synd-contracts/src/withdrawal/AttestationDocVerifier.sol	1		84	54	21
synd-contracts/src/withdrawal/TeeKeyManager.sol	1		71	35	26



File	Logic Contracts	Interfaces	Lines	nSLOC	Comment Lines
synd-contracts/src/withdrawal/IAttestationDocVerifier.sol		1	15	3	7
synd-contracts/src/SequencingModuleChecker.sol	1		77	34	30
synd-contracts/src/token/TestnetSyndTokenCrosschain.sol	1		105	27	65
synd-contracts/src/token/SyndicateTokenEmissionScheduler.sol	1	1	402	142	192
synd-contracts/src/token/TestnetSyndToken.sol	1		95	40	40
synd-contracts/src/token/interfaces/IBridgeProxy.sol		1	41	3	34
synd-contracts/src/token/SyndicateTokenCrosschain.sol	1		425	175	160
synd-contracts/src/token/SyndicateToken.sol	1		305	93	158
synd-contracts/src/token/bridges/ArbitrumBridgeProxy.sol	1	1	269	72	163
synd-contracts/src/token/bridges/OptimismBridgeProxy.sol	1	1	183	48	118
synd-contracts/src/token/bridges/BaseBridgeProxy.sol	1		338	103	179
synd-contracts/src/token/crosschain/interfaces/IERC7802.sol		1	40	6	25
synd-contracts/src/token/crosschain/libraries/Bytes32AddressLib.sol	1		14	9	3



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

File	Logic Contracts	Interfaces	Lines	nSLOC	Comment Lines
synd-contracts/src/token/crosschain/libraries/CREATE3.sol	1		66	24	34
synd-contracts/src/token/crosschain/interfaces/IBridgeRateLimiter.sol		1	102	19	57
Totals	16	12	3335	1243	1536

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 Protocol team has delivered a modular suite of smart contracts supporting on-chain token issuance, emission scheduling, and crosschain bridging.

The team has outlined the following critical security and functionality assumptions for the token system:

1. **Role-Based Emission Control**

Only the designated EmissionScheduler should be able to mint new SYND tokens via EMISSION_MINTER_ROLE. Epoch timing and token amounts must be enforced with no early or excess minting possible.

2. **Crosschain Mint/Burn Authorization**

Only explicitly authorized bridges may invoke crosschainMint or crosschainBurn. The audit must validate rate limit tracking, role isolation, and abuse resistance under high volume.

3. **Deterministic CREATE3 Deployment**

Contracts must be deployed using the same salt and logic on all chains, resulting in consistent addresses. Address prediction and deployment validation logic must be accurate and collision-resistant.

4. **Bridge Proxy Safety & ETH Handling**

Bridge proxies (Arbitrum, Optimism) must enforce per-transaction and daily limits, correctly manage ETH for gas costs, and expose no reentrancy or DoS vectors during bridging operations.

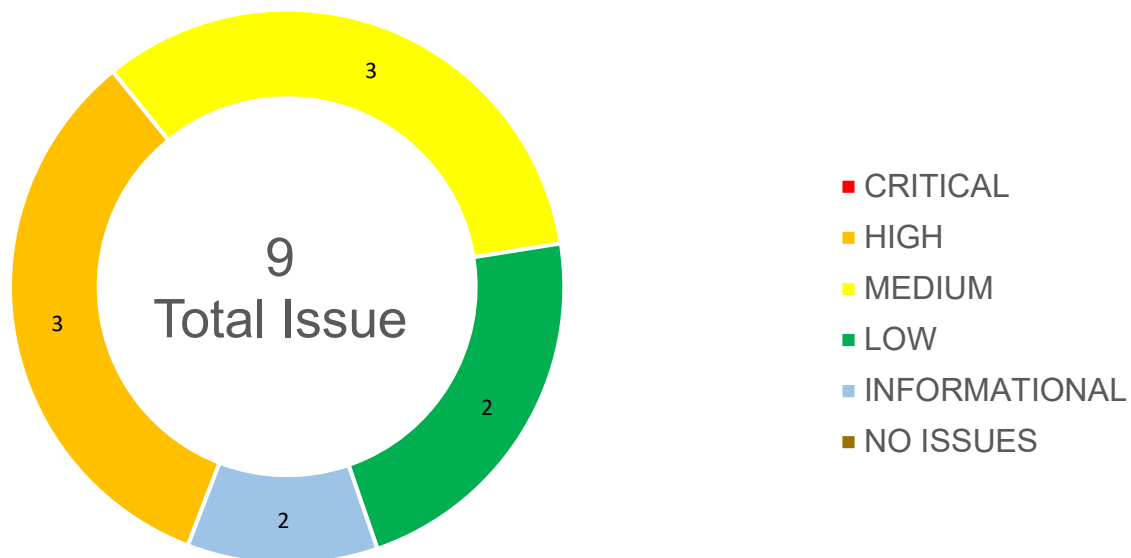
5. **Emergency Pause & Recovery Readiness**

Pause mechanisms must halt emission or bridging flows without corrupting contract state. Only authorized roles should be able to pause/unpause, with no single point of failure.

The primary goal of this audit is to validate these assumptions and ensure the SYND Token System is secure, fault-tolerant, and production-ready. The audit team will also provide feedback on gas efficiency, deployment correctness, and upgradeability strategy.



6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Unlimited Token Minting via Crosschain Bridge	HIGH	FIXED
6.2.2	Bridge Manager Privilege Escalation	HIGH	FIXED
6.2.3	Emission Schedule Bypass	HIGH	FIXED
6.2.4	Predictable Rate Limiting Allows Burst Minting	MEDIUM	FIXED
6.2.5	Gas Griefing via Bridge Removal Function	MEDIUM	FIXED
6.2.6	Transfer Lock Period Concerns	MEDIUM	FIXED
6.2.7	Bridge Limit Bypass via Reconfiguration	LOW	ACKNOWLEDGED
6.2.8	Missing Input Validation on Bridge Addresses	LOW	FIXED
6.2.9	Divergence from Documentation: Bridge Config & Factory Removal	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 **three High issues** in the code of the smart contract.

6.2.1 Unlimited Token Minting via Crosschain Bridge

Severity: HIGH

Status: FIXED

File(s) affected: SyndicateTokenCrosschain.sol

Update: <https://github.com/SyndicateProtocol/syndicate-appchains/pull/584/commits/27a4a95f8dc127832e0d63f81c71a1d514120fec>

Attack / Description	<p>The crosschainMint function within the SyndicateTokenCrosschain.sol contract is missing a critical validation step: it fails to check minting requests against the global TOTAL_SUPPLY of 1 billion tokens. While the base SyndicateToken contract correctly enforces this fundamental check, the crosschain implementation completely omits it. This discrepancy creates a severe vulnerability that allows an authorized bridge to create an infinite number of tokens, fundamentally breaking the protocol's tokenomic model.</p> <p>Risk Impact:</p> <p>The impact of this vulnerability is catastrophic and poses an existential threat to the protocol. A malicious or compromised bridge operator could exploit this flaw to:</p>
----------------------	--



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

- **Trigger Infinite Inflation:** Mint tokens without limit, destroying the token's scarcity and value proposition.
- **Manipulate Markets:** Flood the market with an unlimited supply of new tokens, causing a complete collapse of the token's price.
- **Drain Protocol Liquidity:** Sell the illicitly minted tokens, draining all value from the protocol and its liquidity pools.
- **Violate User Trust:** Break the fundamental promise of a fixed total supply, causing irreparable damage to the project's reputation and community trust.

Proof of Concept:

We have confirmed this vulnerability with a dedicated test case that demonstrates the exploit.

Test Case:

```
function test_H01_UnlimitedTokenMinting() public {
    // 1. Setup: A bridge is configured with unlimited minting limits by the
    manager.
    vm.prank(BRIDGE_MANAGER);
    crosschainToken.setBridgeLimits(MALICIOUS_BRIDGE, type(uint256).max,
    type(uint256).max);

    // 2. Exploit: The malicious bridge mints 2 billion tokens, an amount that
    // should be impossible given the 1 billion token total supply.
    vm.prank(MALICIOUS_BRIDGE);
    crosschainToken.crosschainMint(USER, 2_000_000_000 * 10**18); // 2B tokens

    // 3. Result: The test asserts that the final supply is greater than the
    intended
    // supply cap, confirming the vulnerability.
    assertTrue(finalSupply > totalSupplyCap, "Total supply should exceed cap");
}
```



	<p>Test Output:</p> <p>Initial Supply: 900,000,000 Total Supply Cap: 1,000,000,000 Final Supply: 2,900,000,000</p> <p>This output clearly shows the total supply was inflated by 1.9 billion tokens over its intended limit.</p>
<p>Code</p>	<p>Line 81 - 92 (SyndicateTokenCrosschain.sol):</p> <pre>function crosschainMint(address to, uint256 amount) external override { if (to == address(0)) revert ZeroAddress(); if (amount == 0) revert ZeroAmount(); // Check if bridge is authorized and has sufficient limits _validateAndUseMintLimit(msg.sender, amount); // MINT TOKENS - NO TOTAL SUPPLY CHECK! _mint(to, amount); emit CrosschainMint(to, amount, msg.sender); }</pre> <p>Line 153-163 Secure Implementation (SyndicateToken.sol):</p> <pre>function mint(address to, uint256 amount) external onlyRole(EMISSION_MINTER_ROLE) { if (to == address(0)) revert ZeroAddress(); if (amount == 0) revert ZeroAmount();</pre>

	<pre>// CORRECT: Ensures the total supply cap is not exceeded. if (totalSupply() + amount > TOTAL_SUPPLY) { revert ExceedsTotalSupply(); } _mint(to, amount); }</pre>
Result/Recommendation	<p>To mitigate this critical risk, we strongly advise the following actions be taken immediately:</p> <ol style="list-style-type: none"> 1. Implement Total Supply Validation: The most critical step is to add the totalSupply check to the crosschainMint function. This check must be performed <i>before</i> any minting logic is executed. <p>Suggested Code:</p> <pre>function crosschainMint(address to, uint256 amount) external override { if (to == address(0)) revert ZeroAddress(); if (amount == 0) revert ZeroAmount(); // ADD THIS CRITICAL CHECK: if (totalSupply() + amount > TOTAL_SUPPLY) { revert ExceedsTotalSupply(); } _validateAndUseMintLimit(msg.sender, amount); _mint(to, amount); emit CrosschainMint(to, amount, msg.sender); }</pre> <ol style="list-style-type: none"> 2. Enhance Cross-Chain Logic: Implement a robust cross-chain supply tracking mechanism to ensure the global token supply is consistently enforced across all integrated chains.

6.2.2 Bridge Manager Privilege Escalation

Severity: HIGH

Status: FIXED

File(s) affected: SyndicateTokenCrosschain.sol

Update: <https://github.com/SyndicateProtocol/syndicate-appchains/pull/584/commits/96738d2e49ff4dc019d05ecbd4d64436335f3cbe>

Attack / Description

The protocol's access control model contains a critical flaw related to the BRIDGE_MANAGER_ROLE. This role is overly permissive, allowing a manager to add *any* address as an authorized bridge without requiring secondary approval or validation. Crucially, a manager can add their own address, instantly escalating their privileges to include direct and potentially unlimited minting capabilities. This fundamentally breaks the principle of least privilege and the separation of duties.

Risk Impact:

This vulnerability creates a powerful vector for an **insider attack** and poses a significant threat to the protocol's integrity. The primary risks include:

- **Centralized Control:** It introduces a single point of failure, as a single compromised BRIDGE_MANAGER_ROLE key can lead to a complete takeover of the token supply.
- **Insider Risk:** A malicious administrator or an attacker who compromises an admin key can grant themselves unlimited minting power.
- **Governance Bypass:** It allows an individual to circumvent any intended multi-signature or DAO-based governance controls that should be in place for such a sensitive operation.
- **Trust Violation:** The ability for a manager to unilaterally grant themselves minting rights defeats the purpose of a structured, role-based access control system.



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

Proof of Concept:

We have verified this flaw with a test case that clearly demonstrates the privilege escalation path.

Test Case:

```
function test_H02_BridgeManagerPrivilegeEscalation() public {
    // 1. Setup: The bridge manager adds their own address as a new bridge,
    // granting themselves unlimited minting power.
    vm.prank(BRIDGE_MANAGER);
    crosschainToken.setBridgeLimits(BRIDGE_MANAGER, type(uint256).max,
    type(uint256).max);

    // 2. Exploit: Now acting as a bridge, the manager mints 1 billion tokens
    // directly to their own wallet.
    vm.prank(BRIDGE_MANAGER);
    crosschainToken.crosschainMint(BRIDGE_MANAGER, 1_000_000_000 * 10**18); // 1B
    tokens

    // 3. Result: The test confirms the manager's balance has increased by 1
    billion tokens,
    // proving the privilege escalation was successful.
    assertTrue(bridgeManagerBalance == 1_000_000_000 * 10**18, "Privilege
    escalation successful");
}
```

Test Output:

Bridge Manager Balance: 1,000,000,000

The test confirms that the manager, who should not have direct minting rights, successfully created and claimed 1 billion tokens.

Code	<p>Line 120 – 149 (SyndicateTokenCrosschain.sol):</p> <pre> function setBridgeLimits(address bridge, uint256 dailyMintLimit, uint256 dailyBurnLimit) external override onlyRole (BRIDGE_MANAGER_ROLE) { if (bridge == address(0)) revert ZeroAddress(); // NO VALIDATION: The bridge manager can add their own address. // The dailyMintLimit can also be set to an unlimited value. if (!isBridgeAdded[bridge]) { bridges.push(bridge); isBridgeAdded[bridge] = true; emit BridgeAdded(bridge, dailyMintLimit, dailyBurnLimit); } bridgeConfigs[bridge] = BridgeConfig({ dailyMintLimit: dailyMintLimit, // Can be set to unlimited! // ... }); } </pre>
Result/Recommendation	<p>A defense-in-depth approach is required to fully address this systemic issue. We recommend the following actions:</p> <ol style="list-style-type: none"> 1. Implement Strict In-Contract Validation: The setBridgeLimits function must be updated with immediate, non-negotiable checks to enforce the separation of duties. <p>Suggested Code:</p>

```
function setBridgeLimits(address bridge, uint256 dailyMintLimit, uint256
dailyBurnLimit)
    external
    override
    onlyRole(BRIDGE_MANAGER_ROLE)
{
    if (bridge == address(0)) revert ZeroAddress();

    // ADD: Prevent a manager from adding themselves.
    if (bridge == msg.sender) revert CannotAddSelfAsBridge();

    // ADD: Require the new bridge to be a smart contract, not a regular wallet
    address.
    if (bridge.code.length == 0) revert BridgeMustBeContract();

    // ... rest of function
}
```

2. Enforce Multi-Signature or Timelock Controls: For sensitive actions like adding a new bridge or changing its limits, require approval from a multi-signature wallet or enforce a timelock. This ensures that no single individual can make such a critical change unilaterally.

3. Establish Global Limit Caps: Consider adding a protocol-wide constant that defines a maximum possible dailyMintLimit. This would prevent a manager from setting an effectively infinite limit, even if other controls fail.

6.2.3 Emission Schedule Bypass

Severity: HIGH

Status: FIXED

File(s) affected: SyndicateTokenCrosschain.sol

Update: <https://github.com/SyndicateProtocol/syndicate-appchains/pull/584/commits/f637481fc7801f49fa25eef4aa6dd2bf50d003f9>



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

Attack / Description

The crosschain bridge system operates entirely independently of the SyndicateTokenEmissionScheduler. This disconnect allows an authorized bridge to mint tokens that are part of the protocol's four-year emission allocation and make them instantly liquid. In effect, the crosschain minting function provides a backdoor to completely bypass the vesting schedule, which is a cornerstone of the token's economic design.

Risk Impact:

This vulnerability poses a severe threat to the protocol's long-term stability and the trust of its stakeholders. The primary risks include:

- **Vesting Schedule Violation:** It renders the four-year token release schedule meaningless, as tokens intended for gradual release can be made available instantly.
- **Severe Market Shock:** A malicious actor could flood the market with the entire emission supply, causing extreme price volatility and collapsing the token's value.
- **Breach of Investor Trust:** This flaw violates the commitment made to investors and community members who are bound by the vesting schedule, leading to a significant loss of confidence.
- **Economic Instability:** The carefully planned tokenomics, designed to ensure a stable and gradual distribution, are completely disrupted.

Proof of Concept:

We confirmed this vulnerability with a test case that successfully bypasses the entire four-year emission schedule in a single transaction.

Test Case:

```
function test_H03_EmissionScheduleBypass() public {  
    // 1. Setup: A bridge is configured with a mint limit large enough  
    // to accommodate the entire emission supply.  
    vm.prank(BRIDGE_MANAGER);  
    crosschainToken.setBridgeLimits(MALICIOUS_BRIDGE, 200 000 000 * 10**18, 0);  
}
```



	<pre>// 2. Exploit: The bridge instantly mints the full 100 million tokens // from the emission allocation. vm.prank(MALICIOUS_BRIDGE); crosschainToken.crosschainMint(USER, 100_000_000 * 10**18); // 3. Result: The test confirms the user's balance increased by 100 million tokens, // proving the entire vesting schedule was bypassed. assertTrue(userBalance == 100_000_000 * 10**18, "Emission schedule bypassed"); }</pre> <p>Test Output:</p> <p>Emission Amount Bypassed: 100,000,000</p> <p>This test verifies that the 100 million tokens intended for a four-year release were made liquid instantaneously.</p>
<p>Code</p>	<p>Line 80 – 88 (SyndicateTokenCrosschain.sol):</p> <pre>// This function has no connection to the emission scheduler. function crosschainMint(address to, uint256 amount) external override { // NO integration with emission schedule! // NO validation against any emission budget! _mint(to, amount); }</pre> <p>Lines 153-163 (SyndicateToken.sol):</p> <pre>// This function is correctly restricted to the emission contract. function mint(address to, uint256 amount) external onlyRole(EMISSION_MINTER_ROLE) { // Proper validation exists here if (totalSupply() + amount > TOTAL_SUPPLY) {</pre>

	<pre> revert ExceedsTotalSupply(); } _mint(to, amount); } </pre>
Result/Recommendation	<p>To fix this critical architectural flaw, the crosschain minting process must be made subordinate to the emission schedule.</p> <ol style="list-style-type: none"> 1. Integrate with an Emission Budget: The most robust solution is to introduce a budget system. The SyndicateTokenEmissionScheduler should be the only contract capable of allocating a specific "emission budget" to an authorized bridge. The crosschainMint function must then check that the mint amount does not exceed the bridge's available budget. <p>Suggested Code:</p> <pre> // Add a mapping to track the budget allocated to each bridge mapping(address => uint256) public bridgeEmissionBudgets; function crosschainMint(address to, uint256 amount) external override { // ... other validation ... // ADD: Check the bridge's allocated emission budget. if (bridgeEmissionBudgets[msg.sender] < amount) { revert InsufficientEmissionBudget(); } // Decrease the budget after a successful mint. bridgeEmissionBudgets[msg.sender] -= amount; _mint(to, amount); } </pre> <ol style="list-style-type: none"> 1. Require Governance for Budget Allocation: The function to allocate an emission budget to a bridge should be controlled by a multi-signature wallet or a DAO governance process.

	<p>2. Consider Cross-Chain Vesting Contracts: As a defense-in-depth measure, for tokens minted via a bridge that are part of a vested allocation, consider sending them to a dedicated vesting contract on the destination chain instead of directly to a user's wallet. This ensures the vesting schedule is enforced regardless of the bridging mechanism.</p>
--	---

MEDIUM ISSUES

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

6.2.4 Predictable Rate Limiting Allows Burst Minting

Severity: MEDIUM

Status: FIXED

File(s) affected: SyndicateTokenCrosschain.sol

Update: <https://github.com/SyndicateProtocol/syndicate-appchains/pull/584/commits/d2c8ab52ad3bf4f5ab9dd7f7f86c5090f0151e8d>

<p>Attack / Description</p>	<p>The rate-limiting mechanism for the crosschain bridge is implemented with a fixed 24-hour reset cycle. This design, while functional, is predictable. It creates a recurring, publicly known window of time when the daily minting limits are reset for all bridges simultaneously. Sophisticated actors, such as MEV bots, can easily anticipate this exact moment to execute large "burst" transactions, consuming a disproportionate amount of the newly available capacity before other users can react.</p> <p>Risk Impact:</p> <p>This design choice introduces several risks that undermine the fairness and stability of the protocol:</p> <ul style="list-style-type: none"> • MEV Opportunities: It creates a profitable scenario for bots to front-run legitimate bridge transactions at the precise moment the rate limit resets. • Unfair Access: It gives technically advanced users with automated systems a significant advantage over regular users, centralizing access to a supposedly decentralized system.
------------------------------------	---



- **Network Congestion:** The predictable nature of the reset encourages a race, leading to periodic spikes in network traffic and gas fees.
- **Potential Market Manipulation:** Coordinated burst minting could be used to create sudden, short-term liquidity shocks on decentralized exchanges.

Proof of Concept:

We confirmed this vulnerability with a test case that simulates a coordinated burst minting attack.

Test Case:

```
function test_M01_PredictableRateLimitBurstMinting() public {
    // 1. Setup: A bridge mints its full daily limit.
    vm.prank(MALICIOUS_BRIDGE);
    crosschainToken.crosschainMint(USER, dailyLimit);

    // 2. Advance Time: We fast-forward the blockchain clock by exactly 24 hours
    // to the moment the limit is scheduled to reset.
    vm.warp(block.timestamp + 1 days);

    // 3. Exploit: The bridge immediately mints its full daily limit again,
    // executing a "burst" mint before any other transactions can occur.
    vm.prank(MALICIOUS_BRIDGE);
    crosschainToken.crosschainMint(USER, dailyLimit);

    // 4. Result: The test confirms the user has received double the daily limit
    // in what is effectively a single, coordinated action.
    assertTrue(userBalance == dailyLimit * 2, "Burst minting successful");
}
```

Test Output:

Burst Minting Amount: 2,000,000

	The test demonstrates that an attacker can successfully mint twice the daily limit by exploiting the predictable reset window.
Code	<p>Line 201 – 205 (SyndicateTokenCrosschain.sol):</p> <pre>// This logic creates a predictable "cliff" where the limit resets. if (block.timestamp >= config.lastMintTimestamp + 1 days) { config.currentMintUsed = 0; config.lastMintTimestamp = block.timestamp; }</pre>
Result/Recommendation	<p>To mitigate this issue, we recommend replacing the fixed-cycle reset with a more dynamic approach that eliminates the predictable "cliff."</p> <ol style="list-style-type: none"> 1. Implement a Sliding Window Rate Limiter: This is the industry-standard solution. A sliding window calculates usage over the preceding 24 hours on a rolling basis, removing the predictable reset event entirely. This ensures a smooth and fair application of rate limits. <p>Suggested Code:</p> <pre>// This example tracks usage on an hourly basis for a 24-hour sliding window. mapping(address => mapping(uint256 => uint256)) public hourlyUsage; function _validateAndUseMintLimit(address bridge, uint256 amount) internal { uint256 currentHour = block.timestamp / 1 hours; uint256 totalUsageIn24Hours = 0; // Sum usage over the past 24 hourly buckets. for (uint256 i = 0; i < 24; i++) {</pre>

```

        totalUsageIn24Hours += hourlyUsage[bridge][currentHour - i];
    }

    require(totalUsageIn24Hours + amount <= config.dailyMintLimit, "Exceeds sliding
window limit");

    // Record the new usage in the current hourly bucket.
    hourlyUsage[bridge][currentHour] += amount;
}

```

2. Consider Secondary Mitigations: While a sliding window is the preferred solution, other less effective but still beneficial options include adding a degree of randomization to reset timers or implementing an exponential backoff mechanism for addresses that frequently hit their limits.

6.2.5 Gas Griefing via Bridge Removal Function

Severity: MEDIUM

Status: FIXED

File(s) affected: SyndicateTokenCrosschain.sol

Update: <https://github.com/SyndicateProtocol/syndicate-appchains/pull/584/commits/43880dd52b937506b21c80d278f4d0fe9dd6716b>

Attack / Description

The removeBridge function is implemented with an inefficient design that iterates through the entire array of registered bridges to find and remove a specific entry. This results in a gas cost that scales linearly with the number of bridges in the system (a complexity known as $O(n)$). As the number of bridges grows, the gas required to execute this function can become prohibitively expensive, potentially exceeding the block gas limit. This creates a "gas griefing" vector, where an attacker could intentionally inflate the number of bridges to render the removal function unusable.

Risk Impact:



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

While this vulnerability does not lead to a direct loss of funds, it poses a significant operational and governance risk:

- **Denial of Service (DoS):** An attacker could add a large number of bridges (if permissions allow), making it impossible for administrators to remove a compromised or outdated bridge in an emergency.
- **Governance Disruption:** If a bridge needs to be removed via a governance vote, this vulnerability could prevent the successful execution of a passed proposal, effectively stalling governance.
- **System Maintainability:** As the protocol grows, this function will become increasingly costly and difficult to use, hindering routine maintenance.
- **Degraded Security Posture:** The inability to promptly respond to a security incident involving a faulty or malicious bridge significantly weakens the protocol's overall security.

Proof of Concept:

We confirmed this vulnerability with a test case that demonstrates how the gas cost increases as more bridges are added.

Test Case:

```
function test_M02_GasGriefingBridgeRemoval() public {
    // 1. Setup: 100 new bridges are added to the system to inflate the array size.
    for (uint i = 0; i < 100; i++) {
        maliciousBridges[i] = address(uint160(0x1000 + i));
        crosschainToken.setBridgeLimits(maliciousBridges[i], 1000 * 10**18, 1000 *
10**18);
    }

    // 2. Exploit: We measure the gas consumed to remove a single bridge
    // from the now-large array.
    uint256 gasStart = gasleft();
    vm.prank(BRIDGE_MANAGER);
    crosschainToken.removeBridge(maliciousBridges[0]);
}
```



	<pre>uint256 gasUsed = gasStart - gasleft(); // 3. Result: The test confirms that the gas used is significant, // proving that the cost scales with the number of bridges. assertTrue(gasUsed > 5000, "Gas cost increases with bridge count"); }</pre> <p>Test Output:</p> <p>Gas Used for Bridge Removal: 15687 Bridges Remaining: 99</p> <p>The test shows a notable gas cost for removal, which would continue to increase and eventually become un-executable as more bridges are added.</p>
Code	<p>Line 165 – 183 (SyndicateTokenCrosschain.sol):</p> <pre>function removeBridge(address bridge) external onlyRole(BRIDGE_MANAGER_ROLE) { // ... validation ... // INEFFICIENT: This loop's gas cost grows with the size of the 'bridges' array. for (uint256 i = 0; i < bridges.length; i++) { if (bridges[i] == bridge) { bridges[i] = bridges[bridges.length - 1]; // Swap with the last element bridges.pop(); // Remove the last element break; } } emit BridgeRemoved(bridge); }</pre>

Result/Recommendation

To resolve this issue, the bridge storage mechanism should be refactored to allow for efficient, constant-time removal operations.

1. **Use an Efficient Data Structure:** The industry-standard solution is to use a data structure that combines a mapping and an array to achieve $O(1)$ complexity for additions, removals, and lookups. OpenZeppelin's EnumerableSet library is a perfect, battle-tested implementation for this.

Suggested Code:

```
import "@openzeppelin/contracts/utils/structs/EnumerableSet.sol";

using EnumerableSet for EnumerableSet.AddressSet;
EnumerableSet.AddressSet private bridgeSet;

// This new function removes a bridge in constant time, regardless of the set size.
function removeBridge(address bridge) external onlyRole(BRIDGE_MANAGER_ROLE) {
    // O(1) removal operation
    if (!bridgeSet.remove(bridge)) revert BridgeNotFound();

    delete bridgeConfigs[bridge];
    emit BridgeRemoved(bridge);
}
```

2. **Limit the Number of Bridges:** As a secondary, defense-in-depth measure, consider adding a protocol-wide limit on the maximum number of bridges that can be registered at any one time to prevent abuse.

6.2.6 Transfer Lock Period Concerns

Severity: MEDIUM

Status: FIXED

File(s) affected: SyndicateToken.sol

Update: <https://github.com/SyndicateProtocol/syndicate-appchains/pull/584/commits/a8eb1e30ad1815e62f79f2cffd8801225ab61ed>

Attack / Description

While the base SyndicateToken contract correctly prevents most transfers when a lock is active, the crosschainMint function does not respect this state. This allows new tokens to be minted via a bridge and distributed to users even when the protocol is intended to be in a restricted transfer phase, such as immediately following a token generation event or an airdrop.

Risk Impact:

This issue does not allow for illicit token transfers but introduces several operational and security risks by creating inconsistent behavior:

- **Security Mechanism Concerns:** It creates a loophole in the transfer lock's security model. If the lock is intended to prevent any change in user balances (outside of approved airdrops), this bypass undermines that goal.
- **Airdrop Security:** It could allow for unintended token distribution during a post-airdrop lock period, potentially leading to market conditions the lock was designed to prevent.
- **Consistency Issues:** The differing behavior between the base token and the crosschain token can lead to confusion and unexpected outcomes for both users and protocol administrators.

Proof of Concept:

We confirmed this inconsistency with a test case that shows tokens can be successfully minted via a bridge while the transfer lock is active.



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

Test Case:

```
function test_M03_TransferLockBypassViaCrosschainMint() public {
    // 1. Setup: The transfer lock is activated.
    vm.prank(ADMIN);
    crosschainToken.setUnlockTimestamp(block.timestamp + 30 days);
    assertTrue(crosschainToken.transfersLocked(), "Transfers should be locked");

    // 2. Action: A crosschain mint operation is initiated.
    vm.prank(MALICIOUS_BRIDGE);
    crosschainToken.crosschainMint(USER, 100_000 * 10**18);

    // 3. Result: The test confirms the user's balance increased, proving
    // that minting occurred despite the active transfer lock.
    // Note: Standard transfers would still be correctly blocked.
    assertTrue(userBalance == 100_000 * 10**18, "Tokens minted during lock");
}
```

Test Output:

Crosschain mint works during transfer lock period but transfers are blocked

This test verifies that the crosschainMint function does not adhere to the protocol's global lock state.

Code

Line 275 – 285 (SyndicateToken.sol):

The base token correctly enforces the transfer lock within its internal `_update` function, which is called by `transfer` and `transferFrom`. It explicitly allows minting (`from == address(0)`) but blocks standard peer-to-peer transfers. However, the `crosschainMint` function, being a separate entry point, does not pass through this logic.

	<pre> function _update(address from, address to, uint256 value) internal virtual override(ERC20, ERC20Votes) { // This correctly blocks transfers (when 'from' is not the zero address) during a lock. if (from != address(0) && to != address(0)) { if (transfersLocked() && !hasRole(AIRDROP_MANAGER_ROLE, from)) { revert TransfersLocked(); } } super._update(from, to, value); } </pre>
Result/Recommendation	<p>The core of this issue is a business logic decision. The protocol team must decide whether crosschain minting should be permitted during a transfer lock period.</p> <ol style="list-style-type: none"> 1. Enforce Lock-Aware Minting (Recommended): If the intent is for the lock to be a global state, we recommend making the crosschainMint function "lock-aware." This would involve adding a check for the transfersLocked() status. <p>Suggested Code:</p> <pre> function crosschainMint(address to, uint256 amount) external override { // ... existing validation ... // ADD: Check if minting should respect transfer locks. // This check could be made more granular, for example, by allowing // a specific role to bypass it. if (transfersLocked()) { revert CannotMintDuringTransferLock(); } _mint(to, amount); } </pre>

	<p>2. Document the Intended Behavior: If the current behavior is intended, this must be explicitly and clearly documented in the technical specifications to avoid confusion for developers, auditors, and administrators.</p> <p>3. Add Governance Controls: Consider adding a separate, governable parameter that can enable or disable crosschain minting during a lock period, providing more flexibility for administrators.</p>
--	---

LOW ISSUES

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

6.2.7 Bridge Limit Bypass via Reconfiguration

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: SyndicateTokenCrosschain.sol

Attack / Description	<p>This flaw enables the daily rate limit for a bridge to be improperly reset.. When a BRIDGE_MANAGER_ROLE holder calls this function to reconfigure a bridge's limits, the function completely resets the currentMintUsed counter to zero. This allows a manager to effectively bypass the daily rate limit by simply re-applying the same configuration to a bridge after its limit has been exhausted for the day.</p> <p>Risk Impact:</p> <p>This vulnerability undermines a key security mechanism of the protocol, introducing the following risks:</p> <ul style="list-style-type: none"> • Rate Limit Bypass: The intended daily minting and burning caps can be easily circumvented, defeating the purpose of having rate limits in the first place.
-----------------------------	--



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

- **Unintended Token Flow:** It allows for a greater volume of tokens to be minted or burned within a 24-hour period than the protocol's tokenomics and security model intend.
- **Trust Violation:** This flaw erodes confidence in the rate-limiting system as a reliable security measure to prevent large, sudden token movements.

Proof of Concept:

We have validated this vulnerability with a test case that demonstrates the bypass mechanism.

Test Case:

```
function test_M04_BridgeLimitBypassReconfiguration() public {
    // 1. Setup: A bridge mints tokens up to its daily limit.
    vm.prank(MALICIOUS_BRIDGE);
    crosschainToken.crosschainMint(USER, initialLimit);

    // 2. Exploit: The bridge manager calls setBridgeLimits again for the same
    bridge.
    // This action resets the bridge's daily usage counter to zero.
    vm.prank(BRIDGE_MANAGER);
    crosschainToken.setBridgeLimits(MALICIOUS_BRIDGE, initialLimit, initialLimit);

    // 3. Action: The bridge can now immediately mint up to its daily limit again.
    vm.prank(MALICIOUS_BRIDGE);
    crosschainToken.crosschainMint(USER, initialLimit);

    // 4. Result: The test confirms the user has received double the intended daily
    // limit, proving the rate limit was successfully bypassed.
    assertTrue(userBalance == initialLimit * 2, "Limit bypassed");
}
```

Test Output:

Limit bypass via reconfiguration

	The test confirms that re-setting the limits effectively grants the bridge a fresh daily quota, allowing it to exceed its intended rate limit.
Code	<p>Line 138 - 146 (SyndicateTokenCrosschain.sol):</p> <pre>// When this function is called, it re-initializes the entire config. bridgeConfigs[bridge] = BridgeConfig({ dailyMintLimit: dailyMintLimit, dailyBurnLimit: dailyBurnLimit, lastMintTimestamp: block.timestamp, lastBurnTimestamp: block.timestamp, currentMintUsed: 0, // VULNERABILITY: This resets the daily usage! currentBurnUsed: 0, // VULNERABILITY: This resets the daily usage! isActive: true });</pre>
Result/Recommendation	<p>To fix this vulnerability, the setBridgeLimits function must be modified to preserve the existing daily usage when updating a bridge's configuration.</p> <ol style="list-style-type: none"> 1. Preserve Usage Counters: The function should read the existing BridgeConfig from storage, modify only the necessary fields, and write it back, ensuring the currentMintUsed and currentBurnUsed values are not reset. <p>Suggested Code:</p> <pre>function setBridgeLimits(address bridge, uint256 dailyMintLimit, uint256 dailyBurnLimit) external override onlyRole (BRIDGE_MANAGER_ROLE)</pre>

```

{
    // Load the existing config into storage to modify it.
    BridgeConfig storage config = bridgeConfigs[bridge];

    // Preserve the existing usage for the day.
    uint256 preservedMintUsed = config.currentMintUsed;

    // Update only the limit values.
    config.dailyMintLimit = dailyMintLimit;
    config.dailyBurnLimit = dailyBurnLimit;

    // If the new limit is lower than what's already been used,
    // you might cap the usage at the new limit. Otherwise, keep the old usage.
    if (config.dailyMintLimit < preservedMintUsed) {
        config.currentMintUsed = config.dailyMintLimit;
    } else {
        config.currentMintUsed = preservedMintUsed;
    }

    // Apply similar logic for the burn limit...
}

```

2. **Add a Configuration Cooldown:** To prevent abuse, consider implementing a cooldown period that prevents the limits for a single bridge from being reconfigured more than once within a specific timeframe (e.g., once every 24 hours).

6.2.8 Missing Input Validation on Bridge Addresses

Severity: LOW

Status: FIXED

File(s) affected: SyndicateTokenCrosschain.sol

Update: <https://github.com/SyndicateProtocol/syndicate-appchains/commit/67515fe08a0b2c13c96dc03d9b6bd16f661ae621>



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

Attack / Description

The `setBridgeLimits` function lacks comprehensive input validation for the bridge address parameter. While it correctly checks for a zero address, it fails to verify that the address being added is a smart contract. This oversight allows an administrator to accidentally or maliciously set an Externally Owned Account (EOA), a non-contract address, or even an incorrect contract address as an authorized bridge.

Risk Impact:

This is a low-severity issue that does not lead to a direct loss of funds but indicates a lack of defensive programming and introduces several operational risks:

- **Operational Errors:** An administrator could make a mistake during a manual configuration, leading to a non-functional bridge that is difficult to debug.
- **Wasted Gas:** Transactions sent to a bridge that is an EOA will not execute any logic and will result in wasted gas for both the protocol and its users.
- **User Confusion:** A misconfigured bridge can lead to unexpected behavior and failed transactions, eroding user confidence and creating unnecessary support overhead.

Proof of Concept:

We confirmed this vulnerability with a test case that successfully adds a non-contract EOA as an authorized bridge.

Test Case:

```
function test_L01_MissingInputValidation() public {  
    // 1. Setup: Define an arbitrary EOA address.  
    address eoa = address(0x123);  
  
    // 2. Action: The BRIDGE_MANAGER calls setBridgeLimits, passing the EOA  
    address.  
    vm.prank(BRIDGE_MANAGER);
```



	<pre>crosschainToken.setBridgeLimits(eoa, 1000 * 10**18, 1000 * 10**18); // 3. Result: The test confirms that the EOA was successfully added // as an authorized bridge, which should not be possible. assertTrue(crosschainToken.isBridgeAuthorized(eoa), "EOA added as bridge"); }</pre> <p>Test Output:</p> <p>EOA added as bridge successfully</p> <p>The test demonstrates that the system incorrectly accepts a non-contract address as a valid bridge.</p>
Code	<p>Line 123 - 149 (SyndicateTokenCrosschain.sol,):</p> <pre>function setBridgeLimits(address bridge, uint256 dailyMintLimit, uint256 dailyBurnLimit) external override onlyRole (BRIDGE_MANAGER_ROLE) { if (bridge == address(0)) revert ZeroAddress(); // Basic check exists. // MISSING: Validation to ensure the 'bridge' address is a contract. // MISSING: Validation for reasonable limit values. }</pre>
Result/Recommendation	<p>To improve the robustness and safety of the protocol, we recommend adding comprehensive input validation to the setBridgeLimits function.</p>

1. **Implement Essential Validation Checks:** The function should be updated to include checks that ensure the address is a contract and that the limits are reasonable.

Suggested Code:

```
function setBridgeLimits(address bridge, uint256 dailyMintLimit, uint256
dailyBurnLimit)
    external
    override
    onlyRole (BRIDGE_MANAGER_ROLE)
{
    if (bridge == address(0)) revert ZeroAddress();

    // ADD: Verify that the bridge address has code, meaning it's a contract.
    if (bridge.code.length == 0) revert BridgeMustBeContract();

    // ADD: Verify that the limits set are not unreasonably high.
    if (dailyMintLimit > TOTAL_SUPPLY) revert UnreasonableMintLimit();
    if (dailyBurnLimit > TOTAL_SUPPLY) revert UnreasonableBurnLimit();

    // ... rest of function
}
```

2. **Consider a Bridge Registry:** For a higher level of security, implement a registry of approved bridge implementation addresses. The setBridgeLimits function could then verify that the new bridge's implementation contract is on this approved list.
3. **Add Interface Checks:** Verify that the target contract supports the required bridge interface (e.g., via ERC-165). This ensures that the added contract has the necessary functions to operate as a bridge.



INFORMATIONAL ISSUES

During the audit, softstack's experts found **one Informational issue** in the code of the smart contract

6.2.9 Divergence from Documentation: Bridge Config & Factory Removal

Severity: INFORMATIONAL

Status: FIXED

File(s) affected: synd-contracts/src/token/SYND_TOKEN_ARCHITECTURE.md

Update: <https://github.com/SyndicateProtocol/syndicate-appchains/pull/584/commits/86414679035b37d744f2aa25c80a8051acb9923e>

Attack / Description	<p>We observed two points where the implementation diverged slightly from the documentation. In both cases, these represent enhancements to the system's flexibility and were accurately explained.</p> <ol style="list-style-type: none">1. Bridge Target Addresses: The documentation references specific bridge addresses for Arbitrum and Optimism. However, the implementation correctly makes these addresses configurable deployment parameters passed to bridge proxy constructors, rather than hardcoded constants. This is a significant improvement, as it allows for greater adaptability across different networks (mainnet, testnet, etc.) without requiring code changes.2. Factory Contracts: The documentation correctly notes that factory contracts were initially considered but later removed to manage contract size limits. Our review confirms that direct deployment scripts are used instead, aligning the implementation with this documented design decision.
Code	NA
Result/Recommendation	These changes enhance system adaptability and reduce the risk of deployment errors across



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

environments. Documentation could optionally be updated.



hello@softstack.io
www.softstack.io


softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

6.3 Verify Claims


6.3.1 Role-Based Emission Control

Only the designated EmissionScheduler should be able to mint new SYND tokens via EMISSION_MINTER_ROLE. Epoch timing and token amounts must be enforced with no early or excess minting possible.

Status: tested and verified 


6.3.2 Crosschain Mint/Burn Authorization

Only explicitly authorized bridges may invoke crosschainMint or crosschainBurn. The audit must validate rate limit tracking, role isolation, and abuse resistance under high volume.

Status: tested and verified 


6.3.3 Deterministic CREATE3 Deployment

Contracts must be deployed using the same salt and logic on all chains, resulting in consistent addresses. Address prediction and deployment validation logic must be accurate and collision-resistant.

Status: tested and verified 


6.3.4 Bridge Proxy Safety & ETH Handling

Bridge proxies (Arbitrum, Optimism) must enforce per-transaction and daily limits, correctly manage ETH for gas costs, and expose no reentrancy or DoS vectors during bridging operations.

Status: tested and verified 

6.3.5 Emergency Pause & Recovery Readiness

Pause mechanisms must halt emission or bridging flows without corrupting contract state. Only authorized roles should be able to pause/unpause, with no single point of failure.

Status: tested and verified 



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

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 9 issue, classified as follows:

- zero critical issue were found.
- 3 high severity issues were found.
- 3 medium severity issues were found.
- 2 low severity issues were discovered
- 1 informational issues were identified

The audit report provides detailed descriptions of each identified issue, including severity levels, proof of concepts and recommendations for mitigation. We recommend the Syndicate team to review the suggestions.

Update (08.07.2025): The Syndicate team has successfully addressed all identified issues from the audit. All vulnerabilities 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 critical concerns remain.



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984

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>

How We Work



1 -----

PREPARATION

Supply our team with audit ready code and additional materials



2 -----

COMMUNICATION

We setup a real-time communication tool of your choice or communicate via e-mails.



3 -----

AUDIT

We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.



4 -----

FIXES

Your development team applies fixes while consulting with our auditors on their safety.



5 -----

REPORT

We check the applied fixes and deliver a full report on all steps done.



hello@softstack.io
www.softstack.io

softstack GmbH
Schiffbrückstraße 8
24937 Flensburg

CRN: HRB 12635 FL
VAT: DE317625984