GA GUARDIAN

Synthetix

TLX v2

Security Assessment

March 1st, 2025

Summary

Audit Firm Guardian

Prepared By Roberto Reigada, Zdravko Hristov, Raj Kumar,

Wafflemakr, Cosine, Michael Lett

Client Firm Synthetix

Final Report Date March 1, 2025

Audit Summary

Synthetix engaged Guardian to review the security of their Leveraged token built on top of Synthetix V2. From the 3rd of February to the 26th of February, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 4 High/Critical issues were uncovered and promptly remediated by the Synthetix team.

Security Recommendation Given the number of High and Critical issues detected as well as additional code changes made after the main review, Guardian recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Optimism**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

Table of Contents

Project Overview

Project Summary

Project Name	Synthetix
Language	Solidity
Codebase	https://github.com/Synthetixio/leverage-token-v1.2
Commit(s)	Initial commit: f295293551c1ff9043d6d9a9df7e63cd26729485 Final commit: edac32cf0c6cd62160d45bac5c2f9d43bf173cf8

Audit Summary

Delivery Date	March 1, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	0	0	0	0	1
High	3	0	0	1	0	2
Medium	15	0	0	10	1	4
Low	46	0	0	31	1	14

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian's review of Synthetix, fuzz-testing with Foundry was performed on the protocol's main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared Foundry fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
GLOB-01	Exchange rate should never be zero	✅	✅	✅	10M+
GLOB-02	Total value should match remaining margin	✅	✅	✅	10M+
GLOB-03	Base asset approval for Odos router should always be max	✅	✅	✅	10M+
TLX-01	Notional value should match leverage * remaining margin (with tolerance)	✅	✅	✅	10M+
TLX-02	Streaming fee timestamp should be consistent	✅	✅	✅	10M+
TLX-03	Input validation should prevent zero amounts	✅	✅	✅	10M+
TLX-04	Leveraged token validation should be consistent	✅	✅	✅	10M+
TLX-05	After a successful mintFor call the callers sUSD balance should decrease	✅	✅	✅	10M+
TLX-06	After a successful redeemFor call the callers sUSD balance should increase	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
TLX-07	After a successful redeemFor call the callers LT balance should decrease	✅	✅	✅	10M+
TLX-08	When a user calls mintFor they should receive LTs worth less than the deposited amount	✅	✅	✅	10M+
TLX-09	When a user calls redeemFor they should receive less sUSD than the given LT amt was worth	✅	✅	✅	10M+
TLX-10	After a successful mintFor call the callers LT balance should increase	✅	✅	✅	10M+
TLX-11	After a successful mintWithEth call the callers ETH balance should decrease	✅	✅	✅	10M+
TLX-12	LeveragedToken contract should never hold base Assets after user interactions	✅	✅	✅	10M+
TLX-13	ZapSwap contract should never hold any assets after user interactions	✅	✅	✅	10M+
TLX-14	Successful delayed orders execution should result in leverage close to target	✅	❌	N/A	10M+
TLX-15	Should have pending leverage update if deviation exceeds threshold	✅	✅	✅	10M+
TLX-16	If mintedTimestamp[user] != 0 and block.timestamp - mintedTimestamp[user] >= 300, then decayingRedemptionFee(user, ltAmount, exchangeRate_) == 0	✅	❌	✅	10M+
TLX-17	Streaming fee should be charged on first redeem	✅	❌	N/A	10M+
TLX-18	Minted timestamp should persist on transfer	✅	❌	❌	10M+
TLX-19	Streaming fee charged during redemption should match expected calculation	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Inflation Exploit	Logical Error	Critical	Resolved
H-01	Chainlink's Transmit Call Can Force A LT Position Into Liquidation	Validation	High	Resolved
H-02	Users Will Always Pay The Max. Decaying Redemption Fee	Logical Error	High	Resolved
H-03	Rebalance Extractable Value	Sandwich Attack	High	Acknowledged
M-01	Last User Cannot Fully Redeem If LT Position Is Still Open	Validation	Medium	Acknowledged
M-02	Redeemer Avoids Paying The Last Interval Of The Streaming Fee	Logical Error	Medium	Resolved
M-03	Decaying Redemption Fee Manipulation	Logical Error	Medium	Acknowledged
M-04	Leverage Mismatch Because Different Price Sources	Logical Error	Medium	Acknowledged
M-05	Final Redeemer Pays Decaying Fees And Slippage	Logical Error	Medium	Resolved
M-06	Missing Execute Order Function	Logical Error	Medium	Acknowledged
M-07	Depositing Minimum Amount Leads To Position Closure	Logical Error	Medium	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
M-08	Order Fees Do Not Include Rebalance Costs	Logical Error	Medium	Acknowledged
M-09	Decaying Redemption Fee Can Be Bypassed	Logical Error	Medium	Resolved
M-10	Incorrect Calculation Of currentSize	Logical Error	Medium	Resolved
M-11	Insufficient OdosRouter Calldata Validation	Validation	Medium	Partially Resolved
M-12	Tokens Are Tradable When The LT Is Liquidatable	Validation	Medium	Acknowledged
M-13	Only Current Mint Amount Validated	Logical Error	Medium	Acknowledged
M-14	LeveragedTokens With A Higher TargetLeverage Pay Disproportionately Higher Streaming Fees	Protocol Design	Medium	Acknowledged
M-15	Streaming Fee Starts Too Late	Logical Error	Medium	Acknowledged
L-01	_validateMintAmount May Over/Underestimate Position Size	Validation	Low	Acknowledged
L-02	Possible Full Position DoS Due To Streaming Fee	Validation	Low	Acknowledged
L-03	Unused Functions	Code Best Practices	Low	Partially Resolved
L-04	Referral System Is Not Implemented	Code Best Practices	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-05	Pyth Price Confidence Interval Is Ignored In PerpsV2	Validation	Low	Acknowledged
L-06	LTs Could Be Forced To Remain Outside Exact Target Leverage	Validation	Low	Acknowledged
L-07	Temporary DoS Due To Price Divergence	DoS	Low	Acknowledged
L-08	Incorrect Order acceptedPrice Calculation	Logical Error	Low	Resolved
L-09	LeveragedToken Can Be Reactivated	Validation	Low	Acknowledged
L-10	Charging Wrong Redemption Fee	Logical Error	Low	Acknowledged
L-11	Wrong Comparison Operators In _closePosition And _submitLeverageUpdate	Logical Error	Low	Resolved
L-12	mintedTimestamp Is Not Recorded When The BaseAmount Equals The decayingRedemptionFeeMinBaseAmount	Logical Error	Low	Resolved
L-13	Redemption Fee Bounds	Validation	Low	Acknowledged
L-14	Unnecessary notionalValue Computation	Code Best Practices	Low	Resolved
L-15	DOS For USDT In ZapSwap	Logical Error	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-16	ZapSwap Does Not Use TlxOwnable onlyOwner Modifier	Code Best Practices	Low	Resolved
L-17	Wrong Comment	Code Best Practices	Low	Resolved
L-18	Unused Rebalance Fee	Code Best Practices	Low	Resolved
L-19	isActive() Semantics	Protocol Design	Low	Acknowledged
L-20	Lack Of Storage Gaps In TlxOwnableUpgradeable	Code Best Practices	Low	Acknowledged
L-21	Wrong Emission In Rebalanced Event	Code Best Practices	Low	Resolved
L-22	Unsafe Transfer Of Ownership In The AddressProvider	Code Best Practices	Low	Resolved
L-23	Wrong Emission Of MintedAmountIncreased Event	Code Best Practices	Low	Acknowledged
L-24	Users Pay Slippage If There Is No Position	Logical Error	Low	Resolved
L-25	Precision Loss In _getLeverageUpdateSizeDelta Function	Logical Error	Low	Acknowledged
L-26	Fee Calculations Are Rounded Down	Logical Error	Low	Acknowledged
L-27	Users Can Redeem While The Contract Is Paused	Code Best Practices	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-28	High Rebalance Threshold Causes Funds To Become Locked	Validation	Low	Acknowledged
L-29	Adresses Can Not Be Unfrozen In AddressProvider Contract	Code Best Practices	Low	Acknowledged
L-30	DoS On High Usage	Protocol Design	Low	Acknowledged
L-31	Missing Force Rebalance Functionality	DoS	Low	Acknowledged
L-32	Vault Could Be Drained By Cancellation Fees	Logical Error	Low	Acknowledged
L-33	Missing Check If targetLeverage >= 1e18	Logical Error	Low	Acknowledged
L-34	Centralization Risk	Validation	Low	Acknowledged
L-35	Early Return computePriceImpact	Code Best Practices	Low	Resolved
L-36	_redeemLeveragedToken Return Value Not Used	Code Best Practices	Low	Acknowledged
L-37	_addressProvider Initialized Twice	Code Best Practices	Low	Acknowledged
L-38	Misleading Function Param Name	Code Best Practices	Low	Resolved
L-39	Missing Minimum Amount Checks	Validation	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-40	MINIMUM_MINT_AMOUNT Not Initialized	Code Best Practices	Low	Resolved
L-41	Unnecessary Price Impact Charged	Logical Error	Low	Acknowledged
L-42	Freezing Address Not Yet Set	Validation	Low	Acknowledged
L-43	Remaining Margin Can Be 0	Validation	Low	Acknowledged
L-44	canRebalance Does Not Check Market Limits	Validation	Low	Acknowledged
L-45	Zero Returns On Stale Prices	Logical Error	Low	Acknowledged
L-46	Delayed Offchain Order May Become Unexecutable	Validation	Low	Acknowledged

C-01 | Inflation Exploit

Category	Severity	Location	Status
Logical Error	● Critical	LeveragedToken.sol: 245	Resolved

Description PoC

When the `LeveragedToken` contract has no existing supply (right after its deployment) a new minter operates on an initial exchange rate of 1 for the first deposit. Under normal conditions, once tokens are minted, a user must call `redeemFor` to withdraw margin.

However, the contract also provides a raw `burn` function that simply destroys the user's leveraged tokens without adjusting or withdrawing margin from the underlying Synthetix position.

A malicious first minter can exploit this by:

• Minting a large amount: Suppose the user deposits 100000 sUSD into a brand-new `LeveragedToken`. He receives 100000 leveraged tokens (exchange rate = 1).

• Burning all the leveraged tokens except one: The user calls `burn(leverageTokenBalance - 1)`. This reduces the `totalSupply` drastically (e.g., from 100000 down to 1), but the contract margin remains the same (it does not call `_withdrawMargin`).

• As a result, `exchangeRate` inflates sharply (`totalValue` remains 100000, but `totalSupply` is only 1).

• This single remaining token now has a massively increased claim on the `LeveragedToken`'s margin.

• If a new user attempts to mint with an amount lower than the malicious user's original deposit (e.g., just 10000 sUSD), the exchange rate calculates the new minted tokens at the already-inflated ratio. The first minter's single token "absorbs" the new deposit, growing their share of the margin.

The new minter performs the sUSD deposit but does not receive any leveraged token (`totalSupply` remains 1). This leads to a scenario where the malicious user can redeem their one leveraged token later for a higher amount than the original 100000 sUSD at the expense of other depositors.

• In a situation where future users set slippage, their deposits will always revert if its lower than the initial 100000 sUSD as, due to rounding, they would receive 0 leveraged tokens in exchange for their sUSD.

Recommendation

Consider removing the public `burn` function from the `LeveragedToken` contract.

Resolution

Synthetix Team: The issue was resolved in PR#6.

H-01 | Chainlink's Transmit Call Can Force A LT Position Into Liquidation

Category	Severity	Location	Status
Validation	High	LeveragedToken.sol	Resolved

Description

A malicious user can exploit the fact that the initial `transferMargin` call performed during redemptions to subtract margin from the position, could use a partially stale Chainlink Price Feed. By carefully selecting the withdrawal amount (`marginDelta`), the attacker appears to keep the position above liquidation threshold at the stale price, but once the price feed is updated with the actual current price, the margin ends up below the threshold making it liquidatable.

This exploit could be relatively easy to execute as:
• Chainlink updates the price under two circumstances: When the "heartbeat" time passes (this is one hour for most of the feeds) and if the price changes by more than the deviation threshold which is usually a value between 0.1% and 0.5% (https://data.chain.link/feeds). Therefore it should not be very uncommon to find a Chainlink price feed that deviates 0.4% from the current price.
• The `LeveragedToken` contract allows pulling as much margin as Synthetix PerpsV2 does, or which is the same, as much as the resulting margin would not be lower than the liquidation margin or min. initial margin (`liqMargin` + `liqPremium`).

Given these conditions, a malicious user could:
• Perform a large deposit/mint: The attacker first deposits a large amount of sUSD margin into the `LeveragedToken` contract (which also decreases the overall leverage temporarily). The `LeveragedToken` eventually rebalances to the desired leverage ratio.
• The attacker calls `redeemFor` with an off chain delayed order referencing a stale aggregator price feed. This call is executed right before the Chainlink Price Feed is updated, front-running the aggregator transmit call.
• Under the stale Chainlink Price Feed the position appears to remain safely above `liquidationMargin` + premium.
• A rebalance order is created however, before it is executed, as the Chainlink Price Feed was just updated to the new price, a user calls `PerpsV2MarketLiquidate.flagPosition`. The `LeveragedToken` position will be flagged and the only operation enabled will be a liquidation. The previous delayed order was canceled during the flagging process and can not be executed anymore.
• `LeveragedToken` 's position is liquidated.

The attacker could pocket the flagger fee (between 2$ and 1000$) from the liquidation process. This can yield a net profit for the attacker if the liquidation fee surpasses whatever leveraged tokens value they still held.

Recommendation

Upon redeeming, the `LeveragedToken` contract should require a safety buffer that ensures the liquidation price is significantly (e.g., 10%) below the current aggregator price for longs and higher than the aggregator price for shorts.

Concretely:
• Compute the user's requested redemption.
• Simulate the new margin's "post-close liquidation price."
• Require that `liquidationPrice < (currentPrice × (1 - minBuffer))`. For example, if `minBuffer` = 10%, then `liquidationPrice` < 90% of the aggregator price.

This ensures that even if the aggregator price feed is off by a small fraction (like 0.5% or 1%), the leftover margin won't be driven immediately below the liquidation threshold when the price feed is updated with the newest price. By implementing this buffer, the system blocks partial redemptions that leave no margin for slippage or stale feed differences, thus mitigating the exploit.

Resolution

Synthetix Team: The issue was resolved in PR#10.

H-02 | Users Will Always Pay The Max. Decaying Redemption Fee

Category	Severity	Location	Status
Logical Error	High	LeveragedToken.sol: 264	Resolved

Description

In the Config.sol, the protocol sets:

```
uint256 public constant DECAYING_REDEMPTION_FEE_DURATION = 300e18; // 300 seconds, 5 minutes
```

However, the redemption fee logic in the contract does:

```
uint256 timePassed = block.timestamp - mintedTimestamp[user];
uint256 percentPassed = timePassed.div(redemptionFeeDuration);
```

If redemptionFeeDuration is stored as 300e18, then:

• timePassed is a normal integer in seconds (e.g., 150 for half the interval).

• div is a scaled integer division.

• The result becomes 150 ÷ (300 × 10^18) = 0.5 → 0.

Hence, the code incorrectly sees “0% of the duration has passed,” rather than 50%. This breaks the decay logic and always calculates a near-maximum extra fee.

Recommendation

Consider setting DECAYING_REDEMPTION_FEE_DURATION to 300 instead of 300e18 in the Config contract.

Resolution

Synthetix Team: The issue was resolved in PR#9.

H-03 | Rebalance Extractable Value

Category	Severity	Location	Status
Sandwich Attack	High	LeveragedToken.sol	Acknowledged

Description

In the _submitLeverageUpdate function the acceptablePrice value is determined by applying a standard slippage amount to the result of the fillPrice.

However the result of the fillPrice function itself can be manipulated such that it returns a higher fillPrice and thus allows for significant extractable value by sandwiching the TLX vault's order.

Consider the following scenario:

• A malicious actor observes that a significant amount of PnL has built up for the TLX vault and that a rebalance will be triggered by even a small deposit.

• The malicious actor creates a large long order to push the skew of the market higher.

• The malicious actor triggers a small deposit with the mintFor function, triggering a rebalance.

• The rebalance order is assigned a high acceptablePrice which can be significantly more than the fair market value of the index asset due to the inflated price impact.

• The malicious actor subsequently closes their position directly after the rebalance order is executed, receiving positive impact at the expense of the leveraged token vault holders.

Recommendation

There is no trivial fix. One potential approach is to reduce the single large rebalance into multiple partial rebalances, minimizing the window for exploit.

Resolution

Synthetix Team: Acknowledged.

M-01 | Last User Cannot Fully Redeem If LT Position Is Still Open

Category	Severity	Location	Status
Validation	Medium	LeveragedToken.sol	Acknowledged

Description

In the LeveragedToken contract, when a user calls redeemFor to withdraw all the remaining margin, Synthetix will revert because removing that margin, IPerpsV2MarketConsolidated(marketAddress).transferMargin(-int256(amount)), would cause the position to be liquidatable.

Synthetix enforces that a margin withdrawal can not put the account under the liquidation threshold. The last user would be unable to fully close out his position in a single transaction. He would have to do multiple smaller partial redemptions to avoid the liquidation check.

First, the user would have to call redeemFor (possibly multiple times) until the LeveragedToken's margin would fall below MINIMUM_MARGIN_BALANCE so a delayed close position order would be created. Then, wait for the execution of the delayed order that closes the position.

And finally call redeemFor with the remaining amount of leveraged tokens. All these steps, trusting that no other user would front-run his final redeemFor call reopening the LeveragedToken's position.

Recommendation

If the user is redeeming all the remaining tokens, forcibly do a submitCloseOffchainDelayedOrderWithTracking instead of a partial margin withdrawal that triggers liquidation checks.

Resolution

Synthetix Team: Acknowledged.

M-02 | Redeemer Avoids Paying The Streaming Fee

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol: 165	Resolved

Description PoC

Within the LeveragedToken's redeemFor flow, the contract calculates:

• baseWithdrawn = (leveragedTokenAmount * exchangeRate) - decayingRedemptionFee - slippage.

• Then it calls _withdrawMargin(baseWithdrawn + streamingFee).

• Lastly, it charges the streaming fee (_chargeStreamingFee(streamingFee)) out of the contract's Synthetix margin.

However, the user's baseWithdrawn portion is based on an exchange rate computed before the streaming fee is removed from margin and consequently the user does not pay his pro-rata share of that streaming fee.

All holders end up paying the streaming fee out of the leftover margin collectively, while the redeemer takes out margin as if no fee had been deducted.

By computing baseWithdrawn from the pre-fee exchangeRate, the user is granted a higher share. The streaming fee is then subtracted from the contract's margin but not from the user's final redemption proceeds.

Recommendation

Consider reducing the the user's baseWithdrawn by their portion of the streaming fee. For example, if the user holds X% of the total supply, they pay X% of the streaming fee in the redemption step.

On the other hand, consider also implementing a function that is called frequently to charge the streaming fee manually.

Resolution

Synthetix Team: The issue was resolved in PR#7.

M-03 | Decaying Redemption Fee Manipulation

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol: 685	Acknowledged

Description

In the leveraged token system a decaying redemption fee is applied to users who have recently deposited. Throughout the codebase the decaying redemption fee is assigned to start off at a 1% fee and decay to 0% over the course of 5 minutes.

The minimum deposit for a user which will reset the timer for the decaying redemption fee is set to 5 USD in the `Config` contract as the `DECAYING_REDEMPTION_MIN_BASE_AMOUNT` value.

With these configured parameters it can be significantly profitable for one vault depositor to do a small deposit on behalf of another depositor who is about to redeem and cause them to experience a significant decay fee.

The malicious vault depositor in this case (and the rest of `LeveragedToken` holders) would gain from the significant fee paid by the victim depositor in this case.

On networks without a public mempool specifically frontrunning a user's withdrawal transaction is not reliably possible so this attack may operate based upon key indicators that a user is about to withdraw such as Discord messages or market volatility.

Furthermore, if the `DECAYING_REDEMPTION_MIN_BASE_AMOUNT` is configured too high, then a depositor could simply deposit `DECAYING_REDEMPTION_MIN_BASE_AMOUNT - 1` wei multiple times to avoid the decaying redemption fee while still depositing a large amount.

This could occur in a single transaction with a multicall or for-loop contract call around the `mintFor` function.

Recommendation

Configure the `DECAYING_REDEMPTION_MIN_BASE_AMOUNT`, `decayingRedemptionFeeStart` and `decayingRedemptionFeeDuration` with these behaviors in mind.

Ensuring that the `DECAYING_REDEMPTION_MIN_BASE_AMOUNT` is neither too low to incentivize bad faith mints on behalf of other users and that the `DECAYING_REDEMPTION_MIN_BASE_AMOUNT` value is not too high to incentivize split deposits to avoid the decay fee measure.

Resolution

Synthetix Team: Acknowledged.

M-04 | Leverage Mismatch Because Different Price Sources

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol: 226	Acknowledged

Description

When the `LeveragedToken` prepares a rebalance, it calls `buildTransientState` to retrieve the `assetPrice` from Chainlink's aggregator. It then computes how large its position adjustment (`sizeDelta`) should be to maintain the target leverage ratio.

However, the actual execution of that order in `PerpsV2` uses Pyth for an off-chain delayed fill, which can differ from Chainlink by up to the `offchainPriceDivergence` threshold (e.g. 2%).

Consequently, the leveraged token's final fill price may deviate from the aggregator-based simulation, leaving the vault with a leverage ratio substantially different from what it intended to achieve.

Recommendation

Use the same price source the market will rely on for execution. If the vault's off-chain orders are certain to be filled using Pyth, the `LeveragedToken` could incorporate Pyth's price feed (or a close estimate) when computing its position changes.

Resolution

Synthetix Team: Acknowledged.

M-05 | Final Redeemer Pays Decaying Fees And Slippage

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol	Resolved

Description

Within the `LeveragedToken`'s `redeemFor` method, the redemption amount is reduced by a "decaying redemption fee" plus a slippage deduction (intended to represent expected order fees and price impact). Ordinarily, these fees stay in the contract or go to other token holders.

However, if the user redeeming is the last, meaning he redeems the entire `totalSupply`, there are no other holders to benefit from these leftover fees and the contract itself can no longer distribute them. That portion of sUSD remains stuck in the `PerpsV2` protocol as unused margin.

The final user is penalized, losing this fraction of their redeemable amount for no net benefit to the system. On the other hand, this also happens if the `LeveragedToken` currently has no position as it dropped below 100 sUSD and the rebalancer closed it.

In that case it is not fair to remove this slippage amount from the user and distributing it among the other users.

Recommendation

When `leveragedTokenAmount = totalSupply`, skip collecting the decaying redemption fee and slippage.

Resolution

Synthetix Team: The issue was resolved in PR#11.

M-06 | Missing Execute Order Function

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol	Acknowledged

Description

During `submitOffchainDelayedOrderWithTracking`, a `keeperFee` is deducted from the position margin: `_updatePositionMargin(messageSender, position, sizeDelta, fillPrice, -int(keeperDeposit));`

This fee is reserved for the caller of `executeOffchainDelayedOrder` that will settle the delayed order. However, according to the `PerpsV2MarketDelayedExecution.sol` contract: `If this is called by the account holder the keeperFee is refunded into margin, otherwise it sent to the msg.sender`.

For the ETH Perp market, this keeper fee ranges from 1.05 to 100 sUSD. The `LeveragedToken` is the account holder in this case, but it does not contain any function to execute the delayed order, missing out on the fee refunds.

Recommendation

Consider adding a public function to execute delayed orders and get the keeper fee back.

Resolution

Synthetix Team: Acknowledged.

M-07 | Depositing Minimum Amount Leads To Position Closure

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol	Acknowledged

Description

When a user deposits the MINIMUM_MINT_AMOUNT into an empty LT a position will be created and most likely closed right after. This punishes the user with paying for order fees twice without any benefit:

• User deposits the MINIMUM_MINT_AMOUNT (100 sUSD) into an empty LT

• The _validateMintAmount check is executed and it passes as the given amount is not less than the MINIMUM_MINT_AMOUNT of 100 sUSD

• The given amount of 100 sUSD is deposited as margin

• The _canRebalance check is executed and it passes as the remainingMargin (100 sUSD) in the position is not less than the MINIMUM_MARGIN_BALANCE (100 sUSD) it is equal and the position is heavily under leveraged as it has no size yet

• Therefore _rebalance will be executed and a delayed order to open a position in perps v2 is created

• The delayed order is executed:

• To open the position order fees and price impact must be paid and therefore the position's remainingMargin is probably < 100 sUSD now

• This will trigger the rebalancer:

• The _canRebalance check is executed and it will pass as the remainingMargin after paying for order fees is less than the MINIMUM_MARGIN_BALANCE now while the notional value of the position is > 0

• Therefore _rebalance will be executed and a delayed order to close the position in perps v2 is created

• The delayed order is executed:

• The position is closed and more order fees are paid

Recommendation

The MINIMUM_MINT_AMOUNT should be significantly more than the MINIMUM_MARGIN_BALANCE.

Resolution

Synthetix Team: Acknowledged.

M-08 | Order Fees Do Not Include Rebalance Costs

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol	Acknowledged

Description

In the `_orderFee` function the order fee computed does not include the fees to cover Perps V2 keeper fee to execute an order or the `rebalance` gas cost (if a rebalance is not triggered during the action)

This is a one time hard cost that will be applied to every rebalance that occurs and is not specifically remunerated by the depositors/withdrawers who are triggering the rebalance.

Recommendation

Consider if this is acceptable. If it is not, consider requiring that the actor who triggers the rebalance covers these fees.

Resolution

Synthetix Team: Acknowledged.

M-09 | Decaying Redemption Fee Can Be Bypassed

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol	Resolved

Description

The `decayingRedemptionFee` function decays the user's fee as time passes. However, the if-else statement's conditions are checked in wrong order. The code tries to set `percentPassed` if `mintedTimestamp[user] = 0` only if `percentPassed = 1e18`.

But if `mintedTimestamp[user] = 0`, the value of `percentPassed` will be much higher than 1e18. This can be combined with the `decayingRedemptionFeeMinBaseAmount` (which is currently set to 5e18) to avoid paying fees.

When a user wants to exit the system, instead of redeeming and paying a decaying fee, they can transfer 5e18 of their tokens as many times as they want to a brand new account.

This will result in them having all of the tokens in that new account and the `mintedTimestamp = 0 = percentPassed = 1e18` and no fees being paid.

Recommendation

Switch the `if-else` conditions - first check `mintedTimestamp[user] = 0` and then `percentPassed > 1e18`.

Resolution

Synthetix Team: The issue was resolved in PR#13.

M-10 | Incorrect Calculation Of currentSize

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol	Resolved

Description

In `_validateMintAmount`, we should avoid using `currentSize = long_ + short_` because `maxMarketValue` returns the maximum allowable value for each side of the market.

Let's say the `maxMarketSize` for each side of the market is 1000, meaning we can open 1000 in long and 1000 in short using the market, which is possible. However, if we do the same using `LeveragedToken`, it causes a DoS.

Recommendation

`currentSize = isLong * long_ : short_`

Resolution

Synthetix Team: The issue was resolved in PR#14.

M-11 | Insufficient OdosRouter Calldata Validation

Category	Severity	Location	Status
Validation	Medium	ZapSwap.sol	Partially Resolved

Description

ZapSwap._validateOdosSwapAllData() tries to validate the whole amount of received sUSDC tokens is being spent by calling swapCompact.

OdosRouter.swapCompact() supports two main formats for each token - input and output. Each of these tokens may be specified directly in the calldata or loading them by the odos router storage.

The code in ZapSwap.validateOdosSwapAllData adjust the amountLengthPosition accordingly depending on which format is used for the input token. However, it assumes the output token will always use the second format where there is no token address in the calldata.

This will result in a failed validation. One of the tokens' bytes will be checked instead of checking the byte showing whether there is input amount specified. This will either result in allowing not all tokens to be spent or reverting if that token byte is not 0.

Recommendation

Consider both formats for both tokens.

Resolution

Synthetix Team: The issue was resolved in PR#16.

M-12 | Tokens Are Tradable When The LT Is Liquidatable

Category	Severity	Location	Status
Validation	Medium	LeveragedToken.sol: 668-685	Acknowledged

Description

• The docs state out that LTs are supposed to be used in third-party DeFi protocols and they are transferable.

• The `mintFor` function checks that the LT's position is not liquidatable and active, to make sure that a user does not enter the system and likely loses their invested funds right after

The `_update` function executed during transfers when users acquire LTs on third-party protocols does not perform these checks. Therefore users could end up acquiring worthless LTs.

Recommendation

Consider reverting in the `_update` function if the LT's position is liquidatable or no longer active.

Resolution

Synthetix Team: Acknowledged.

M-13 | Only Current Mint Amount Validated

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol: 625-666	Acknowledged

Description

In the `_validateMintAmount` function only the current amount being minted, represented as the `mintAmount` is validated against the max market size and value validations.

However several smaller mints could take place where each of the individual mint amounts remain below the max market validations, while the summation of the mints are above the max market validations. All of these mints may occur before a rebalance is triggered.

Additionally, price action can also create an imbalance scenario that will need an increase in position size (additional size delta), which is not contemplated by `_validateMintAmount`.

Finally, order fees could be charged incorrectly, as the outstanding size delta plus the new mint amount, could be a position decrease (i.e. minting when price moves up in a long LT).

Recommendation

Consider validating the current outstanding rebalance `sizeDelta` against the market maximums as opposed to the immediate `mintAmount` that is currently being minted.

Resolution

Synthetix Team: Acknowledged.

M-14 | LeveragedTokens Pay Disproportionately Higher Streaming Fees

Category	Severity	Location	Status
Protocol Design	Medium	LeveragedToken.sol: 195	Acknowledged

Description

Within the LeveragedToken contract, the getStreamingFee(remainingMargin) function is named and documented as if it calculated the streaming fee based on the "remaining margin". However, rather than passing the contract's remainingMargin to the _getStreamingFee call, the code calls it with the position's notionalValue:

```
uint256 streamingFee = _getStreamingFee(transientState.notionalValue);
```

As a result, the leveraged token charges streaming fees on the full leveraged exposure instead of the LeveragedToken position's margin. A 10x LeveragedToken thus would pay 5 times higher streaming fees than a 2x one that has the same actual collateral.

This is at odds with the function's parameter naming convention, which implies that the fee should be assessed against the margin balance rather than the entire notional.

Recommendation

Consider calculating the streaming fee as a percentage of the remaining margin instead of the LeveragedToken's total position notional.

Resolution

Synthetix Team: Acknowledged.

M-15 | Streaming Fee Starts Too Late

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol: 193-202	Acknowledged

Description

The streaming fee is 2%/year of the LT position notional value and starts to accrue when the first redemption happens. If users decide to not redeem for a while after the LT is deployed the protocol may lose significant revenue.

Recommendation

Start to accrue the streaming fees in the first deposit/rebalance instead of the first redemption.

Resolution

Synthetix Team: Acknowledged.

L-01 | _validateMintAmount Over/Underestimate Position Size

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol: 648	Acknowledged

Description

In the LeveragedToken contract, the _validateMintAmount function calculates:

```
uint256 increaseSideSize = mintAmount.mul(targetLeverage).div(state.assetPrice);
```

to represent how many "units" of the underlying asset could be added if the protocol rebalanced toward targetLeverage. This is then used to ensure that (currentSize + increaseSideSize) = maxSideSize, preventing the market from exceeding Synthetix's maxMarketValue.

However, this logic ignores the existing leverage level and the actual final position size post-rebalance. For example:

• Over-levered scenario: If the current actual leverage of the contract is already above targetLeverage, adding margin might reduce or not increase the net side at all. The system's calculation incorrectly assumes the new margin will open a large new position, potentially blocking an otherwise safe mint.

• Under-levered scenario: Even if the protocol is significantly under target leverage, the computed mintAmount * targetLeverage / assetPrice will deviate from the real final notional after rebalancing.

Typically the code is overestimating the side to remain safe, but it can lead to unnecessary reverts. Hence, the contract can revert in borderline cases where, in reality, the final post-rebalance position would not exceed maxMarketValue.

The user sees MaxMarketValueExceeded error even though the final real position is safe. The root cause of this issue is that the contract uses the targetLeverage for the increaseSideSize calculation instead of the current LeveragedToken's position leverage.

Recommendation

Consider accounting for the the current notional vs. margin to see whether the new deposit will actually buy more underlying or simply reduce the LeveragedToken's leveraged ratio.

By adjusting _validateMintAmount to more accurately model the actual final position size (or clarifying in the documentation that an overestimation is intentionally used), the protocol can avoid unnecessary revert scenarios while still respecting the Synthetix maxMarketValue constraints.

Resolution

Synthetix Team: Acknowledged.

L-02 | Possible Full Position DoS Due To Streaming Fee

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol: 199	Acknowledged

Description

When a `LeveragedToken`'s underlying PerpsV2 position saturates `maxMarketValue`, no additional mints are allowed (they revert in `_validateMintAmount`). If, over time, the streaming fee accumulates to a large amount, a `rebalance` eventually calls:

```
_withdrawMargin(streamingFee + ... )
```

to pay that fee. However, Synthetix reverts if removing that margin would leave the position liquidatable or under the maintenance threshold.

Because no user can deposit new margin (the market is at `maxMarketValue`), there is no way to replenish margin. The rebalance reverts, blocking the system from collecting streaming fees and leaving the position stuck.

Recommendation

Consider implementing a function that is called frequently to charge the streaming fee manually. On the other hand, in an extreme edge case like the one described, the protocol might discount or waive streaming fees if the market is at max capacity and the position is near liquidation, preventing an indefinite stuck state.

Resolution

Synthetix Team: Acknowledged.

L-03 | Unused Functions

Category	Severity	Location	Status
Code Best Practices	Low	LeveragedToken.sol	Partially Resolved

Description

The following libs/params/functions are not used:

`LeveragedToken.sol`

• constant `_COLLATERAL_ID`

• constant `_SETTLEMENT_STRATEGY_ID`

• internal `function _marginBelowMinimum`

• internal `function _chargeRebalanceFee`

`Config.sol`

• constant `PYTH_PRICE_HANDLER_INITIAL_ETH`

`LeveragedTokens`

• entire library

`ProxyOwnerDelays`

• entire library

Recommendation

Implement the functions/params in the code or consider removing them to avoid confusion.

Resolution

Synthetix Team: The issue was resolved in PR#8.

L-04 | Referral System Is Not Implemented

Category	Severity	Location	Status
Code Best Practices	Low	LeveragedToken.sol	Acknowledged

Description

Within the `LeveragedToken` contract, there are references to a referral mechanism, such as a `referralRatio` or a parameter for `referralCode` in mint/redeem flows. However, no actual referral logic is integrated to distribute a share of fees on-chain to referrers.

Recommendation

If the protocol has no near term plan to implement a real referral program, remove `referralCode` parameters and `referralRatio` references from the codebase.

On the other hand, if a referral system is desired in the future, add actual logic that calculates a portion of the redemption fee or streaming fee to route to the specified referral address.

Resolution

Synthetix Team: Acknowledged.

L-05 | Pyth Price Confidence Interval Is Ignored In PerpsV2

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol	Acknowledged

Description

PerpsV2 relies on Pyth for the execution of offchain delayed orders, yet it only checks the final numeric Pyth price and its timestamp, ignoring the confidence interval (`conf`) that signals how uncertain Pyth is about the asset's value.

While the protocol partially guards against extreme price deviations by comparing the Pyth price to Chainlink and reverting if the two differ too sharply, it can still accept a Pyth price with a very wide confidence band so long as it numerically aligns with Chainlink.

In conditions of high volatility or limited liquidity, a large `conf` should raise caution; by disregarding it, PerpsV2 (and thus the `LeveragedToken` that interacts with it) may proceed with the execution of delayed orders even though Pyth itself indicates low price confidence.

Recommendation

Consider introducing a maximum allowable confidence threshold for Pyth-based off chain delayed orders.

Resolution

Synthetix Team: Acknowledged.

L-06 | LTs Could Be Forced To Remain Outside Exact Target Leverage

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol	Acknowledged

Description

The `LeveragedToken` contract checks if the current leverage deviation factor is at or above `rebalanceThreshold` (e.g., 10%) to decide whether to call `submitLeverageUpdate`.

If `leverageDeviationFactor < rebalanceThreshold()`, no rebalance occurs and the system assumes the token is "close enough" to its target leverage.

While this avoids excessive rebalances and gas costs, it means a user can consistently keep the leveraged token at, say, 8% off the target, never quite triggering the threshold, yet still meaningfully deviating from the exact leverage ratio the token aims to maintain.

In practical terms, the token never enters the "must rebalance" zone, so it settles into a zone below that threshold. Any small price fluctuation or marginal deposit might keep the leverage from hitting the threshold, thereby preventing a forced `_submitLeverageUpdate`.

Over time, new participants might assume the token is rigorously pinned to the advertised leverage ratio, but it can persist at some modest offset that never crosses the threshold.

This discrepancy could introduce a minor but continuous tracking error between the token's actual leverage and its nominal target, especially if the threshold is relatively large (like 10%).

On the other hand, if the `LeveragedToken` is rebalanced, forced by the mint/redeem of other user, the malicious user can just execute another mint/redeem that resets the `LeveragedToken` 's position leverage back to the previous value (i.e. 8% off the target).

Recommendation

This is largely an informational issue, as no trivial fix exists without undermining the benefit of skipping small or constant rebalances.

Resolution

Synthetix Team: Acknowledged.

L-07 | Temporary DoS Due To Price Divergence

Category	Severity	Location	Status
DoS	Low	LeveragedToken.sol	Acknowledged

Description

The `LeveragedToken` triggers its rebalances through off chain delayed orders in PerpsV2. If the Chainlink price and the Pyth off chain price diverge beyond the `offchainPriceDivergence` threshold, all of chain orders revert with a “price divergence too high” error.

Because the `LeveragedToken` relies on off chain orders for key operations (like `_submitLeverageUpdate`), it becomes blocked from rebalancing while this mismatch persists. This will block as well any calls to the `mintFor` and `redeemTo` functions.

During that time, the contract’s leverage might drift dangerously close to liquidation. Once the oracle feeds realign, it is possible that the `LeveragedToken`'s position has reached the liquidatable state and gets flagged for liquidation before the rebalance delayed order is executed.

Recommendation

Implement an owner-only “emergency” function that bypasses the reliance on off chain orders if prolonged feed divergence occurs.

In other words, allow the contract owner (or a trusted multisig) to manually control or adjust leverage on-chain so that the vault can not face a forced liquidation the moment the feeds revert to normal alignment.

Resolution

Synthetix Team: Acknowledged.

L-08 | Incorrect Order acceptedPrice Calculation

Category	Severity	Location	Status
Logical Error	Low	LeveragedToken.sol: 547	Resolved

Description

During `_submitLeverageUpdate` and `_closePosition`, an `acceptablePrice` is calculated using the `slippageTolerance`.

If the `slippageTolerance` is 2% (default value), the `acceptedPrice` will be:
• `assetPrice` * (1.02) = 2% above
• `assetPrice` / (1.02) = 1.96% below

Recommendation

Consider updating the acceptedPrice formula for lower price as follows:

`acceptedPrice = state.assetPrice.mul(1e18 - slippageTolerance);`.

Resolution

Synthetix Team: Resolved.

L-09 | LeveragedToken Can Be Reactivated

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol	Acknowledged

Description

`LeveragedToken.isActive()` should return false if the token's position has been liquidated. The active status is equal to the expression `exchangeRate() > 0`. If the `totalSupply` of the tokens is 0, `exchangeRate()`, will return 1e18.

The token allows anyone to burn their tokens. Since the `burn()` function can be executed at any point in time, if all of the holders of the inactive token burn their holdings, the token will be reactivated.

Recommendation

Consider not allowing `burn()` if the token is inactive.

Resolution

Synthetix Team: Acknowledged.

L-10 | Charging Wrong Redemption Fee

Category	Severity	Location	Status
Logical Error	Low	LeveragedToken.sol	Acknowledged

Description

The redemption fee is currently charged on the amount of sUSD redeemed minus slippage and then multiplied by the target leverage, instead of charging it on the amount of sUSD redeemed multiplied by the target leverage as stated in the documentation.

Recommendation

Consider charging fee on the amount of sUSD redeemed multiplied by the target leverage.

Resolution

Synthetix Team: Acknowledged.

L-11 | Wrong Comparison Operators In _closePosition And _submitLeverageUpdate

Category	Severity	Location	Status
Logical Error	Low	LeveragedToken.sol	Resolved

Description

In `_closePosition` and `_submitLeverageUpdate`, we are using the wrong comparison operators. For example, in `_closePosition`, if `estimatedSizeDelta` is greater than 0, we check whether `fillPrice` is less than `acceptedPrice`.

However, we should also submit the order when `fillPrice` is equal to `acceptedPrice`. We should also submit the order when `estimatedSizeDelta` is less than 0 and `fillPrice` is equal to `acceptedPrice`. The same should be done in `_submitLeverageUpdate`.

Recommendation

In `_closePosition`, use `estimatedSizeDelta > 0  fillPrice = acceptedPrice : fillPrice = acceptedPrice`, and in `_submitLeverageUpdate`, use `sizeDelta > 0  fillPrice = acceptedPrice : fillPrice = acceptedPrice` in If-else.

Resolution

Synthetix Team: Resolved.

L-12 | mintedTimestamp Is Not Recorded

Category	Severity	Location	Status
Logical Error	Low	LeveragedToken.sol	Resolved

Description

When `baseAmount` equals `decayingRedemptionFeeMinBaseAmount`, the `mintedTimestamp` is not recorded.

Recommendation

Consider updating to the following implementation::

```
function _update(
    address from,
    address to,
    uint256 value
) internal virtual override {
    super._update(from, to, value);
    uint256 baseAmount = value.mul(exchangeRate());
    if (
        to != address(0) &&
        baseAmount >=
        _addressProvider
            .parameterProvider()
            .decayingRedemptionFeeMinBaseAmount()
    ) {
        mintedTimestamp[to] = block.timestamp;
        emit MintedAmountIncreased(to, value);
    }
}
```

Resolution

Synthetix Team: Resolved.

L-13 | Redemption Fee Bounds

Category	Severity	Location	Status
Validation	Low	Global	Acknowledged

Description

The value of the `REDEMPTION_FEE` in %s, should always be less than `1 / maxLeverage`, where `maxLeverage` is the maximum leverage that's going to be supported by the system.

Otherwise, the computed `redemptionFee` in `redeemFor` will be greater than `baseWithdrawn` and the transaction will revert because of subtraction underflow.

Recommendation

Be aware of these bounds limitations.

Resolution

Synthetix Team: Acknowledged.

L-14 | Unnecessary notionalValue Computation

Category	Severity	Location	Status
Code Best Practices	Low	LeveragedToken.sol	Resolved

Description

Inside `_closePosition()`, when computing `estimatedSizeDelta`, the `notionalValue()` function is unnecessary called instead of using `state.notionalValue`.

Recommendation

Use `state.notionalValue()`.

Resolution

Synthetix Team: Resolved.

L-15 | DOS For USDT In ZapSwap

Category	Severity	Location	Status
Logical Error	Low	ZapSwap.sol	Resolved

Description

`ZapSwap.mint()` approves the `odosRouter` to spend a `zapAssetAmountIn` tokens before the swap. However, this amount is not checked to match the actually swapped amount passed to the router.

This allows malicious users to block the `ZapSwap` functionality for tokens that revert when their approval is changed from a non-zero value to another non-zero value.

Recommendation

If you want to support the aforementioned tokens, validate the whole `zapAssetAmountIn` is being spent.

Resolution

Synthetix Team: Resolved.

L-16 | ZapSwap Does Not Use TlxOwnable onlyOwner Modifier

Category	Severity	Location	Status
Code Best Practices	Low	ZapSwap.sol	Resolved

Description

`ZapSwap` contract currently does not use the `onlyOwner` modifier from `TlxOwnable`.

Recommendation

Do not inherit `TlxOwnable` in `ZapSwap` contract.

Resolution

Synthetix Team: Resolved.

L-17 | Wrong Comment

Category	Severity	Location	Status
Code Best Practices	Low	Config.sol	Resolved

Description

The comment next to `REBALANCE_FEE` in `Config.sol` says `2sUSD`, but the actual value is 0.

Recommendation

Correct the comment.

Resolution

Synthetix Team: Resolved.

L-18 | Unused Rebalance Fee

Category	Severity	Location	Status
Code Best Practices	Low	Global	Resolved

Description

The `LeveragedToken.chargeRebalanceFee()` is never called which means the rebalance fee is never charged.

Recommendation

Consider removing the code before deployment.

Resolution

Synthetix Team: Resolved.

L-19 | isActive() Semantics

Category	Severity	Location	Status
Protocol Design	Low	LeveragedToken.sol	Acknowledged

Description

According to the comment in ILeveragedToken.sol, isActive() will return true if the position of the leveraged token has not been liquidated.

isActive() returns true if exchangeRate > 0. This means it will return false when the position has been closed and all the margin has been withdrawn, even if it hasn't been liquidated.

If that's the intended behavior, keep in mind that closing a position may also result in leftover margin, which will make the isActive() function return true again.

Recommendation

Be aware of the different behaviors of this function.

Resolution

Synthetix Team: Acknowledged.

L-20 | Lack Of Storage Gaps In TlxOwnableUpgradeable

Category	Severity	Location	Status
Code Best Practices	Low	TlxOwnableUpgradeable.sol	Acknowledged

Description

Currently there are no storage gaps in `TlxOwnableUpgradable` which means if a new variable is added to it, the storage layout of the inheriting contracts will be corrupted.

Recommendation

Consider adding storage gaps.

Resolution

Synthetix Team: Acknowledged.

L-21 | Wrong Emission In Rebalanced Event

Category	Severity	Location	Status
Code Best Practices	Low	LeveragedToken.sol	Resolved

Description

There is a possibility that `isSuccess` is `false`, which means we should not emit the `Rebalanced` event because technically no rebalance occurred.

Recommendation

Do not emit the `Rebalanced` event even when `isSuccess` is `false`.

Resolution

Synthetix Team: Resolved.

L-22 | Unsafe Transfer Of Ownership In The AddressProvider

Category	Severity	Location	Status
Code Best Practices	Low	AddressProvider.sol	Resolved

Description

Currently, we have an `updateAddress` function that can be used to change the owner by updating `AddressKeys.OWNER`, but this is a very unsafe method.

Recommendation

Consider using a two step ownership transfer, similar to the implementation of `Ownable2Step` but customized for the `AddressProvider` contract.

Resolution

Synthetix Team: Resolved.

L-23 | Wrong Emission Of MintedAmountIncreased Event

Category	Severity	Location	Status
Code Best Practices	Low	LeveragedToken.sol	Acknowledged

Description

`MintedAmountIncreased` event is emitted only when `baseAmount` is more than `decayingRedemptionFeeMinBaseAmount`, which is incorrect because technically we are increasing the mint amount when `baseAmount` is equal or less than `decayingRedemptionFeeMinBaseAmount`.

Recommendation

Consider also emitting the `MintedAmountIncreased` event when `baseAmount` is equal or less than `decayingRedemptionFeeMinBaseAmount`.

Resolution

Synthetix Team: Acknowledged.

L-24 | Users Pay Slippage If There Is No Position

Category	Severity	Location	Status
Logical Error	Low	LeveragedToken.sol	Resolved

Description

The slippage (expected order fees and price impact) is decreased from the users received sUSD amount when a user redeems LTs. This also happens if the LT currently has no position as it dropped below 100 sUSD and the rebalancer closed it.

In that case it is not fair to remove this slippage amount from the user and distributing it among the other users.

Recommendation

Only calculate and remove the slippage from the users received sUSD amount in the `redeemFor` function if the LT has a open position.

Resolution

Synthetix Team: Resolved.

L-25 | Precision Loss In _getLeverageUpdateSizeDelta Function

Category	Severity	Location	Status
Logical Error	Low	LeveragedToken.sol	Acknowledged

Description

The `_getLeverageUpdateSizeDelta` calculates `sizeDelta` by dividing twice by `assetPrice` which leads to double the precision loss and inaccurate leverage size update.

Recommendation

The `sizeDelta` calculation can be changed to: `int256 sizeDelta = (marginAmount.mul(targetLeverage) - notionalValue_).div(assetPrice_);`

Resolution

Synthetix Team: Acknowledged.

L-26 | Fee Calculations Are Rounded Down

Category	Severity	Location	Status
Logical Error	Low	LeveragedToken.sol	Acknowledged

Description

All the fee calculations in the `LeveragedToken` are rounding down. This precision loss is more significant in the `_getStreamingFee()` function where `annualStreamingFee` can be with 1 wei less.

Since that value is multiplied by the passed seconds since the last streaming update, a total loss of 1wei * seconds will be experienced for the protocol.

Recommendation

Consider rounding up the fee calculations.

Resolution

Synthetix Team: Acknowledged.

L-27 | Users Can Redeem While The Contract Is Paused

Category	Severity	Location	Status
Code Best Practices	Low	LeveragedToken.sol	Acknowledged

Description

When the `LeveragedToken` is paused, users cannot mint anymore, but they can still redeem which may lead to unexpected results if the assumption is that they can't.

Recommendation

Consider pausing `redeemFor` as well.

Resolution

Synthetix Team: Acknowledged.

L-28 | High Rebalance Threshold Causes Funds To Become Locked

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol	Acknowledged

Description PoC

This issue occurs because users are only redeeming shares and no one is minting new shares. The problem is that if we have high leverage and a high rebalance threshold, you cannot redeem shares beyond a certain amount.

This is because before the rebalance threshold is reached and a rebalance is triggered, the transaction will revert with a `MaxLeverageExceeded` error. For example, I was testing with a 10x leverage token and a rebalance threshold of 50%.

Before the threshold was reached and the rebalance could occur, I was trying to redeem shares such that it triggers the rebalance and then redeem again to redeem all shares, the transaction reverted with a `MaxLeverageExceeded` error because we could not reach rebalance threshold as `MaxLeverageExceeded` occurred first.

See below the formula to calculate how much remaining margin we should have such that it triggers the rebalance

IM = Initial margin
RM = Remaining margin
TP = Rebalance Threshold percentage, like if 50% then 50
LV = Leverage
RM = (IM * LV * 100)/(50*(LV*2) + 100*LV)

Recommendation

By limiting the maximum rebalance threshold to about 0.2% or more, so we can ensure that all shares can still be redeemed without triggering the `MaxLeverageExceeded` error.

Resolution

Synthetix Team: Acknowledged.

L-29 | Adresses Can Not Be Unfrozen In AddressProvider Contract

Category	Severity	Location	Status
Code Best Practices	Low	AddressProvider.sol	Acknowledged

Description

Currently we have a function to freeze an address, but we don't have one to unfreeze it.

Recommendation

Include a function to unfreeze the address as well if required.

Resolution

Synthetix Team: Acknowledged.

L-30 | DoS On High Usage

Category	Severity	Location	Status
Protocol Design	Low	LeveragedToken.sol: 135	Acknowledged

Description PoC

During the initial phase of a `LeveragedToken`, most mints and redeems will likely trigger a rebalance as the threshold can be easily breached. This will submit a leverage update order in SNX.

Therefore, an attacker may use this to grief other users from minting tokens, by constantly minting and redeeming from the contract, causing a leverage update every time.

The `_ensureNoPendingLeverageUpdate` check will prevent any user actions until the order is executed.

Recommendation

Here are a few possible mitigations:

• Implement an action queue and a keeper bot that executes these actions and do not allow users to spam this queue

• Do not allow redemptions for a configured timespan after the LT is deployed

• Document this behavior so users are aware of this DoS attack and can counter it by increasing the gas amount they are willing to pay

Resolution

Synthetix Team: Acknowledged.

L-31 | Missing Force Rebalance Functionality

Category	Severity	Location	Status
DoS	Low	LeveragedToken.sol: 531-566	Acknowledged

Description

When the rebalancer rebalances the LT's position, it calls `submitOffchainDelayedOrderWithTracking` in the perps v2 system with a fixed slippage check of the current price +/- 2% and there is no way to change that.

This can lead to the LT getting liquidated in a black swan event, for example:

• LT longs an asset

• A black swan event occurs and the asset's price falls relatively fast

• Traders want to profit from that and open a lot of interest on the short side

• The position's notional falls as its PnL decreases and it therefore becomes overleveraged

• The `LeveragedToken` tries to rebalance but the order will revert because many users short right now and therefore the price impact outweighs the 2% slippage

• The price decreases further and the LT is liquidated

A malicious actor could also on purpose push the price impact so high that the keeper is not able to rebalance and the LT gets liquidatable over time to profit from the liquidation fees.

This may not be profitable under normal balanced conditions, but could become a valid attack path in black swan events.

Recommendation

Add the functionality that the rebalance is able to enforce a rebalance with a bigger slippage parameter to protect users from getting liquidated in such black swan events.

Resolution

Synthetix Team: Acknowledged.

L-32 | Vault Could Be Drained By Cancellation Fees

Category	Severity	Location	Status
Logical Error	Low	LeveragedToken.sol: 540-562	Acknowledged

Description

In the SNX perps market when a cancellation is performed the user who initiates the order cancellation is rewarded for invoking the transaction.

This allows to drain an LT in the following way:

• The current price impact leads to the fill price almost reaching the 2% slippage in the `_submitLeverageUpdate` function

• The keeper bot calls `rebalance`

• A malicious actor pushes the skew even further so that the LT's order is not fulfillable as the accepted price slippage check will fail

• The order becomes cancellable and the malicious actor takes the fee

• The malicious actor pushes the skew back so that the keeper bot calls `rebalance` again

• repeat

Recommendation

Consider making the slippage check to create the delayed order a bit stricter than the accepted price passed on to the Perps V2 system.

Resolution

Synthetix Team: Acknowledged.

L-33 | Missing Check If targetLeverage = 1e18

Category	Severity	Location	Status
Logical Error	Low	LeveragedTokenFactory.sol: 54	Acknowledged

Description

Deploying a `LeveragedToken` is permissioned, as only the owner can execute the `createLeveragedTokens`. The function validates if `targetLeverage` is below 50% of the market max leverage, and avoid leverages with more than 2 decimals (i.e. 1.435x).

However, there is no validation for minimum leverage, so values below 1x leverage are still considered valid. Therefore, this will result in a `LeveragedToken` that is not very attractive but will be displayed in the UI and receive deposits.

Recommendation

Validate that `targetLeverage` is above 1e18.

Resolution

Synthetix Team: Acknowledged.

L-34 | Centralization Risk

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol: 155-160	Acknowledged

Description

The owner of the `AddressProvider` contract is able to update the `ZapSwap` contract address to any arbitrary address.

This address is then able to drain the protocol completely by calling the `redeemFor` function to redeem/steal the funds of all users. This can be very dangerous if the private key of the owner falls into the wrong hands.

Recommendation

Work with an allowance mechanic here instead.

Resolution

Synthetix Team: Acknowledged.

L-35 | Early Return computePriceImpact

Category	Severity	Location	Status
Code Best Practices	Low	LeveragedToken.sol: 359	Resolved

Description

When `priceImpact = 0` the `computePriceImpact` function will not early return with `orderFee`. Instead, it will spend gas calculating `priceImpactPercent` and `rebalanceCharge` but both will be zero. Therefore, the answer is the same as with the early return.

Recommendation

Consider adding an equality check to the early return:
`if (priceImpact = 0) return (orderFee);`

Resolution

Synthetix Team: Resolved.

L-36 | _redeemLeveragedToken Return Value Not Used

Category	Severity	Location	Status
Code Best Practices	Low	ZapSwap.sol: 226	Acknowledged

Description

The `_redeemLeveragedToken` returns the amount of base assets redeemed. However, this value is not read anywhere in the `ZapSwap` contract.

Recommendation

Remove the return value from `_redeemLeveragedToken` function.

Resolution

Synthetix Team: Acknowledged.

L-37 | _addressProvider Initialized Twice

Category	Severity	Location	Status
Code Best Practices	Low	LeveragedTokenFactory.sol: 48	Acknowledged

Description

During the initialization of `LeveragedTokenFactory`, the `__TlxOwnableUpgradeable_init` call stores the `addressProvider` in the state. However, the `initialize` function also initializes the same state address.

Recommendation

Remove the `_addressProvider` initialization in the `initialize` function as the `TlxOwnableUpgradeable` already takes care of this.

Resolution

Synthetix Team: Acknowledged.

L-38 | Misleading Function Param Name

Category	Severity	Location	Status
Code Best Practices	Low	LeveragedToken.sol: 481	Resolved

Description

The `_getStreamingFee` has one function param, `remainingMargin_`. However, the streaming fee is calculated based on the `notionalValue`. The param name is misleading.

Recommendation

Update the `_getStreamingFee` function param name to `notionalValue_`.

Resolution

Synthetix Team: Resolved.

L-39 | Missing Minimum Amount Checks

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol: 84-223	Acknowledged

Description

The `mintFor` & `redeemFor` functions do not always enforce minimum amounts. The best practice would be to add minimum amount checks as nothing good comes from 1 wei actions.

It is for example possible for users to avoid paying fees by redeeming dust amounts as the fee will round down to zero.

Recommendation

Consider adding minimum amount checks.

Resolution

Synthetix Team: Acknowledged.

L-40 | MINIMUM_MINT_AMOUNT Not Initialized

Category	Severity	Location	Status
Code Best Practices	Low	Global	Resolved

Description

The following parameters are not initialized during protocol deployment:
• `MINIMUM_MARGIN_BALANCE`
• `MINIMUM_MINT_AMOUNT`

As the default value is 0, this will allow users to mint small amounts, and the `_canRebalance` function will always calculate deviation factor.

Recommendation

Consider adding these parameters to the deployment script to be sure they are properly initialized with config values.

Resolution

Synthetix Team: Resolved.

L-41 | Unnecessary Price Impact Charged

Category	Severity	Location	Status
Logical Error	Low	LeveragedToken.sol	Acknowledged

Description

Price impact is calculated when minting and redeeming LTs. However, a mint or redeem could actually balance the LT position to be in perfect leverage again and the user has to pay price impact on it. This will effectively punish users who contribute to the health of the system.

Recommendation

Consider to incentivize users to deposit/redeem to balance the system by not letting them pay for price impact and order fees when their deposit/redeem balances the LT further instead of imbalancing it.

Resolution

Synthetix Team: Acknowledged.

L-42 | Freezing Address Not Yet Set

Category	Severity	Location	Status
Validation	Low	AddressProvider.sol: 43	Acknowledged

Description

Owner is able to `freezeAddress`, which will lock this address value forever. However, the frozen address could be a value that is not yet set in the `AddressProvider` contract, and there is no validation for `address(0)`. This will prevent `updateAddress` to be called to initialize the value.

Recommendation

Validate if `_addresses[key]` is not `address(0)` during the `freezeAddress` call.

Resolution

Synthetix Team: Acknowledged.

L-43 | Remaining Margin Can Be 0

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol: 141	Acknowledged

Description

The `redeemFor` does not check for `canLiquidate` like in `mintFor`, so remaining margin can be 0, but it does not fail until the `_withdrawMargin` is called.

Recommendation

Consider adding the `canLiquidate` validation in `redeemFor`

Resolution

Synthetix Team: Acknowledged.

L-44 | canRebalance Does Not Check Market Limits

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol: 297	Acknowledged

Description

The `_canRebalance` verifies if the leveraged token can be rebalanced due to the deviation factor between target and current leverage.

However, there may be cases where the market value is close to max limit, and the rebalance `sizeDelta` increase order can't be executed. The `_canRebalance` does not check for these limits.

Recommendation

During `_canRebalance` verify if the current rebalance `sizeDelta` will breach market value limits.

Resolution

Synthetix Team: Acknowledged.

L-45 | Zero Returns On Stale Prices

Category	Severity	Location	Status
Logical Error	Low	LeveragedToken.sol: 403	Acknowledged

Description

Multiple SNX `PerpsV2` market calls like for example `notionalValue` or `remainingMargin` return 0 when the asset price is invalid (invalid exchange rate, price is 0 or synth is suspended).

This can lead to frontend bugs and critical vulnerabilities in third party protocols which interact with LTs and are not aware of this behaviour.

Recommendation

Consider reverting in these functions if the price is invalid, same as in `assetPrice`, or returning the `invalid` state.

If this is an expected behavior, consider documenting this clearly so integrations are aware of unexpected values being returned in special cases.

Resolution

Synthetix Team: Acknowledged.

L-46 | Delayed Offchain Order May Become Unexecutable

Category	Severity	Location	Status
Validation	Low	LeveragedToken.sol	Acknowledged

Description

When a `LeveragedToken` submits an off chain delayed order to Synthetix via `submitOffchainDelayedOrderWithTracking`, there is a significant risk of revert at execution time if market conditions have changed or if the order constraints are no longer satisfied.

In particular:
• The final fill price may exceed the order's `acceptablePrice` if the market moves against them in the interim. Synthetix enforces a strict price range, so attempting execution then, reverts with a price out-of-range error.

• Exceeding `maxMarketValue`: If other traders open large positions after the user's order was submitted, the order size might exceed the Synthetix `maxMarketValue` at execution time. Even though it was valid at submission, the system will now reject the trade.

If that execution can not complete (due to slippage, `maxMarketValue` reached or other constraints), there is no direct function in the `LeveragedToken` contract to cancel the order until it becomes stale (`offchainDelayedOrderMaxAge` has passed), at which point anyone may cancel it.

During that period, further rebalances cannot proceed, because Synthetix disallows submitting a new order while one is still open. Moreover, any call to `mintFor` and `redeemFor` would revert due to the `_ensureNoPendingLeverageUpdate()` check. To get out of this temporary DoS state, eventually someone must either execute the order if conditions improve, or cancel it.

Recommendation

Consider implementing a function in the `LeveragedToken` contract allowing an authorized party (e.g., owner or designated keeper) to cancel the pending off chain order before it goes stale, if the protocol detects it is clearly unexecutable.

On the other hand, introduce (or incentivize) a keeper that tracks pending rebalancing orders. If the order fails to execute due to slippage or market limits, the keeper can quickly cancel it once it's allowed, minimizing the time rebalances are blocked.

Resolution

Synthetix Team: Acknowledged.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits