NUTS TAPIO SECURITY AUDIT REPORT

September 19, 2025

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 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.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals

- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

The codebase provides a rebaseable LPToken and an ERC4626-compliant WLPToken for flexible liquidity management. The main SelfPeggingAsset contract employs a StableSwap invariant-based pool to facilitate swapping, minting, and redemption of pool tokens backed by multiple underlying assets. Finally, the SelfPeggingAssetFactory contract orchestrates creation of new pools and associated tokens, streamlining deployments.

1.4 Project Dashboard

Project Summary

Title	Description
Client	NUTS Finance
Project name	Tapio
Timeline	15.01.2025 - 08.09.2025
Number of Auditors	3

Project Log

Date	Commit Hash	Note
15.01.2025	661172589070b415c95f7cda24b53a63437601be	Initial commit
11.03.2025	b5b927838510edcf2672ca4ba8601dccf1c757c8	Re-audit commit
27.03.2025	55c07d343b446f702c46bbac83d899e651f6f3b3	Second re-audit commit
07.04.2025	9ad2653917fab0e93db7d172906c338670ef2400	Third re-audit commit
16.04.2025	08b66f145cd12b8f961df5d5709b55ed58497f27	Fourth re-audit commit
05.05.2025	730e8a19fd7ad936c02445df2a36b18f2902f239	Fifth re-audit commit
09.05.2025	8ae30e3d44c84484ef0c917c876222eb84da6dbb	Sixth re-audit commit
06.06.2025	4f6171e19b6c53b9979504cac6c62f38a19f6f93	Seventh re-audit commit
10.06.2025	db85b66eaac16c284525e0b0d075d3958bfe7d13	Eighth re-audit commit
27.06.2025	a6695ca480360e4f72838c27c62f5fca86932747	Ninth re-audit commit

Date	Commit Hash	Note
04.09.2025	500d7bc253a23874a7a2fcb1eb7a984992fd90ac	Commit for the re-audit
08.09.2025	e861622becabb7b3de4cf2d6e1cf326592e8e6c9	Commit for the re-audit

Project Scope

The audit covered the following files:

File name	Link
src/SelfPeggingAssetFactory.sol	SelfPeggingAssetFactory.sol
src/SelfPeggingAsset.sol	SelfPeggingAsset.sol
src/LPToken.sol	LPToken.sol
src/SPAToken.sol	SPAToken.sol
src/WLPToken.sol	WLPToken.sol
src/WSPAToken.sol	WSPAToken.sol
src/periphery/Zap.sol	Zap.sol
src/periphery/RampAController.sol	RampAController.sol
src/periphery/Keeper.sol	Keeper.sol
src/periphery/ParameterRegistry.sol	ParameterRegistry.sol

Deployments

Sonic: mainnet

File name	Contract deployed on mainnet	Comment
ERC1967Proxy.sol	0x0b4C3118D8A48267b2E04E074b516f62401A8D81	Factory Proxy
SelfPeggingAssetFactory.sol	0x7637cFaBE82C983967c202598Ee721909E2412d9	
UpgradeableBeacon.sol	0x5ED560d705A7a8D3bA555BFaD50AE03F07Ad2B74	RampAController Beacon
RampAController.sol	0xDF5553D1Cb12AE0Ec18e39e708477604B9A949dc	
UpgradeableBeacon.sol	0xAda2Dd1344919ddb164c09eB6532f7b51f657341	SelfPeggingAsset Beacon
SelfPeggingAsset.sol	0xAACF301E4397171bf838592b9f4b58Ae44cCf285	
UpgradeableBeacon.sol	0x46695c4a72589Aa29062ad6Ab8Ac0eFd953bfD18	SPAToken Beacon
SPAToken.sol	0x58f335c88c42A23184973561b934444205E906cD	
UpgradeableBeacon.sol	0x00224992044d8E80830076172aDaE85A0005A095	WSPAToken Beacon
WSPAToken.sol	0x14e1Fb1D38AB9fd9c7d423EdE84B86B1F9E5c82c	
ERC1967Proxy.sol	0xa0b447A730A3417aF697CA6fBbC0CF8aa5f8bca9	wSstS Pool Keeper Proxy
Keeper.sol	0xfeb3Fd963F04114ff9825756eBc463743838746D	
ParameterRegistry.sol	0xADf52437fb78b15ad2b68808917e529c42dA1102	wSstS Pool ParameterRegistry
BeaconProxy.sol	0x5f6a416AA8ac47E28b89B2C9b9A65f1139A35469	wSstS Pool Proxy
BeaconProxy.sol	0xA4d07FFB03afC8b137E26E1D4d2Bc2ce7954A670	wSstS Pool SPAToken Proxy

File name	Contract deployed on mainnet	Comment
BeaconProxy.sol	0xd775315136167b50c1B819978A3919925e01E4b3	wSstS Pool WSPAToken Proxy
BeaconProxy.sol	0x536b62f5bdA29eD004A304293Def2E66A7781379	wSstS Pool RampAController Proxy
Zap.sol	0x020DD92C8Fb17D0858dc243c1D1fA1a941Bb8b89	Deployed from db85b66eaac16c284525e0b0d075d3958bfe7d13 commit, no impact on the contract's functionality or security

1.5 Summary of findings

Severity	# of Findings
Critical	1
High	4
Medium	7
Low	22

ID	Name	Severity	Status
C-1	Initial Mint Front-Run Inflation Attack	Critical	Fixed
H-1	Exchange Rate Fluctuations Can Freeze Protocol or Inflate LP Total Supply	High	Fixed
H-2	Arbitrage on Oracle-Driven Price Updates	High	Fixed
H-3	Buffer Drainage Through Repeated Rebase Calls Due to Stale State Variables	High	Fixed
H-4	Flawed Precision and Scaling Logic in `price()`	High	Acknowledged
M-1	Changing `A` Abruptly Without Timelock	Medium	Fixed
M-2	Centralization Risks in Emergency Scenarios	Medium	Acknowledged
M-3	Unclaimed LP Tokens in Zap Contract	Medium	Fixed
M-4	Syncing Issue During Ramping	Medium	Fixed
M-5	Inconsistent A Value in Calculations	Medium	Fixed
M-6	Potential DOS via External Token Transfer Affecting `assert` Checks	Medium	Fixed

M-7	`Zap.zapOut` Uses LP shares Instead of Tokens	Medium	Fixed
L-1	Unbounded Fee Parameters	Low	Acknowledged
L-2	Name and Symbol Do Not Reference Underlying LP Token	Low	Fixed
L-3	Use of String Instead of `bytes` for Oracle Signatures	Low	Fixed
L-4	Public `allowances` Mapping	Low	Fixed
L-5	Fee-On-Transfer Tokens Not Supported	Low	Acknowledged
L-6	Misleading Function Names and Variable	Low	Fixed
L-7	`rebase()` Reverts If `D` Does Not Increase	Low	Fixed
L-8	Inconsistent Share Allocation Due to Exchange Rate Fluctuations Within Margin	Low	Acknowledged
L-9	RampA Logic Not Handling Very Low `initialA`	Low	Fixed
L-10	Lack of `spa` and `wlp` Address Validation in Zap Contract	Low	Acknowledged
L-11	Unnecessary `getCurrentA()` Calls in Non-View Functions	Low	Fixed
L-12	Missing Check for Matching `A` Parameter	Low	Fixed
L-13	Missing Zero-Address Check for Every Feed Entry	Low	Acknowledged
L-14	Initialized `assetDecimals` May Become Stale	Low	Fixed
L-15	Inefficient and Overcomplicated `decimals()` Implementation	Low	Fixed
L-16	Rounding-Error Amplification When Chaining Inverted Price Feeds	Low	Acknowledged
L-17	Volatility Fee Ratio Calculated from The Wrong Denominator	Low	Fixed

L-18	First Trader After Idle Period Pays Accumulated Volatility Fees	Low	Fixed
L-19	`decimals()` Can Be a Constant Value	Low	Acknowledged
L-20	Keeper Contract Lacks a Function to Call `LPToken.setSymbol`	Low	Fixed
L-21	Unused Errors and Event	Low	Fixed
L-22	`getMintAmount()` Misreports Fees	Low	Fixed

1.6 Conclusion

During the audit, we examined the listed attack vectors, verifying the protocol's resistance to potential vulnerabilities, correctness of logic implementation, and resilience against malicious or unintended actions.

Vectors of attacks analyzed during the audit:

- **Donation Attack (Rounding Error Exploit)**
  An attacker could exploit rounding errors arising when tokens are "donated" (transferred without corresponding minting) to the pool, potentially enabling unauthorized withdrawals of other users' funds. We checked if such an attack scenario is feasible within the rounding logic of token accounting.

- **Protocol Lock via Excessive Invariant Checks**
  The protocol could become blocked if overly strict invariant checks fail during normal operations or malicious user actions, resulting in a denial-of-service situation. We checked under which circumstances (normal or malicious user actions) the protocol might experience operational disruptions due to invariant enforcement.

- **Arbitrage via Oracle Price Prediction**
  An attacker might profit unfairly by predicting oracle price updates, executing swaps or deposits before and withdrawals after such updates. We checked if users capable of anticipating oracle-driven price changes could gain unjustified profits.

- **Losses from Curve Parameter Changes**
  Changing curve parameters could potentially result in financial losses for users or the protocol. We checked whether modifying curve parameters could disrupt the protocol's logic or functionality.

- **Protocol Behavior During Negative Yield Events**
  Improper handling of negative yield scenarios could potentially lead to losses for liquidity providers or operational disruptions. We checked that the protocol correctly handles situations involving negative yield, ensuring no unexpected losses occur.

- **Misconfiguration Risk Due to Parameter Validation**
  Incorrectly configured parameters might impact protocol usability or stability. We checked whether configuration parameters are adequately validated by the smart contract logic to prevent accidental or malicious misconfigurations.

The results of our analysis, along with corresponding recommendations, are detailed in the findings below.

2. FINDINGS REPORT

2.1 Critical

C-1	Initial Mint Front-Run Inflation Attack
Severity	Critical
Status	Fixed in b5b92783

Description

This issue has been identified within the `mint()` function of the `SelfPeggingAsset` contract.

An attacker can front-run the initial liquidity provider by depositing a minimal amount (for example, 1 wei) before the first legitimate mint, bypassing the `_minMintAmount` protection. As a result, the legitimate provider may receive substantially fewer LP tokens than expected, effectively suffering a loss of deposited assets.

Attack Scenario

- The attacker front-runs the initial mint operation by first transferring 1 wei of each token. They then mint themselves 1 wei of shares by depositing 1 wei of either token and subsequently transfer the same amounts of tokens that the legitimate minter intended to deposit, thereby inflating the exchange rate.
- The transaction of the initial minter bypasses the `minMintAmount` check because it involves passing the amounts of tokens (not shares) to `poolToken.mintShares`. Inside `poolToken.mintShares`, the amount of tokens is rounded down, resulting in 0 shares being minted.
- As a result, the legitimate provider, attempting to mint even with a slippage `minMintAmount` parameter being set, is effectively undercut and receives no newly minted shares, while the attacker ends up with nearly all of them.

The issue is classified as **critical** severity because it enables the theft of the entire initial liquidity deposit, causing significant financial loss and severely damaging user confidence.

Recommendation

We recommend minting a small "dead share" (e.g., `1000 wei`) as part of the initial LPToken mint to ensure that a strict minimum number of shares is always created in the first deposit. Alternatively, consider initializing the pool in a paused state, allowing only the admin to perform the initial mint. Additionally, consider minting shares equal to the sum of the `totalSupply` and `_tokenAmount` if the current `totalShares` value of the

LPToken is zero. This approach ensures that the exchange rate cannot be unfairly inflated during the first deposit, effectively preventing this front-running attack.

Client's Commentary

> We will mint shares equal to the sum of the totalSupply and _tokenAmount if the current totalShares value of the LPToken is zero

2.2 High

H-1	Exchange Rate Fluctuations Can Freeze Protocol or Inflate LP Total Supply
Severity	High
Status	Fixed in b5b92783

Description

This issue has been identified within the `collectFeeOrYield()` logic of the `SelfPeggingAsset` contract.

If yield margin is low and pool has sufficient liquidity, then even a slight decrease in an underlying token's exchange rate triggers the `ImbalancedPool` error during fee or yield collection, causing the invariant check to revert. Because yield collection occurs in every user-facing function (mint, swap, redeem, etc.), this effectively locks the entire pool until the owner or an administrator intervenes.

To prevent such freezes, one could set the existing yield margin parameter to a high value. However, if price fluctuations occur both up and down, simply using a margin to ignore smaller downward movements will allow total LP supply to inflate when prices fluctuate upward, while downward fluctuations remain unaccounted for. Over time, this mismatch between the total LP supply and actual locked liquidity `D` could diminish the real value of LP tokens or even risk pool insolvency.

This is classified as **high** severity because a downward price move can create a partial or complete denial-of-service, and ignoring repeated negative fluctuations inflates total LP supply without real backing.

Recommendation

We recommend adjusting the system so that negative fluctuations are accounted for by proportionally reducing `LPToken.totalSupply`, rather than relying solely on a high yield margin. This ensures impermanent losses caused by decreased exchange rate are distributed among LP holders and prevents LP supply from growing unchecked when both upward and downward price movements occur.

Client's Commentary

> Client: We will implement the latest recommended solution which combines the buffer management and governance-controlled negative rebasing with optional emergency withdrawals in a locked state

> Mixbytes: While the main problem addressed, the issue is partially fixed:
>
> - Accounting for the removed amount from the buffer has been added when the exchange rate fluctuates. This should prevent inflation of LPToken shares due to exchange rate changes.

- A centralized `distributeLoss` function is refactored to handle negative rebasing of `LPToken`, distributing losses among users when the buffer is insufficient to cover losses.
- If exchange rate decreases fall within `feeErrorMargin` or `yieldErrorMargin`, inflation of LPToken shares remains possible. However, since these margins are no longer required to be large enough to prevent entire operation reverts, a new Low 8 issue has been created to address this specific case.
- The emergency withdrawal function has not been implemented. A new Medium 2 issue has been created specifically for the lack of emergency withdrawals.
- The original issue is marked as fixed.

H-2	Arbitrage on Oracle-Driven Price Updates
Severity	High
Status	Fixed in 08b66f14

Description

This issue has been identified within the `swap`, `mint`, and `redeem` operations of the `SelfPeggingAsset` contract.

The pool's effective price relies on external oracle exchange rates, causing it to instantly reflect any oracle-based updates. Attackers who can detect or anticipate these rate changes (e.g., via mempool monitoring or privileged oracle access) can profit by:

- **Swap before** the oracle change, then **swap again** after the change, or
- **Mint before** the new rate is applied, then **redeem** once the oracle rate shifts in a favorable direction.

Because the amplification coefficient `A` can be large, the pool's price is highly sensitive, allowing attackers to execute substantial swaps with minimal slippage. Attempting to compensate by raising fees poses a trade-off between competitiveness and effective front-run prevention. Even then, sudden high oracle-driven price changes (for instance, 0.5% or greater) can exceed those fees, making arbitrage profitable.

The issue is classified as **high** severity because high oracle-driven changes can be exploited to consistently extract value, particularly when large trading volumes incur minimal slippage.

Recommendation

We recommend implementing dynamic fees that temporarily increase when the pool becomes imbalanced, similar to the approach in Curve's stableswap-ng design. This mitigates arbitrage opportunities caused by sudden oracle updates while keeping fees lower during normal market conditions. Once balances are stable again, fees can revert to a baseline level.

Client's Commentary

> Client: We will implement a dynamic fees mechanism.
>
> Mixbytes: The dynamic fees mechanism is improperly implemented.
>
> - Fees are not applied in the `redeemSingle` function.
> - In the `redeemMulti` and `mint` functions, dynamic fees are calculated using `oldD` and `newD`, meaning fees depend on the amount of liquidity minted or redeemed rather than the pool's imbalance before or

> after the operation. Instead, use the average of the old and new reserve values of the added or removed token and the ideal reserve value in a balanced pool for each affected token.
> - In the `swap` function, dynamic fees are calculated using the old reserve values before operation. This means users who imbalance the pool after a swap do not incur additional fees, while subsequent users do. Instead, use the average of the old and new reserve values of tokens `i` and `j` to reflect the post-swap imbalance.
>
> Client: We will make changes in the mentioned functions to fix the implementation of dynamic fees, closely following the Curve's model.

H-3	Buffer Drainage Through Repeated Rebase Calls Due to Stale State Variables
Severity	High
Status	Fixed in 55c07d34

Description

This issue has been identified within the `rebase` function of the `SelfPeggingAsset` contract.

When the old invariant `D` (`oldD`) exceeds the new invariant `D` (`newD`), the function calls `poolToken.removeTotalSupply(oldD - newD, true, true)` to reduce the total supply. However, it fails to update the contract's internal state variables `balances` and `totalSupply` to reflect this change. As a result, if `D` decreases, these variables remain unchanged while the buffer is reduced. Since `rebase` is an external, unprotected function that anyone can call, an attacker could repeatedly invoke it to drain the buffer entirely, operating on outdated state data with each call.

The issue is classified as **high** severity because it enables unauthorized depletion of the protocol's buffer, potentially leading to significant financial loss or disruption of the pool's operations.

Recommendation

We recommend updating the `balances` and `totalSupply` state variables after the `poolToken.removeTotalSupply` call when `oldD > newD` to ensure they accurately reflect the adjusted state. Additionally, consider implementing access controls (e.g., restricting `rebase` to authorized users) or a cooldown mechanism to prevent repeated exploitation.

Client's Commentary

> Yes, we will need to update balances and total supply after reducing the total supply of LP token. We will update the `balances` to the newly calculated `_balances` and `totalSupply` to `newD`.

H-4	Flawed Precision and Scaling Logic in `price()`
Severity	High
Status	Acknowledged

Description

The `price()` function in `ChainlinkCompositeOracleProvider` begins with a 36-decimal fixed-point accumulator (`highPrecisionPrice = PRECISION`). Each feed is then folded into that value, but two subtle arithmetic mistakes distort the final price:

1. **Inverted feeds (`isInverted == true`)**
   The code multiplies the accumulator by `(10**assetDecimals) * (10**feed.decimals()) / feedPrice`.
   Because both exponents are reapplied for every inverted feed, the precision scale increases by `assetDecimals` on each iteration, corrupting the effective price by several orders of magnitude, or exceeding the 256-bit limit causing overflow.

2. **Non-inverted feeds (`isInverted == false`)**
   Here, the accumulator is multiplied by `feedPrice` and divided by `10**currentDecimals`, that keeps the scale of price. When all feeds are non-inverted, the accumulator retains its initial 36-decimal offset while the final division by `PRECISION` truncates all fractional data, returning a whole-number price.

Combined, these errors can cause large under- or over-flows, creating a clear path for profitable arbitrage in outer contracts. Hence, the issue is classified as **High** severity.

Recommendation

We recommend adopting a uniform fixed-point workflow that keeps every intermediate value on the same scale:

```
uint256 rate = cfg.isInverted
    ? SCALING_FACTOR.mulDiv(
        10 ** config.feed.decimals(),
        uint256(feedPrice)
    )
    : uint256(feedPrice) * (
        10 ** (SCALING_DECIMALS - config.feed.decimals())
    );

highPrecisionPrice = highPrecisionPrice.mulDiv(
    rate,
    SCALING_FACTOR
);
```

and only converts to final decimals once, at the very end:

```
return highPrecisionPrice.mulDiv(10 ** this.decimals(), SCALING_FACTOR);
```

Client's Commentary

> MixBytes: Contract was removed from the audit scope

2.3 Medium

M-1	Changing `A` Abruptly Without Timelock
Severity	Medium
Status	Fixed in b5b92783

Description

This issue has been identified within the `updateA` function of the `SelfPeggingAsset` contract.

The parameter `A` can be changed abruptly in a single transaction, which could significantly alter the price curve. If the pool is imbalanced, an unexpected `A` increase can create arbitrage opportunities, exposing liquidity providers to sudden risks or losses.

This is classified as **medium** severity because abrupt changes may lead to notable losses for honest LPs who have little time to react.

Recommendation

We recommend implementing a public timelock or a gradual-style mechanism. This would give liquidity providers time to exit or rebalance before a major curve adjustment, mitigating sudden changes and reducing unintended impermanent loss risks.

Client's Commentary

> We will update A through governance process so there will be enough time for LPs to remove liquidity.

M-2	Centralization Risks in Emergency Scenarios
Severity	Medium
Status	Acknowledged

Description

This issue has been identified within the overall design of the `SelfPeggingAsset` contract.

The contract lacks an emergency redemption mechanism for users to withdraw their tokens when losses from exchange rate fluctuations exceed the buffer capacity, causing the pool to become locked. In such scenarios, users' funds remain inaccessible until the owner executes the `distributeLoss` function to adjust the total supply and release the pause. This design introduces a centralization risk, as access to funds depends solely on the owner's intervention, which could be delayed or withheld if the owner is unavailable or unwilling to act.

The issue is classified as **medium** severity because it could result in funds being temporarily or permanently inaccessible, creating potential for misuse or prolonged disruption in emergency situations.

Recommendation

We recommend implementing an emergency redemption function that allows users to withdraw their tokens proportionally during a paused state, potentially with a penalty or under specific conditions to deter abuse while enhancing user autonomy.

Client's Commentary

> It was decided after internal discussion and discussion with Mixbytes that when the buffer is not enough we will distribute the loss using governance process. So there is no need to implement a separate emergency withdrawal mechanism.

M-3	Unclaimed LP Tokens in Zap Contract
Severity	Medium
Status	Fixed in 9ad26539

Description

This issue has been identified within the `zapOut` function of the `Zap` contract.

When the `proportional` parameter is set to `false`, LP tokens may remain on the contract after the operation completes. An attacker could exploit this by passing custom SPA and WLP addresses to steal these tokens or by leveraging excessive approvals within the contract. This vulnerability arises because the function does not return residual LP tokens to the user or validate the provided SPA and WLP addresses.

The issue is classified as **medium** severity because it could lead to loss of funds under specific conditions, though exploitation requires deliberate manipulation by an attacker.

Recommendation

We recommend modifying the `zapOut` function to return any remaining LP tokens to the user after the operation.

Client's Commentary

> We will fix it by repaying the remaining LP shares in `zapOut` functions.

M-4	Syncing Issue During Ramping
Severity	Medium
Status	Fixed in 9ad26539

Description

This issue has been identified within the `syncRamping` modifier of the `SelfPeggingAsset` contract.

If the `syncRamping` modifier is triggered between the start and end of an `A` value ramping period, the `A` value may not be updated as intended, leading to inconsistencies in the pool's state. A similar problem could occur if the `rampAController` address is changed, as the modifier does not account for such transitions.

The issue is classified as **medium** severity because it could disrupt the intended ramping mechanism, potentially affecting pool stability and user expectations, though it requires specific timing to take effect.

Recommendation

We recommend revising the `syncRamping` modifier to always fetch the current `A` value from the `rampAController` using `getCurrentA()` and compare it with the stored value. This ensures proper synchronization and updates to the `A` parameter, regardless of when the sync occurs or if the controller changes.

Client's Commentary

> Will fix the `syncRamping` modifier

M-5	Inconsistent A Value in Calculations
Severity	Medium
Status	Fixed in 9ad26539

Description

This issue occurs in the `SelfPeggingAsset` contract, where calculations use a locally stored `A` parameter instead of fetching the current value from the `rampAController`.

During ramping periods, this leads to inconsistencies because the stored `A` value does not reflect the controller's intended updates. As a result, balance and invariant *D* computations rely on the outdated `A`, while the ramped `A` is only applied to buffer updates in the `_syncTotalSupply` function when the pool is imbalanced.

The issue is classified as **medium** severity because it may cause miscalculations that disrupt pool behavior and user interactions during `A` value transitions.

Recommendation

We recommend updating the contract to consistently retrieve the current `A` value from the `rampAController` by calling `getCurrentA()` (or a similar method) in all functions where `A` is used for calculations. This ensures that dynamic updates are accurately reflected.

Client's Commentary

> Will fix it by retrieving latest value using `getCurrentA()`

M-6	Potential DOS via External Token Transfer Affecting `assert` Checks
Severity	Medium
Status	Fixed in 08b66f14

Description

This issue has been identified in multiple locations (`zapIn`, `zapOut`, and `zapOutSingle`) within the `Zap` contract.
Specifically, an `assert` statement checks that the contract holds zero tokens after an operation. If an attacker (or any user) transfers even a minimal amount (e.g., 1 wei) of the same token directly to the contract, the `assert` can fail and revert the transaction, causing a potential Denial of Service (DOS).

Additionally, while appropriate for internal invariants, `assert` should generally not be used for flow control. `assert` reverts consume all remaining gas and convey less explicit error messaging compared to `require`.

The issue is classified as **medium** severity because it can disrupt normal contract usage by reverting user operations if the contract's token balance unexpectedly becomes non-zero.

Recommendation

We recommend removing these `assert` statements.

Client's Commentary

> Will remove asserts to address the issue.

M-7	`Zap.zapOut` Uses LP shares Instead of Tokens
Severity	Medium
Status	Fixed in 730e8a19

Description

This issue has been identified within the `zapOut` function of the `Zap` contract.

```
uint256 lpTokenShares = ILPToken(lpToken).getSharesByPeggedToken(lpAmount);
IERC20(lpToken).forceApprove(spa, lpTokenShares);
```

The receiving `SelfPeggingAsset` contract expects `lpAmount` in token units, not in share units. When the share-to-token ratio drifts from 1 : 1, the SPA either reverts (too few tokens approved) or over-burns the user's balance. This blocks withdrawals and breaks Zap's single-transaction UX.

The issue is classified as *medium* severity because it disrupts normal withdrawals once the ratio changes, but does not break pool solvency.

Recommendation

We recommend approving and passing the raw `lpAmount` (token units) to the `SelfPegingAsset` contract.

Client's Commentary

> Will implement changes following recommendation

2.4 Low

L-1	Unbounded Fee Parameters
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the `initialize()`, `setMintFee`, `setRedeemFee`, and `setSwapFee` functions of the `SelfPeggingAsset` contract, and in the `bufferPercent` parameter of the `LPToken` contract.

Although each fee is capped individually, the owner can still set all fees close to their maximum, resulting in very high cumulative fees.

The issue is classified as **low** severity because it does not present an immediate exploit but may create unfavorable conditions for liquidity providers and traders.

Recommendation

We recommend implementing a combined check so that total fees never exceed a reasonable threshold (for example, 10% for mint/swap/redeem fees and 50% for `bufferPercent`), ensuring fair usage of the protocol.

Client's Commentary

> The fee will be set by the pool owner and we expect it to be set correctly. Setting high fees will anyways won't wrong LPs therefore making the pool unusable.
> Later updates to the fees will be done via governance process so its safe.

L-2	Name and Symbol Do Not Reference Underlying LP Token
Severity	Low
Status	Fixed in b5b92783

Description

This issue has been identified in the constructor of `WLPToken.sol`. The default `name` and `symbol` (`"Wrapped LP Token"`, `"wlpToken"`) do not reference the name or symbol of the underlying `lpToken`, which can confuse users about the relationship between the wrapped token and its underlying asset.

The issue is classified as **low** severity because it impacts clarity rather than functionality or security.

Recommendation

We recommend setting the `name` and `symbol` of `WLPToken` to include the underlying LP token's identity, improving clarity for users and integrators.

Client's Commentary

> We will pass the token name and symbol in the initialiser

L-3	Use of String Instead of `bytes` for Oracle Signatures
Severity	Low
Status	Fixed in b5b92783

Description

This issue has been identified in `SelfPeggingAssetFactory`, where new `OracleExchangeRateProvider` contracts receive function signatures as strings (`tokenAFunctionSig` or `tokenBFunctionSig`). Using strings instead of `bytes` can introduce extra overhead and complexity in function selector handling.

The issue is classified as **low** severity because it concerns best practices rather than an immediate vulnerability.

Recommendation

We recommend passing the precomputed `bytes` function selector instead of a string to reduce overhead and improve clarity.

Client's Commentary

> We will make this enhancement

L-4	Public `allowances` Mapping
Severity	Low
Status	Fixed in b5b92783

Description

This issue has been identified in `LPToken.sol`, where `allowances` is declared as a public mapping. The contract already provides an `allowance()` function, making the public mapping redundant and potentially cluttering the interface.

The issue is classified as **low** severity because it mainly affects code clarity and standard compliance rather than security.

Recommendation

We recommend making the `allowances` mapping private or internal, relying on the `allowance()` function to provide any necessary external access.

Client's Commentary

> We will fix this

L-5	Fee-On-Transfer Tokens Not Supported
Severity	Low
Status	Acknowledged

Description

This issue has been identified in the contract's token-handling logic (for example, in the `mint`, `swap`, and `redeem` functions). The contract assumes standard ERC20 transfers without accounting for fee-on-transfer tokens (e.g., USDT and USDC, which currently have fees set at 0%). This can result in smaller received amounts than expected, causing unpredictable behavior or potential loss of funds.

The issue is classified as **low** severity because it only affects scenarios where a token has a fee-on-transfer mechanism.

Recommendation

We recommend either disallowing fee-on-transfer tokens in the `swap` and `mint` functions by reverting when the received balance does not match the expected amount, or calculating the invariants after the transfer is performed, using the actual balance increase.

Client's Commentary

> We will implement the check as we don't want to support fee on transfer tokens.

L-6	Misleading Function Names and Variable
Severity	Low
Status	Fixed in b5b92783

Description

The `getPendingYieldAmount` function name of `SelfPeggingAsset` contract suggests it returns only "pending yield," but it actually provides updated balances and a new invariant value.
Additionally, the `totalSupply` variable of the `SelfPeggingAsset` contract represents invariant `D`, not the usual definition of "total supply" of `SelfPeggingAsset` contract.

The issue is classified as **low** severity because it does not break functionality but can confuse developers or integrators.

Recommendation

We recommend renaming the function to something like `getUpdatedBalancesAndD()` or clarifying the returned values in comments. Likewise, consider renaming `totalSupply` to `D`.

Client's Commentary

> We will fix this

L-7	`rebase()` Reverts If `D` Does Not Increase
Severity	Low
Status	Fixed in b5b92783

Description

In the `rebase()` function of the `SelfPeggingAsset` contract, if the new invariant (`newD`) does not exceed the old invariant (`oldD`), a zero `addBuffer` call triggers a revert. This can cause automated scripts or user calls to fail when no net increase occurs.

The issue is classified as **low** severity because it does not pose a direct exploit risk but can disrupt normal operations when `D` remains unchanged.

Recommendation

We recommend checking for a zero increase in `D` and returning early if no rebase is needed, thereby avoiding an unnecessary revert.

Client's Commentary

> We will check for a zero increase in D and return early if no rebase is needed, therefore avoiding an unnecessary revert.

L-8	Inconsistent Share Allocation Due to Exchange Rate Fluctuations Within Margin
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the `collectFeeOrYield` function of the `SelfPeggingAsset` contract.

When the exchange rate fluctuates within the defined margin (`feeErrorMargin` or `yieldErrorMargin`), small decreases in the invariant `D` are ignored, and no adjustment is made to the total supply. However, if the rate subsequently increases beyond the old value (`newD > oldD`), the function triggers `pool.addTotalSupply`, increasing the total supply and thus inflating the `LPToken` share price. This behavior leads to an unintended and inconsistent increase of `LPToken` total supply, potentially skewing the pool's economic balance.

The issue is classified as **low** severity because it may cause small, unintended increases in the total supply under specific conditions, but it does not directly result in significant financial loss or major operational disruptions, due to limited margin value.

Recommendation

We recommend revising the logic in `collectFeeOrYield` to handle exchange rate fluctuations more consistently, such as by tracking cumulative changes over time or dynamically adjusting the margin based on historical volatility, to prevent unintended supply increases.

Client's Commentary

> Normally decrease in exchange rate is not common unless it's manually decreased by a third party protocol or bad debt is realised among users. In these case the decrease will be more than margin so it will be handled by buffer or governance. So we can acknowledge this issue.

L-9	RampA Logic Not Handling Very Low `initialA`
Severity	Low
Status	Fixed in 9ad26539

Description

This issue has been identified within the `rampA` function of the `RampAController` contract.

If `initialA` is extremely low (e.g., set to `1`), the code's checks may cause the function to revert due to the ratio constraints. This leads to an inability to initiate ramping from a very small `initialA`.

The issue is classified as **low** severity because it restricts certain use cases rather than posing an immediate security risk.

Recommendation

We recommend either restricting the `initialA` to equal to 1 or adjusting the logic to handle very small `initialA` values gracefully, for example by allowing a special case or appropriately relaxing the ratio constraints when `initialA` is set to 1.

Client's Commentary

> Will address this by adding conditional logic

L-10	Lack of `spa` and `wlp` Address Validation in Zap Contract
Severity	Low
Status	Acknowledged

Description

The `zapIn` and `zapOut` functions in the `Zap` contract accept `spa` (SelfPeggingAsset) and `wlp` (wrapped LP token) addresses provided by the caller without confirming they are valid contracts.

Attackers can pass arbitrary contracts as `spa` and `wlp`, creating an opportunity to grant an allowance of any token from the `Zap` contract to these malicious addresses.

Recommendation

We recommend implementing a whitelist of verified `spa` and `wlp` contracts.

Client's Commentary

> Client: We will fix the issue by reseting the approval of `spa` and `wlp` address in zap functions.
> MixBytes: Resetting approvals at the end of zap functions does not resolve the problem, as tokens residing in the contract can still be transferred to arbitrary external addresses provided by the user. The main issue here is that the owner-restricted `recoverERC20` function is meaningless, since any user can sweep the tokens from the contract using this workaround.
> Client: We will remove `recoverERC20` and ownership from Zap contract and acknowledge the issue since whitelisting addresses is not an option for us.

L-11	Unnecessary `getCurrentA()` Calls in Non-View Functions
Severity	Low
Status	Fixed in 08b66f14

Description

This issue has been identified within the non-view functions of the `SelfPeggingAsset` contract, where the code calls `getCurrentA()` even though the `A` value is already updated inside the `syncRamping` modifier:

- `mint()`
- `swap()`
- `redeemSingle()`
- `redeemMulti()`
- `donateD()`

Since syncRamping internally synchronizes the local `A` with `getCurrentA()`, additional explicit calls to `getCurrentA()` in non-view contexts become redundant and add unnecessary overhead.

Recommendation

We recommend removing unnecessary calls to `getCurrentA()` inside these non-view functions that are already using the `syncRamping` modifier.

Client's Commentary

> We will remove unnecessary `getCurrentA()` calls.

L-12	Missing Check for Matching `A` Parameter
Severity	Low
Status	Fixed in 08b66f14

Description

In the `setRampAController()` function of the `SelfPeggingAsset` contract, there is no check ensuring that the `initialA` parameter in the designated `RampAController` matches the current `A` value in `SelfPeggingAsset`. If these values are not aligned, the internal `A` within the pool can abruptly jump to a new level.

This issue is classified as `low` severity because it does not directly lead to loss of funds under correct owner actions. However, if the owner accidentally or incorrectly configures this parameter, a sudden and unfavorable change in `A` could be exploited by arbitrageurs.

Recommendation

We recommend checking that the controller's `initialA` value matches the `A` stored in the `SelfPeggingAsset` contract, when setting a new `RampAController`.

Client's Commentary

> We will add the check for setting new Ramp A controller initialA.

L-13	Missing Zero-Address Check for Every Feed Entry
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the constructor of the `ChainlinkCompositeOracleProvider` contract.

Only the first feed is validated:

```
if (i == 0 && address(_configs[i].feed) == address(0)) revert InvalidFeed();
```

Subsequent feeds can be the zero address, causing `latestRoundData()` to revert at runtime.

Recommendation

We recommend validating every feed.

Client's Commentary

> Client: Will implement changes following recommendation
> MixBytes: The address check was removed completely, instead of adding validation for each feed as recommended.
> MixBytes: Contract was removed from the audit scope

L-14	Initialized `assetDecimals` May Become Stale
Severity	Low
Status	Fixed in 730e8a19

Description

This issue has been identified within the constructor of the `ChainlinkCompositeOracleProvider` contract.

`assetDecimals` is copied once from `feed.decimals()`. If the chainlink oracle's aggregator is upgraded or replaced, the stored value may become wrong, producing mis-scaled prices.

Recommendation

We recommend querying `feed.decimals()` on every call or adding an owner-only function to update `assetDecimals` through a timelock.

Client's Commentary

> Will implement changes following recommendation

L-15	Inefficient and Overcomplicated `decimals()` Implementation
Severity	Low
Status	Fixed in 730e8a19

Description

This issue has been identified within the `decimals` function of the `ChainlinkCompositeOracleProvider` contract.

The function iterates through all configured feeds, overwriting the `_decimals` value in each iteration and ultimately returning the value from the last feed. This logic is unnecessary, especially when multiple feeds are configured, and adds complexity without providing meaningful benefit. Additionally, calling `decimals()` on each feed increases gas usage and contract size unnecessarily.

Recommendation

We recommend simplifying the implementation by setting the effective decimals once in the constructor and returning the value directly in the `decimals` function.

Client's Commentary

> We'll address this finding by optimizing the `decimals()` function to directly use the last config

L-16	Rounding-Error Amplification When Chaining Inverted Price Feeds
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the `price` function of the `ChainlinkCompositeOracleProvider` contract.

Each time an inverted feed is processed, the intermediate result is truncated to the current decimal scale.

Repeating the operation three times with 8-decimal feeds, assuming their inverted price is around 0.001 yields a cumulative error of about:

```
(1 - 10^-5)^3 ≈ 1 - 3 × 10^-5
```

(~0.003 %). While acceptable for typical 8–18-decimal feeds, lower-precision feeds or more chained operations could magnify the error.

Recommendation

We recommend performing all multiplications in a higher-precision accumulator (e.g., 1e36) with taking into account possible overflow errors that could arise.

Client's Commentary

> Client: Will implement changes following recommendation
> MixBytes: See High 4
> MixBytes: Contract was removed from the audit scope

L-17	Volatility Fee Ratio Calculated from The Wrong Denominator
Severity	Low
Status	Fixed in 730e8a19

Description

This issue has been identified within the `_updateMultiplierForToken` function of the `SelfPeggingAsset` contract.

The contract calculates the volatility ratio as follows:

```
uint256 ratio = (diff * FEE_DENOMINATOR) / oldRate;
```

where `diff = |newRate - oldRate|`. The ratio is incorrectly calculated using only the `oldRate` as a denominator, regardless of whether the rate has increased or decreased. Specifically, when the rate increases significantly, dividing by the smaller `oldRate` artificially inflates the ratio.

For example, if the rate triples from 1 to 3, the correct ratio should be:

```
(3 - 1) / 3 ≈ 0.66
```

However, the current code calculates it as:

```
(3 - 1) / 1 = 2
```

This results in an exaggerated volatility fee, unfairly increasing costs for users after upward rate movements.

The issue is classified as *low* severity as it does not critically affect protocol operations but results in slightly inflated fees during specific rate movements.

Recommendation

We recommend computing the volatility ratio using the higher of the two rates:

```
uint256 ratio = (diff * FEE_DENOMINATOR) / max(oldRate, newRate);
```

Client's Commentary

> Will implement changes following recommendation

L-18	First Trader After Idle Period Pays Accumulated Volatility Fees
Severity	Low
Status	Fixed in 8ae30e3d

Description

This issue has been identified within the `_updateMultiplierForToken` function of the `SelfPeggingAsset` contract.

Currently, the volatility multiplier increases only when the `_updateMultiplierForToken` function is triggered by user interaction. If no interaction occurs for an extended period while the external token price significantly changes, the first subsequent trade triggers a multiplier spike. This first trader unfairly incurs the entire accumulated volatility fee due to historical price changes, rather than just the volatility arising from their specific trade.

As an illustrative scenario:

1. The pool remains inactive for several days (e.g. due to admin's pause), during which the token price doubles.
2. The first swap executed after this inactive period experiences an unusually high volatility fee, even if the price remains stable around the new value at the trade time.

This results in a poor user experience and creates a potential denial-of-service vector, as users may avoid trading after prolonged inactivity periods.

Recommendation

We recommend applying a continuous linear decay mechanism to the `candidateMultiplier` as well, ensuring new multipliers decay proportionally over time in the same way as current multipliers.

Client's Commentary

> Client: We will implement configurable threshold periods for skipping fee calculations for inactive pools as we agreed in Telegram

> MixBytes: The implemented fix adds `block.timestamp - lastActivity > rateChangeSkipPeriod` check in `_currentMultiplier` to reset the multiplier to `FEE_DENOMINATOR` after inactivity.
> However, it doesn't update `st.lastRate` or reset `st.raisedAt` when returning from inactivity. The first trader after inactivity will still face volatility spikes because `_updateMultiplierForToken` will

> compare the current `newRate` against outdated `st.lastRate` from before the inactive period.
> To properly fix this, also reset `st.lastRate` to the current `exchangeRateProviders[i].exchangeRate()` and update `st.raisedAt` to `block.timestamp` in the beginning of the `_updateMultiplierForToken` function when returning from an inactive period.

L-19	`decimals()` Can Be a Constant Value
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the `decimals()` function of the `ChainlinkCompositeOracleProvider` contract.

The function returns the decimal count of the last entry in the `configs` array. If the final config has a few decimals (e.g., 8), this can lead to precision loss. Using a fixed value for `decimals()` improves performance and makes the contract logic more clear.

Recommendation

We recommend defining a fixed decimal value for the composite oracle in the constructor (for example, 18 or `36 - assetDecimals`) and returning this constant from `decimals()`.

Client's Commentary

> MixBytes: contract was removed from the audit scope

L-20	Keeper Contract Lacks a Function to Call `LPToken.setSymbol`
Severity	Low
Status	Fixed in db85b66e

Description

The issue has been identified in the `Keeper` contract.
The comment on the `LPToken.setSymbol` function states:

```
* @notice This function is called by the keeper to set the token symbol.
```

However, the `Keeper` contract does not implement a call to the `setSymbol` function to manage the `tokenSymbol` value.

Recommendation

We recommend implementing a `setSymbol` function in the `Keeper` contract that calls `lpToken.setSymbol` with appropriate access control.

Client's Commentary

> We will implement `setSymbol` in the keeper contract for Governor.

L-21	Unused Errors and Event
Severity	Low
Status	Fixed in db85b66e

Description

The following errors and event are declared but not used in the contracts:

- `UnauthorizedAccount` in the `Keeper` contract
- `PoolAlreadySet` in the `LPToken` contract
- `UnauthorizedAccount`, `ZeroAddress`, `CannotChangeControllerDuringRamp`, `InitialANotMatchCurrentA`, `AccountIsZero`, `PastBlock`, `SameTokenInTokenOut`, `ImbalancedPool` in the `SelfPeggingAsset` contract
- `PoolSet` event in the `LPToken` contract

Recommendation

We recommend either removing these errors and the event if they are unnecessary or using them in the appropriate places where the corresponding checks or logs are relevant.

Client's Commentary

> We will remove the unused event and errors.

L-22	`getMintAmount()` Misreports Fees
Severity	Low
Status	Fixed in e861622b

Description

In `SelfPeggingAsset.getMintAmount()`, the function sums per-token fees to report `feeAmount`. This does not match the actual fee charged, which should be the difference between the pre-fee and post-fee mint amounts. Because of non-linear effects and rounding in `_getD()`, the reported fee can be inaccurate.

The bug affects reporting and user expectations but not pool accounting, so severity is **low**.

Recommendation

We recommend reporting `feeAmount` as `preFeeMint - postFeeMint`, similar to `getRedeemMultiAmount()`, instead of summing individual token fees.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes