Smart Contract Code Review And Security Analysis Report

CUSTOMER Minted

SCOPE Institutional Vault

DATE 28.02.2026

Made in Germany by softstack GmbH

Public review redacted copy - implementation excerpts omitted

Table of Contents

1. Disclaimer 4
2. About the Project and Company 5
3. Vulnerability & Risk Level 6
4. Auditing Strategy and Techniques Applied 7
5. Metrics 8
   5.1 Tested Contract Files 8
   5.2 Call Graph 9
   5.3 Source Lines & Risk 10
   5.4 Capabilities 10
   5.5 Dependencies / External Imports 11
   5.6 Source Units in Scope 12
6. Scope of Work 14
   6.1 Findings Overview 14
   6.2 Manual and Automated Vulnerability Test 16
      6.2.1 Vault Insolvency From globalTotalAssets() vs Local Balance Mismatch 16
      6.2.2 Silent Treasury Fallback in globalTotalAssets() Enables Share Price Manipulation 17
      6.2.3 Canton Share Sync Rate Limit Compounds Exponentially 18
      6.2.4 First Canton Sync Bypass Accepts Arbitrary Share Value at Zero Supply 19
      6.2.6 _fullDeleverage() May Fail to Fully Unwind Due to Interest-Driven LTV Drift 21
      6.2.7 No Bad Debt Handling Creates Permanent Zombie Debt 21
      6.2.8 emergencyClosePosition() Reverts When Swap Fails Trapping User Funds 22
      6.2.10 TVL Unit Mismatch Between SY-Denominated Value and USD Threshold 24
      6.2.11 lastLnImpliedRate Is a Spot Value That Can Be Manipulated 24
      6.2.12 Deposit During Rollover Window Orphans Old PT and Corrupts ptBalance 25
      6.2.13 distributeYield() and receiveInterest() Are Sandwich-Attackable 27
      6.2.14 emergencyWithdraw Can Drain USDC Backing accumulatedFees 28
      6.2.16 setReserveBps() Lacks Allocation Consistency Validation 30
      6.2.17 emergencyWithdrawAll() Does Not Pause the Contract 31
      6.2.19 setParams() Does Not Accrue Interest Before Rate Change 32

6.2.20 minSupplyRateRequired Is Set But Never Enforced 33
6.2.21 withdraw() Deducts Full Principal Regardless of Actual Amount Freed 34
6.2.22 No Oracle Cardinality Check Causes TWAP Query to Revert 35
6.2.23 emergencyWithdraw Emits Wrong ptBalance Value in Event 36
6.2.24 resetLastKnownPrice() Skips Staleness and Circuit Breaker Checks 37
6.2.25 Legacy withdraw() Does Not Update lastRecordedValue, Causing Fee Drift 38
7. Executive Summary 40
8. About the Auditor 41
9. Glossary 42

Public review redacted copy - implementation excerpts omitted

1. Disclaimer

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

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

Version / Date	Description
0.1 (15.02.2026)	Layout
0.5 (20.02.2026)	Manual + Automated Security Testing
1.0 (28.02.2026)	Final document
1.1 (28.02.2026)	Re-check

2. About the Project and Company

Company: MintedAssociates Corp
Address: 511 South DuPont Highway Ste 10, Dover, Delaware 19901, USA
Website: https://minted.app/
Product: mUSD
Network: Canton Network

Minted is building mUSD, a Canton-native institutional stable settlement token for treasury, collateral, repo, and tokenized securities settlement workflows. mUSD is designed as a non-yield-bearing settlement instrument for institutional use on Canton.

smUSD is a separate Institutional Yield Vault structure for approved participants seeking access to Canton-native and real-world yield strategies. smUSD is structurally separate from mUSD and is not a payment stablecoin.

Key capabilities include:
- Canton-native stable settlement token design for institutional cash movement.
- Tokenized securities collateral workflows for approved institutional participants.
- Privacy-preserving authorization and workflow controls through DAML.
- Vault, collateral, lending, liquidation, treasury management, and yield strategy components.

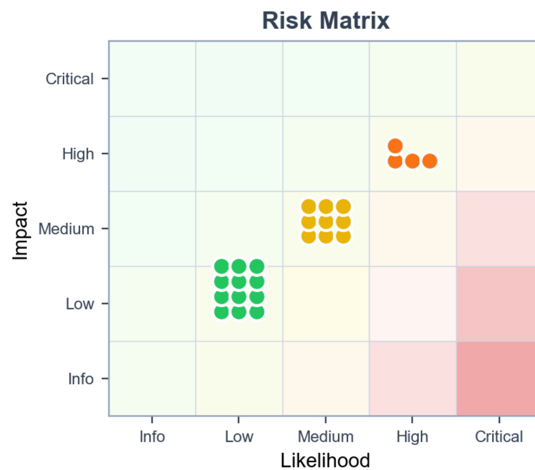
The smart contracts under audit relate to vault and lending infrastructure supporting Minted's institutional mUSD architecture.

Public review redacted copy - implementation excerpts omitted

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.1.

Level	CVSS Score	Vulnerability	Required Action
CRITICAL	9.0 – 10.0	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
HIGH	7.0 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
MEDIUM	4.0 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
LOW	0.1 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
INFO	0.0	A vulnerability that has informational character but is not affecting any of the code.	An observation that does not determine a level of risk.



Public review redacted copy - implementation excerpts omitted

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

The auditing process follows a routine series of steps:

Code review that includes the following:

- Review of the specifications, sources, and instructions provided to softstack to make sure we understand the size, scope, and functionality of the smart contract.
- Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Comparison to specification, which is the process of checking wh er the code does what the specifications, sources, and instructions provided to softstack describe.

Testing and automated analysis that includes the following:

- Test coverage analysis, which is the process of determining wh er the test cases are actually covering the code and how much code is ex ised when we run those test cases.
- Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.

Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness,

recommendations, and research.

Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

Public review redacted copy - implementation excerpts omitted

5. Metrics

The metrics section gives the reader an overview of the size, quality, flows and capabilities of the codebase.

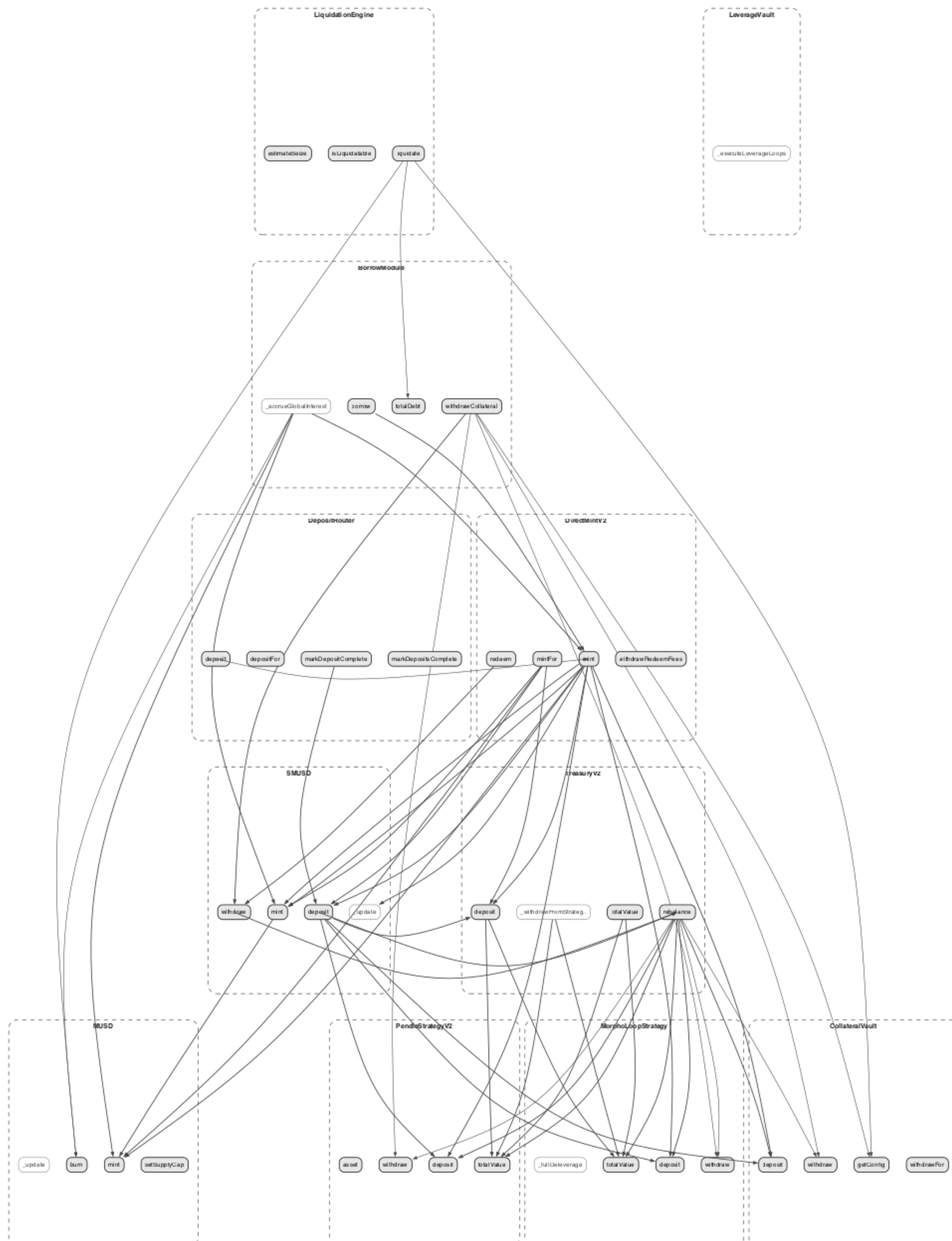
5.1 Tested Contract Files

Source: https://github.com/luthatdude/Minted-mUSD-Canton
Commit: `393fd96f322674b25dd4b0c21f593f0874b86211`

File	Fingerprint (MD5)

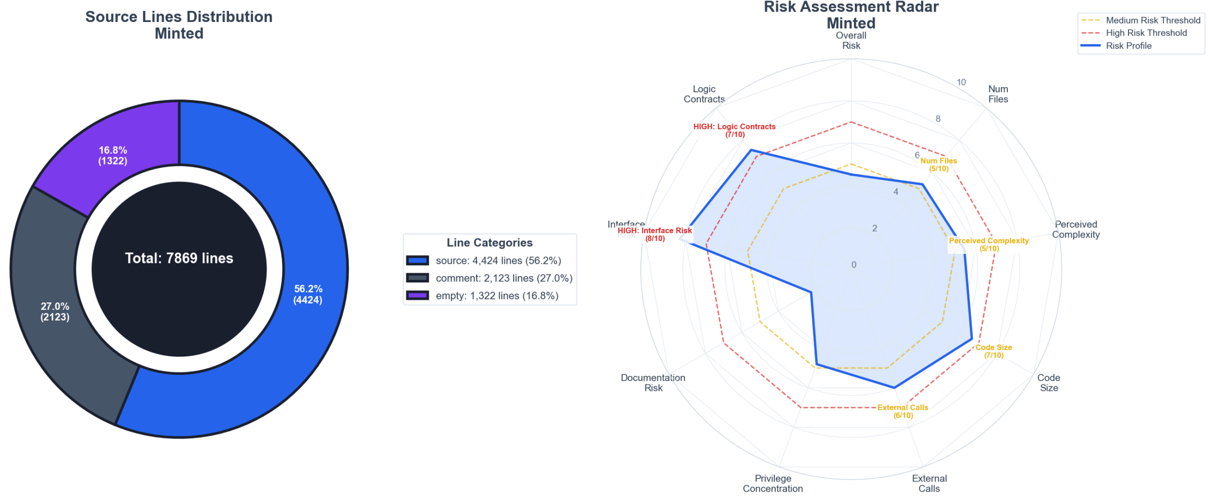
Public review redacted copy - implementation excerpts omitted

5.2 Call Graph



Public review redacted copy - implementation excerpts omitted

5.3 Source Lines & Risk

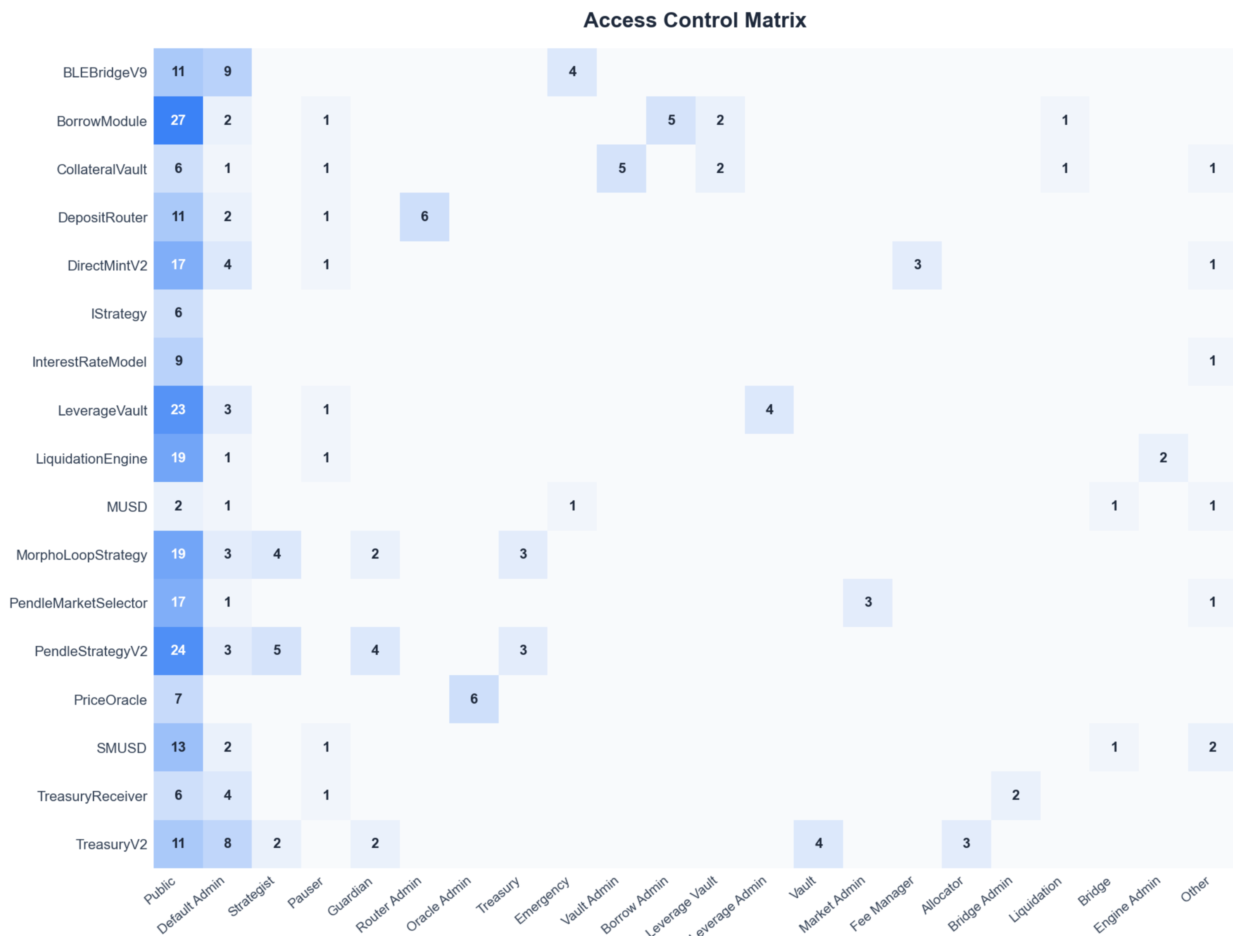


5.4 Capabilities

Module	Functions	Public	Restricted	Key Roles
V9	24	11	13	DEFAULT_ADMIN, EMERGENCY
BorrowModule	38	27	11	BORROW_ADMIN, LEVERAGE_VAULT, DEFAULT_ADMIN
CollateralVault	17	6	11	VAULT_ADMIN, LEVERAGE_VAULT, BORROW_MODULE
DepositRouter	20	11	9	ROUTER_ADMIN, DEFAULT_ADMIN, PAUSER
DirectMintV2	26	17	9	DEFAULT_ADMIN, FEE_MANAGER, MINTER
IStrategy	6	6	0	-
InterestRateModel	10	9	1	RATE_ADMIN
LeverageVault	31	23	8	LEVERAGE_ADMIN, DEFAULT_ADMIN, PAUSER
LiquidationEngine	23	19	4	ENGINE_ADMIN, PAUSER, DEFAULT_ADMIN
MUSD	6	2	4	COMPLIANCE, , EMERGENCY
MorphoLoopStrategy	31	19	12	STRATEGIST, TREASURY, DEFAULT_ADMIN
PendleMarketSelector	22	17	5	MARKET_ADMIN, PARAMS_ADMIN, DEFAULT_ADMIN

Public review redacted copy - implementation excerpts omitted

Module	Functions	Public	Restricted	Key Roles
PendleStrategyV2	39	24	15	STRATEGIST, GUARDIAN, TREASURY
PriceOracle	13	7	6	ORACLE_ADMIN
SMUSD	19	13	6	DEFAULT_ADMIN, YIELD_MANAGER, INTEREST_ROUTER
TreasuryReceiver	13	6	7	DEFAULT_ADMIN, _ADMIN, PAUSER
TreasuryV2	30	11	19	DEFAULT_ADMIN, VAULT, ALLOCATOR



5.5 Dependencies / External Imports

Import Path	Version
@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable	^5.0.0

Import Path	Version
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable	^5.0.0
@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable	^5.0.0
@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable	^5.0.0

5.6 Source Units in Scope

File	Instructions	Functions	Lines	nSLOC	Comments
DepositRouter	16	21	422	229	126
PendleMarketSelector	17	26	534	266	170
PriceOracle	11	13	256	168	50
TreasuryReceiver	13	13	295	156	95
CollateralVault	12	17	299	169	82
InterestRateModel	2	10	276	142	96
LeverageVault	28	36	770	467	168
DirectMintV2	25	26	324	195	70
MUSD	6	7	104	70	18

Public review redacted copy - implementation excerpts omitted

File	Instructions	Functions	Lines	nSLOC	Comments
LiquidationEngine	21	23	274	150	79
File	Instructions	Functions	Lines	nSLOC	Comments
TreasuryV2	23	34	1000	534	291
SMUSD	19	23	329	167	105
BorrowModule	33	46	830	481	204
MorphoLoopStrategy	16	36	811	440	235
PendleStrategyV2	31	46	830	500	187
IStrategy	6	6	40	9	25
Total	279	383	7394	4143	2001

6. Scope of Work

The Minted team provided vault and lending smart contracts supporting mUSD and smUSD infrastructure. The audit focuses on validating security, compliance, and robustness.

6.1 Findings Overview

23

Applicable Issues

HIGH (4) MEDIUM (8) LOW (11)

No	Title	Severity	Status
6.2.1	Vault Insolvency From globalTotalAssets() vs Local Balance Mismatch	HIGH	FIXED
6.2.2	Silent Treasury Fallback in globalTotalAssets() Enables Share Price Manipulation	HIGH	FIXED
6.2.3	Canton Share Sync Rate Limit Compounds Exponentially	HIGH	FIXED
6.2.4	First Canton Sync Bypass Accepts Arbitrary Share Value at Zero Supply	HIGH	FIXED

Public review redacted copy - implementation excerpts omitted

No	Title	Severity	Status
6.2.6	_fullDeleverage() May Fail to Fully Unwind Due to Interest-	MEDIUM	ACKNOWLEDGED

No	Title	Severity	Status
6.2.6	_fullDeleverage() May Fail to Fully Unwind Due to Interest-Driven LTV Drift	MEDIUM	ACKNOWLEDGED
6.2.7	No Bad Debt Handling Creates Permanent Zombie Debt	MEDIUM	FIXED
6.2.8	emergencyClosePosition() Reverts When Swap Fails Trapping User Funds	MEDIUM	FIXED
6.2.10	TVL Unit Mismatch Between SY-Denominated Value and USD Threshold	MEDIUM	FIXED
6.2.11	lastLnImpliedRate Is a Spot Value That Can Be Manipulated	MEDIUM	FIXED
6.2.12	Deposit During Rollover Window Orphans Old PT and Corrupts ptBalance	MEDIUM	FIXED
6.2.13	distributeYield() and receiveInterest() Are Sandwich-Attackable	MEDIUM	FIXED
6.2.14	emergencyWithdraw Can Drain USDC Backing accumulatedFees	LOW	FIXED
6.2.16	setReserveBps() Lacks Allocation Consistency Validation	LOW	FIXED
6.2.17	emergencyWithdrawAll() Does Not Pause the Contract	LOW	FIXED
6.2.19	setParams() Does Not Accrue Interest Before Rate Change	LOW	FIXED
6.2.20	minSupplyRateRequired Is Set But Never Enforced	LOW	ACKNOWLEDGED
6.2.21	withdraw() Deducts Full Principal Regardless of Actual Amount Freed	LOW	ACKNOWLEDGED
6.2.22	No Oracle Cardinality Check Causes TWAP Query to Revert	LOW	FIXED
6.2.23	emergencyWithdraw Emits Wrong ptBalance Value in Event	LOW	ACKNOWLEDGED
6.2.24	resetLastKnownPrice() Skips Staleness and Circuit Breaker Checks	LOW	FIXED
6.2.25	Legacy withdraw() Does Not Update lastRecordedValue, Causing Fee Drift	LOW	FIXED

Public review redacted copy - implementation excerpts omitted

6.2 Manual and Automated Vulnerability Test

6.2.1 Vault Insolvency From globalTotalAssets() vs Local Balance Mismatch

DESCRIPTION

The SMUSD vault computes share-to-asset conversions usingglobalTotalAssets(), which derives fromTreasury.totalValue() * 1e12and includes all capital deployed across yield strategies. However, the actual withdrawal mechanism inherited from OpenZeppelin's 4626 can only transfer mUSD from the vault's localbalanceOf(address(this)).

When the treasury's total value exceeds the vault's local mUSD balance (the normal operating state since the treasury deploys capital into strategies), the vault becomes structurally insolvent. It promises more mUSD per share than it holds. Users callingredeem()get a revert because the vault tries to transfer more than its local balance. This creates a first-come-first-served bank run where the first withdrawer drains the vault and remaining users are locked out.

cause a revert.

CODE LOCATION

RECOMMENDATION

Override_withdraw()to pull from treasury when local balance is insufficient:

Also overridemaxWithdraw()andmaxRedeem()to cap at local liquidity.

Public review redacted copy - implementation excerpts omitted

6.2.2 Silent Treasury Fallback in globalTotalAssets() Enables Share Price Manipulation

DESCRIPTION

balance). This creates a massive share price discontinuity with no event emission, no circuit breaker, and no cached fallback value.

When the treasury manages 5M USDC but the vault only holds 500K mUSD locally, a treasury failure causesglobalTotalAssets()to drop from 5M to 500K (a 10x drop). An attacker who observes or triggers the treasury failure deposits at the crashed price, receiving roughly 10x more shares than normal. After the treasury recovers and the 24-hour cooldown expires, the attackerredeems at the restored price, extracting value from existing depositors.

CODE LOCATION

RECOMMENDATION

Remove the silent fallback and let the revert propagate:

Public review redacted copy - implementation excerpts omitted

If availability is needed, use a cached last-known-good value with a staleness check instead of falling back to the local balance.

6.2.3 Canton Share Sync Rate Limit Compounds Exponentially

DESCRIPTION

sudden large changes in a single transaction, the rate limit compounds across consecutive syncs because each sync's cap is computed against the current (already inflated)cantonTotalShares. A compromised _ROLE can inflate canton shares by 3.18x in 24 hours, 10.2x in 48 hours, and 129x in 100 hours.

Since_convertToAssets()usesglobalTotalShares()(shares + Canton shares) as the denominator, inflated canton shares dilute every depositor's per-share asset claim. There is no cumulative cap, no absolute maximum ratio between Canton and shares, and no governance escalation threshold.

CODE LOCATION

RECOMMENDATION

Add a cumulative change cap over a rolling time window:

Also add an absolute maximum ratio between Canton and shares.

Public review redacted copy - implementation excerpts omitted

6.2.4 First Canton Sync Bypass Accepts Arbitrary Share Value at Zero Supply

DESCRIPTION

The first-sync guard insyncCantonShares()is designed to cap initial Canton shares at 2x shares. But whentotalSupply() == 0(no deposits yet), the guard degrades

provides zero protection.

A _ROLE holder can inject any arbitrary value ascantonTotalShareson a freshly deployed vault. Setting it to a huge value either bricks the vault permanently (all deposits revert with overflow in_convertToShares) or enables phantom Canton shares to steal over 99% of all future yield from depositors.

The irrecoverable nature comes from the 5% per-hour rate limit on subsequent syncs. Reducing from a massive value to a sane one would take hundreds of hours.

CODE LOCATION

RECOMMENDATION

Require deposits to exist before the first Canton sync:

Public review redacted copy - implementation excerpts omitted

Public review redacted copy - implementation excerpts omitted

6.2.6 _fullDeleverage() May Fail to Fully Unwind Due to Interest-Driven LTV Drift

DESCRIPTION

_fullDeleverage() is capped at MAX_LOOPS * 2 = 10 iterations. In Morpho Blue, collateral does not earn interest, but borrow debt accrues continuously. Over time, the effective LTV drifts upward toward LLTV as debt grows while collateral stays constant.

When the effective LTV exceeds safeLtv (targetLtvBps - safetyBufferBps), _maxWithdrawable()

progress for all remaining iterations and exits silently with the position partially unwound.

withdrawAll() then sets totalPrincipal = 0, creating an accounting desync where the strategy believes it is empty while funds remain trapped in Morpho with accruing debt and no recovery mechanism. emergencyDeleverage() uses the same _fullDeleverage() and also stalls.

RECOMMENDATION

Increase the iteration limit. Add a fallback that uses LLTV-based withdrawal when safeLtv is too restrictive. Add a require check after the loop to verify the position is fully unwound (borrowShares == 0 and collateral == 0) before setting totalPrincipal to 0.

6.2.7 No Bad Debt Handling Creates Permanent Zombie Debt

DESCRIPTION

When a position becomes severely und   ollateralized (collateral value less than total debt), the

value minus penalty. The remaining debt is never cleared and has no recovery mechanism anywhere in the protocol.

Public review redacted copy - implementation excerpts omitted

LiquidationEngine or BorrowModule. The zombie debt remains in totalBorrows permanently, accruing interest on debt backed by nothing, inflating the utilization rate, and raising interest rates for all other borrowers.

RECOMMENDATION

remaining debt when a position has zero collateral. Socialize the loss across the protocol (e.g., through a reserve fund or pro-rata deduction from suppliers). This matches the approach used by Aave V3 (handleBadDebt), Compound, and Liquity.

6.2.8 emergencyClosePosition() Reverts When Swap Fails Trapping User Funds

DESCRIPTION

_swapCollateralToMusd() which reverts on swap failure. If the Uniswap pool has insufficient liquidity, the token is paused or blacklisted, or the pool does not exist, the emergency close itself reverts, defeating its purpose and permanently trapping user funds.

emergency close to revert on swap failure.

No alternative recovery path exists. emergencyWithdraw() only affects tokens held in LeverageVault (which holds 0 since collateral goes to CollateralVault). CollateralVault has no emergencyWithdraw() function.

Public review redacted copy - implementation excerpts omitted

RECOMMENDATION

the cost of the emergency recovery.

DESCRIPTION

thanfromorto. A blacklisted operator can still calltransferFromto move tokens between non-blacklisted addresses, as long as they have an existing approval. This defeats the purpose of the blacklist for compliance, since a sanctioned entity can continue to facilitate token movements.

OpenZeppelin's transferFromflow calls_spendAllowanc en_update. Since_updateonly

CODE LOCATION

RECOMMENDATION

OverridetransferFromor add amsg.send heck inside_updatefor non-mint transfers:

Public review redacted copy - implementation excerpts omitted

6.2.10 TVL Unit Mismatch Between SY-Denominated Value and USD Threshold

DESCRIPTION

againstminTvlUsd(a threshold denominated in USD). For example, if the SY token is wst (18 decimals, price ~$2000), a market with 500 wst ($1M) would havetvlSy = 500e18. IfminTvlUsdis set to1_000_000e6($1M in 6-decimal USD), the comparisontvlSy >= minTvlUsdbecomes500e18 >= 1_000_000e6, which passes even though both represent $1M. The comparison is meaningless because the units are incompatible.

This can cause markets with insufficient real TVL to pass the filter (false positives) or adequate markets to be rejected (false negatives), depending on the SY token's price and decimals.

CODE LOCATION

RECOMMENDATION

ConverttvlSyto a USD value before comparing againstminTvlUsd. Use the Pendle oracle or a Chainlink feed to get the SY token price:

6.2.11 lastLnImpliedRate Is a Spot Value That Can Be Manipulated

Public review redacted copy - implementation excerpts omitted

DESCRIPTION

is a spot value that reflects the most recent trade and can be manipulated via a large swap within the same block. An attacker can inflate or deflate the implied rate by making a large trade right beforeselectBestMarket()is called, causing the selector to pick a suboptimal market. The attacker then profits from this mispricing by trading in the selected market.

The contract already uses the Pendle TWAP oracle for thegetPtToSyRatecall, showing awareness of manipulation risks. But the implied rate scoring bypasses this protection entirely by using the raw spot value.

CODE LOCATION

RECOMMENDATION

Use a TWAP-based implied rate instead of the spotlastLnImpliedRate. Pendle

observation from the market's oracle:

6.2.12 Deposit During Rollover Window Orphans Old PT and Corrupts ptBalance

Public review redacted copy - implementation excerpts omitted

DESCRIPTION

switches all market pointers to a new Pendle market without first redeeming the existing PT tokens from the old market. This permanently orphans the old PT tokens and corrupts theptBalanceaccounting variable.

The correct rollover implementation inrollToNewMarket()first redeems old PT, then selects the new

After the auto-rollover,ptBalancecontains the sum of old orphaned PT count plus new PT count, but the contract only holds new PT. This causeswithdrawAll()to revert (tries to redeem more PT than exists),totalValue()to report inflated NAV, and individual withdrawals to fail once cumulative withdrawals exceed actual PT held.

CODE LOCATION

RECOMMENDATION

Redeem old PT before selecting the new market indeposit():

Or remove auto-rollover fromdeposit()entirely and require an explicitrollToNewMarket()call.

Public review redacted copy - implementation excerpts omitted

6.2.13 distributeYield() and receiveInterest() Are Sandwich-Attackable

DESCRIPTION

BothdistributeYield()andreceiveInterest()transfer mUSD lump-sum into the SMUSD vault, instantly inflatingtotalAssets()and the share price for all holders. Shares become yield-eligible immediately upon deposit. The 24-hour cooldown blocks atomic same-block sandwich attacks but does not prevent strategic yield timing attacks.

An attacker who deposits a large amount 25 hours before a predictable yield distribution captures a proportional share of the yield. Since yield distributions are permissioned (role-gated), their timing is predictable. A whale depositing equal to the vault total captures 50% of the yield despite holding shares for only 25 hours, while a long-term staker who held for 30 days receives only 50%.

CODE LOCATION

RECOMMENDATION

Implement yield streaming instead of lump-sum distribution:

This spreads yield over time so short-term depositors cannot capture a disproportionate share.

Public review redacted copy - implementation excerpts omitted

6.2.14 emergencyWithdraw Can Drain USDC Backing accumulatedFees

DESCRIPTION

emergencyWithdraw() allows DEFAULT_ADMIN_ROLE to withdraw any token including USDC, but it never reads or writes accumulatedFees. There is also no admin setter for accumulatedFees anywhere in the contract. After an emergency withdrawal of USDC, the accumulatedFees state variable still reflects the old balance, creating a permanent accounting desync.

When withdrawFees() is called after such a drain, it reads the stale accumulatedFees value, sets it to 0, and attempts to transfer that amount. Since the actual USDC balance is insufficient, the transfer reverts. This permanently blocks fee collection with no recovery path.

Public review redacted copy - implementation excerpts omitted

RECOMMENDATION

In emergencyWithdraw(), when the token is USDC, reduce accumulatedFees by the withdrawn amount (capping at zero). This keeps the accounting variable in sync with the actual contract balance.

DESCRIPTION

insiderebalance()at Lines 842 and 870. If any active strategy'stotalValue()reverts (due to an external protocol pause, upgrade, or bug), the entirerebalance()transaction reverts. This blocks rebalancing for all strategies, not just the failing one.

297),_withdrawFromStrategies()loops (Lines 567, 587), andremoveStrategy()(Line 753) all havetry/

The three integrated DeFi protocols (Pendle, Morpho, Sky) all have known pause and upgrade mechanisms that could trigger this.

CODE LOCATION

RECOMMENDATION

Public review redacted copy - implementation excerpts omitted

6.2.16 setReserveBps() Lacks Allocation Consistency Validation

DESCRIPTION

check wh erreserveBpsplus the sum of all active strategytargetBpsexceeds 10000 (100%). BothaddStrategy()andupdateStrategy()enforce this invariant, butsetReserveBps()bypasses it.

When the invariant is broken (e.g., strategies total 9000 bps and reserve is set to 3000 bps = 12000 bps total),addStrategy()andupdateStrategy()revert withTotalAllocationInvalid, locking strategy

of total, but actual reserve is only 10%, so no funds are deployed to under-allocated strategies. New deposits route 30% to reserve instead of the intended 10%, reducing depositor yield.

CODE LOCATION

RECOMMENDATION

Add the same allocation consistency check used byaddStrategy()andupdateStrategy():

Public review redacted copy - implementation excerpts omitted

6.2.17 emergencyWithdrawAll() Does Not Pause the Contract

DESCRIPTION

pause the contract or deactivate strategies. Between the emergency withdrawal and a separate manualpause()call, a vault deposit can arrive. Since the contract is unpaused and strategies remain active,depositFromVault()executes and_autoAllocate()re-deploys funds to all active strategies, including the potentially compromised one that triggered the emergency.

This effectively nullifies the emergency action. At minimum a 1-block race window exists between emergency withdrawal and manual pause.

CODE LOCATION

RECOMMENDATION

Public review redacted copy - implementation excerpts omitted

6.2.19 setParams() Does Not Accrue Interest Before Rate Change

Public review redacted copy - implementation excerpts omitted

DESCRIPTION

When rate parameters are updated via InterestRateModel.setParams() or the model is swapped via BorrowModule.setInterestRateModel(), there is no mechanism to accrue pending interest under the

rates retroactively to the entire elapsed period since lastGlobalAccrualTime, including time that passed before the rate change.

A rate increase causes borrowers to be ov harged for the pre-change period. A rate decrease causes the protocol and suppliers to lose interest they should have earned.

RECOMMENDATION

public accrueGlobalInterest() function. Add a callback in InterestRateModel.setParams() that triggers accrual before overwriting parameters.

6.2.20 minSupplyRateRequired Is Set But Never Enforced

DESCRIPTION

TheminSupplyRateRequiredstate variable is set in the constructor and can be updated by admin viasetMinSupplyRate(), but it is never actually checked anywhere in the contract.

validate wh er the current supply rate meets this minimum threshold before depositing. This means the rate protection is entirely absent despite appearing to be configured.

CODE LOCATION

Public review redacted copy - implementation excerpts omitted

RECOMMENDATION

Add a supply rate check at the start of_leverage():

Or remove the variable entirely if rate checks are not intended, to avoid giving a false sense of security.

6.2.21 withdraw() Deducts Full Principal Regardless of Actual Amount Freed

DESCRIPTION

calls_deleverage()to free the collateral. However,_deleverage()may free less than requested due to iteration limits or liquidity constraints. Despite this,withdraw()deducts the fullprincipalToWithdrawfromprincipalDeposited, creating a mismatch between the tracked principal and the actual funds in the strategy. Over multiple partial withdrawals,principalDepositedcan underflow or reach zero while significant capital remains deployed.

CODE LOCATION

Public review redacted copy - implementation excerpts omitted

RECOMMENDATION

Scale the principal deduction by the ratio of actual to requested withdrawal:

6.2.22 No Oracle Cardinality Check Causes TWAP Query to Revert

DESCRIPTION

TWAP_DURATION)without first checking wh er the market's oracle has sufficient observation cardinality to support the requested TWAP duration. By default, all new Pendle markets haveobservationCardinality = 1, which is insufficient for a 900-second TWAP (requires cardinality of 85).

this call, the revert propagates up through_getMarketInfo()togetValidMarkets()andselectBestMarket(). A single uninitialized market in the whitelist causes the entire market selection to revert, blocking all other valid markets as well.

The contract readsobservationCardinalityfrom_storage()at Line 318 but discards it without any validation.

CODE LOCATION

Public review redacted copy - implementation excerpts omitted

RECOMMENDATION

Also validate oracle readiness during whitelisting:

6.2.23 emergencyWithdraw Emits Wrong ptBalance Value in Event

DESCRIPTION

TheEmergencyWithdrawevent always emitsptRedeemed = 0becauseptBalanceis read after_redeemPt()has already set it to zero. Inside_redeemPt(),ptBalance -= ptAmountruns before the

balance)executes,ptBalanceis always zero regardless of how much PT was actually redeemed.

This causes off-chain monitoring, dashboards, and treasury accounting to record zero PT redeemed during emergency withdrawals, making post-incident forensics unreliable.

CODE LOCATION

RECOMMENDATION

CacheptBalancebefore the mutation:

6.2.24 resetLastKnownPrice() Skips Staleness and Circuit Breaker Checks

DESCRIPTION

the latest Chainlink round data without checking staleness, round completeness, or the circuit breaker deviation threshold. The only validation isprice > 0. This means an admin can anchor the circuit breaker to a stale or manipulated price, which then affects all subsequentgetPrice()calls that compare against this anchor.

If the Chainlink feed is stale or returning an outdated price at the time of the reset, the anchor becomes incorrect. Future legitimate prices that deviate significantly from this bad anchor will trigger the circuit breaker unnecessarily, causing a denial of service for all protocol operations that depend on the oracle.

CODE LOCATION

Public review redacted copy - implementation excerpts omitted

RECOMMENDATION

Apply the same staleness, round completeness, and deviation checks used ingetPrice():

6.2.25 Legacy withdraw() Does Not Update lastRecordedValue, Causing Fee Drift

DESCRIPTION

Neitherwithdraw()norwithdrawToVault()updatelastRecordedValueafter the withdrawal. When these

updating the baseline. The withdrawal then reducestotalValue()butlastRecordedValuestays at the stale pre-withdrawal level.

On the next accrual (when the interval passes),currentValue < lastRecordedValue, so no fees are accrued even though real yield was earned. For example, if 5,000 USDC of yield accrues and then a 50,000 USDC withdrawal happens within the interval, the accrual sees965,001 < 1,010,001and records zero fees. The protocol loses 2,000 USDC (40% fee on 5,000) per occurrence.

CODE LOCATION

Public review redacted copy - implementation excerpts omitted

RECOMMENDATION

AddlastRecordedValue = totalValue()at the end of both withdrawal functions:

Public review redacted copy - implementation excerpts omitted

7. Executive Summary

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

Overall, the current-scope public copy identifies a total of 23 applicable issues, classified as follows:

- No critical issues were found
- 4 high severity issues were found
- 8 medium severity issues were found
- 11 low severity issues were found
- No informational issues were found

The Minted team addressed all fixed findings in scope. Remaining applicable findings were acknowledged by the team.

The current-scope public copy comprises 4,143 normalized source lines of code across 16 files.

8. About the Auditor

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

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

softstack offers industry-leading smart contract audit services. Committed to evolving with their clients'

serves.

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

Public review redacted copy - implementation excerpts omitted

9. Glossary

Term	Definition
CVSS	Common Vulnerability Scoring System
DeFi	Decentralized Finance
	Request for Comments
	Virtual Machine
LTV	Loan-to-Value ratio
TVL	Total Value Locked
TWAP	Time-Weighted Average Price

Public review redacted copy - implementation excerpts omitted