softstack

Smart Contract Code Review And Security Analysis Report

CUSTOMER	Frgmnt
SCOPE	fUSD Protocol
DATE	20.10.2026

Made in Germany by softstack GmbH

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 Source Lines & Risk 16

5.3 Capabilities 16

5.4 Dependencies / External Imports 20

5.5 Source Units in Scope 22

6. Scope of Work 28

6.1 Findings Overview 28

6.2 Manual and Automated Vulnerability Test 30

6.2.1 Fresh-deployment script leaves the deployer as owner of all three Transparent proxy ProxyAdmins 30

6.2.2 execTransaction missing nonReentrant: cross-function reentrancy via callback-exempt manager 32

6.2.3 executeOperation missing nonReentrant: partial NAV reentrancy during Aave flashloan unwind 33

6.2.4 onMorphoFlashLoan missing nonReentrant: partial NAV reentrancy during Morpho flashloan unwind 34

6.2.5 Fee unit-of-account mismatch in FundCalculationLibrary fee summation 35

6.2.6 Hub-wide liquidity overstatement in AaveV4SpokeAssetGuard: per-Spoke cross-pool capping absent 36

6.2.7 withdrawalSlippageBps dual-use: spot-TWAP deviation gate and LP output slippage share one threshold 38

6.2.8 UniswapV3SelectiveAssetGuard inherits withdrawalSlippageBps dual-use 39

6.2.9 AaveLendingPoolAssetGuard: unconfigured collateral swap route causes permanent pool-wide DoS 40

6.2.10 UniswapV3RouterGuard does not validate amountOutMinimum on manager swaps 41

6.2.11 Contracts that the documents call protocol-owned are owned by the pool manager Safe, so the manager can move the pool's USDC out through a Morpho market it controls 42

6.2.12 The Uniswap router guard validates only the first and last token of a multi-hop path, and PoolLogic.execTransaction is unlocked, so a manager or trader can mint unbacked fees and rewards 44

6.2.13 The manager can act as any user on deposit, stake, unstake and immediate withdrawal by listing a contract as a callback sender 46

6.2.14 Liquidity provision through the Uniswap guard has no loss accounting beyond a price band of about 1 percent, so a trader can sandwich the pool's own mints repeatedly 47

6.2.15 Reference attester service lets the requesting user choose which assets a plan draws from 49

6.2.16 A token-only plan in a pool with a debt-bearing position leaves accountedAssets below the gross NAV, so the next accrual mints a fee and rewards from the exiter's haircut 50

6.2.17 A dust queued request lets any fUSD holder block plan draws of a deposit asset in queue mode 52
6.2.18 The attested-only upgrade path leaves validated protections inactive and cannot be completed by the core upgrade script 53
6.2.19 A floored NAV hides a direct-leg over-draw when the deficit comes from another guard 55
6.2.20 The emergency stop is not independent of the manager, or arrives after a rotation matures, in configurations the repository's own scripts produce 57
7. Executive Summary 59
8. About the Auditor 60
9. Glossary 61

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
0.2 (05.10.2026)	Report creation
0.5 (06.10.2026)	Manual and automated security testing
1.0 (12.10.2026)	Findings submitted
1.5 (16.10.2026)	Remediation reviewed
2.0 (20.10.2026)	Final document

2. About the Project and Company

Company: GRANIT FINANCE SAS

Address: 60 Rue Bernard Giraudeau, 34080 Montpellier, France

Website: https://frgmnt.fi/

Twitter (X): https://x.com/frgmnt

GitHub: https://github.com/frgmnt-dao

2.1 Project Overview

Frgmnt is a decentralized stablecoin issuer that brings the yields of the best stablecoins on the market together in a single token, fUSD. Its aim is to give holders the performance of stablecoin yield farming in one accessible product, without the need to monitor, manage or chase yields themselves.

Today most stablecoin issuers keep the yield generated by the funds they hold, and the higher returns available in decentralized finance remain complex, costly and largely reserved for experienced participants. Frgmnt addresses this by automatically allocating funds to the strongest stablecoin yield opportunities and returning that value to its users rather than to intermediaries.

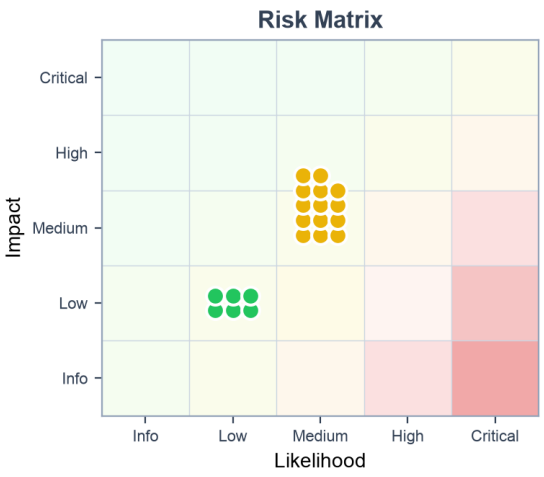
The protocol selects where funds are placed on three criteria: the best-performing yields, the security of the underlying protocols, and the liquidity of the tokens involved. fUSD is designed to stay pegged to the US dollar, while holders who stake it share in the yield the protocol earns.

Frgmnt is built for anyone who wants passive, dollar-denominated income from decentralized finance in a simple and transparent form, combining performance, security and liquidity in a single token.

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.
INFORMATIONAL	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.



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 whether 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 whether the test cases are actually covering the code and how much code is exercised 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, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

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

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 repositories and commits for each contract in scope:

Contract	Source	Commit
Audited codebase	https://github.com/frgmnt-dao/Frgmnt_fUSD	b4640647afe7761183257d4a729ab28663ea05ac
Remediation (pull request	https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/1	01a08516382cd99858ae7449c2213b5dff83259e
Remediation (pull request	https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5	ceff2c727871baf0dba9f3834338f90d25e0a4b0

File	Fingerprint (MD5)
./Frgmnt_fUSD/contracts/contracts/FrgmntUserActions.sol	25e97633fba9e1b4ff5b870f50a07d5e
./Frgmnt_fUSD/contracts/contracts/Governance.sol	b90076250f1f6311962cc8821eedb8fa
./Frgmnt_fUSD/contracts/contracts/Managed.sol	b3cf42a0e50111b946a427123f5a3a87
./Frgmnt_fUSD/contracts/contracts/PoolLogic.sol	dba8726e0f12b1d250523cea5d375ea3
./Frgmnt_fUSD/contracts/contracts/PoolManagerLogic.sol	7bf60c6153b0c9e96baadea5b08eb15d
./Frgmnt_fUSD/contracts/contracts/Timelock.sol	7a41696ea07269ee30b63f6e6e74f73d
./Frgmnt_fUSD/contracts/contracts/TokenLogic.sol	f27b8fa9dbd9fd103f91ea585932d52d
./Frgmnt_fUSD/contracts/contracts/WithdrawalEscrow.sol	0655bd32aa59441eaa52b8a90ad0f3ef
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/AaveLendingPoolAssetGuard.sol	81d12e27c7198d6fbbdb5605a70410fc
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/AaveV3LendingPoolSelectiveAssetGuard.sol	7686cdb43fcb67d1546ed5772d524347
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/AaveV4SpokeAssetGuard.sol	e069b8d91490aa1e19885c243607c08f

File	Fingerprint (MD5)
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/AaveV4SpokeSelectiveAssetGuard.sol	32f8222fcc71ab3908cfc1eccbaa38e3
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/AaveV4TokenizationAssetGuard.sol	708e27ea28d2ab336fee8ae20522995a
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/ClosedAssetGuard.sol	c9ba44f4614ba0b3c59ecb8fff7e3485
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/ERC20Guard.sol	d0c69e28ea2415024741affa0e79981c
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/MorphoBlueLendingPoolAssetGuard.sol	0ba09e344737a9aaab0c969054720e0e
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/MorphoBlueLendingPoolSelectiveAssetGuard.sol	6d144068638ddf468ad048f3b186713f
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/MorphoVaultV2AssetGuard.sol	00df41636fd241ec5355399517cc0b8e
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/UniswapV3AssetGuard.sol	f60f48e01941ca64b60d26054f99e4f5
./Frgmnt_fUSD/contracts/contracts/guards/assetGuards/UniswapV3SelectiveAssetGuard.sol	b45ab414bc0b0665e3c85b2f9d983e0d
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/AaveLendingPoolGuardV3.sol	5e5835a7cdd1c6fb28cb7ef7a9fd40a6
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/AaveV4SpokeContractGuard.sol	08a800e8385e0eba8c088262d5af8cb3
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/AaveV4SpokeManager.sol	e175898b8c766bd172a79c44e56c4bf7
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/AaveV4TokenizationContractGuard.sol	397a9a7d6d845a15a00b755c83308eac
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/AaveV4TokenizationManager.sol	e22d6325cb34a58273297df6d24959f6
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/MerklRewardClaimGuard.sol	b7149d61f6d2b5e717f04db16d68caf2
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/MorphoBlueContractGuard.sol	7775d0bda4409f9655f5713e831fd3f8
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/MorphoBlueManager.sol	64749571f83495e2151f9303f32278b1

File	Fingerprint (MD5)
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/MorphoVaultV2ContractGuard.sol	924eea35dc8d810c5e3f09c5b82e4533
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/MorphoVaultV2Manager.sol	d94df6fd2c6f16429e7b6c6a1f1c0b24
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/uniswapV3/UniswapV3NonfungiblePositionGuard.sol	e650e8e97fd2caebc1a7912ff276ce45
./Frgmnt_fUSD/contracts/contracts/guards/contractGuards/uniswapV3/UniswapV3RouterGuard.sol	f599429c03f0fa05983471e2095129a3
./Frgmnt_fUSD/contracts/contracts/interfaces/IAaveV4SpokeManager.sol	d57b8b510cc7a0ac1e5d5e8e7bb4011e
./Frgmnt_fUSD/contracts/contracts/interfaces/IAaveV4TokenizationManager.sol	bd5d298ad30bfd2abe93560204a378af
./Frgmnt_fUSD/contracts/contracts/interfaces/IAggregatorV3Interface.sol	92d7956b87abae83e0a52a955ccc9e08
./Frgmnt_fUSD/contracts/contracts/interfaces/IAggregatorV3InterfaceWithOwner.sol	649875ad22f2b64994a9e2e746e90da4
./Frgmnt_fUSD/contracts/contracts/interfaces/IAssetHandler.sol	750e8245cb1cc3b081898340c1105edb
./Frgmnt_fUSD/contracts/contracts/interfaces/IERC20.sol	bde3677ae98ed06dbc6f5da757f7ee6d
./Frgmnt_fUSD/contracts/contracts/interfaces/IERC20Extended.sol	c5cea03c38f098e3660ec91226ba34c5
./Frgmnt_fUSD/contracts/contracts/interfaces/IERC721Enumerable.sol	678df80095d46aae7f2dd55d2dc9d61f
./Frgmnt_fUSD/contracts/contracts/interfaces/IGovernance.sol	764b0a0d18982cd1607655bc45950273
./Frgmnt_fUSD/contracts/contracts/interfaces/IHasAssetInfo.sol	1b8bb29da5bd1a1329f2609ce9143544
./Frgmnt_fUSD/contracts/contracts/interfaces/IHasDaoInfo.sol	a06d87c93f7217b65879b5a6e14dc0e7
./Frgmnt_fUSD/contracts/contracts/interfaces/IHasFeeInfo.sol	ff1b3812dff544ade75b5ba2ac93da9a
./Frgmnt_fUSD/contracts/contracts/interfaces/IHasGuardInfo.sol	703bd31a9971f8bd2a2fe9dc10dd0953
./Frgmnt_fUSD/contracts/contracts/interfaces/IHasOwnable.sol	7ae56a2279e66b35d9625b19ec93e5c9
./Frgmnt_fUSD/contracts/contracts/interfaces/IHasPausable.sol	9fedad38a1edacf5c2757240d2a054d4
./Frgmnt_fUSD/contracts/contracts/interfaces/IHasSupportedAsset.sol	5fcd277bed1fd344bed3d21bc6fc131e
./Frgmnt_fUSD/contracts/contracts/interfaces/IManaged.sol	61ef410971a6048411250de3bea13610

File	Fingerprint (MD5)
./Frgmnt_fUSD/contracts/contracts/interfaces/IMorphoBlueManager.sol	9390fd9f7ac021e22e933a172fe3fc43
./Frgmnt_fUSD/contracts/contracts/interfaces/IMorphoFlashLoanCallback.sol	e25ad828c93ecfc6b61ade57a181885e
./Frgmnt_fUSD/contracts/contracts/interfaces/IMorphoVaultV2.sol	21c7f326e8eb61f5a6543a3c5c955410
./Frgmnt_fUSD/contracts/contracts/interfaces/IMorphoVaultV2Manager.sol	4a7194f71ab95f6c41ed39de6f103964
./Frgmnt_fUSD/contracts/contracts/interfaces/IPoolFactory.sol	060e323627c305022a5100b2bb60e3e3
./Frgmnt_fUSD/contracts/contracts/interfaces/IPoolLogic.sol	5ca6c19d07542df8e363a4e9bf8464ee
./Frgmnt_fUSD/contracts/contracts/interfaces/IPoolManagerLogic.sol	ad1bcbdf4f4ace749f879132d215269a
./Frgmnt_fUSD/contracts/contracts/interfaces/ITransactionTypes.sol	282d2dfea1369568f54700ad373020d3
./Frgmnt_fUSD/contracts/contracts/interfaces/IV3SwapRouter.sol	36000725d03f5b7309212c4b686b33cb
./Frgmnt_fUSD/contracts/contracts/interfaces/IWETH.sol	e07a85a4ca2a382280fd87f3ff73d456
./Frgmnt_fUSD/contracts/contracts/interfaces/IWithdrawalEscrow.sol	972717a091c9526ff62a30667b9b2f95
./Frgmnt_fUSD/contracts/contracts/interfaces/aave/IAaveProtocolDataProvider.sol	2fa4f57c87b4e99189831e6fcdf1d3e5
./Frgmnt_fUSD/contracts/contracts/interfaces/aave/IFlashLoanReceiver.sol	6cecc2a90abf12b4bccc2ee45fb17e27
./Frgmnt_fUSD/contracts/contracts/interfaces/aave/v3/DataTypes.sol	ee8667e20f3ea0c17546f6fc0b165f03
./Frgmnt_fUSD/contracts/contracts/interfaces/aave/v3/IAaveV3Pool.sol	5bcb2f65655f2cfcae8816ff85d665d2
./Frgmnt_fUSD/contracts/contracts/interfaces/aave/v4/IGiverPositionManager.sol	86ff51b1afca85807c44fdadce477e79
./Frgmnt_fUSD/contracts/contracts/interfaces/aave/v4/IHubBase.sol	0c92726b32fed03a195addfb450b51d7
./Frgmnt_fUSD/contracts/contracts/interfaces/aave/v4/ISpoke.sol	477c43ce1fa7f24d7477d9d5a105b2bf
./Frgmnt_fUSD/contracts/contracts/interfaces/aave/v4/ITakerPositionManager.sol	30a304a882f0a1a4c5106faecf3fd08c

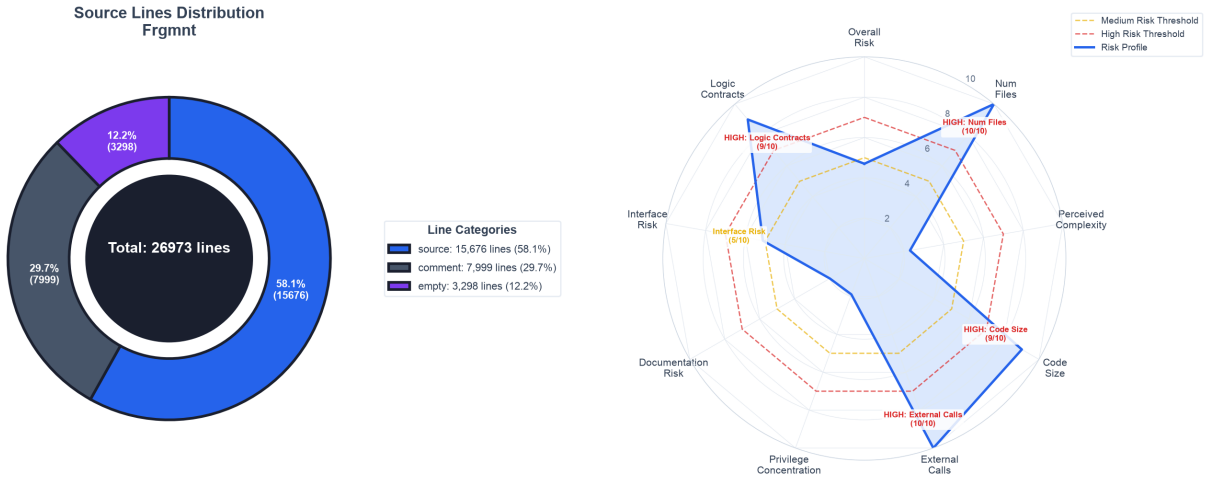
File	Fingerprint (MD5)
./Frgmnt_fUSD/contracts/contracts/interfaces/aave/v4/ITokenizationSpoke.sol	11c9f922db0a3ee2b1bbe8730c597f4c
./Frgmnt_fUSD/contracts/contracts/interfaces/compound/ICompoundV3Comet.sol	108b93ab36d02f65e58367f72fdf24ec
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IAaveLendingPoolAssetGuard.sol	d02a42090c8c5682e59bd83853e73bcb
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IAddAssetCheckGuard.sol	978951d85b09737e934c664677eac114
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IAssetGuard.sol	54623c0c4476cd2a1314e5883747f53a
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IComplexAssetGuard.sol	4d054fa4385473c581020e2d59a4a1da
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IDeficitReportingGuard.sol	28e7a3c78966b51e5d31073aedadbdef
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IERC721VerifyingGuard.sol	09f9f360832c74362e2a88830743158a
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IGuard.sol	1a3ac576a5227d0162b29d4cb973cf9a
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IHasGuardInfo.sol	703bd31a9971f8bd2a2fe9dc10dd0953
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IIncompleteValuationGuard.sol	5577ef3f0ae4c779aaf4ed2c01e0c9c0
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IMorphoBlueLendingPoolAssetGuard.sol	0fcb8dda0cdec89596b87995001bc0ce
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IMutableBalanceAssetGuard.sol	c43a66ad8399b180cfe95c7d0bc0cec6
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IPreValuedAssetGuard.sol	f5d7b884b6c4cca7317579f96ffc0eea
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/ISlippageCheckingGuard.sol	dd40a890fcd37fd1c55fcc5660d1da63
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/ISubPositionGuard.sol	9febb70deb263e55d3cbe19115ec74b9
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/ITxTrackingGuard.sol	06b6902ed47a7d1d0172b27b1646c0bb

File	Fingerprint (MD5)
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IUnwindCostAwareGuard.sol	4e0237ce0a21c4b48d2faf516e8c3786
./Frgmnt_fUSD/contracts/contracts/interfaces/guards/IWithdrawableBalanceGuard.sol	af51c793ec7fbfe4b2cab93978923924
./Frgmnt_fUSD/contracts/contracts/interfaces/uniswapV2/IUniswapV2Factory.sol	c3dbf6c224152582867c46f0f9f3ebfe
./Frgmnt_fUSD/contracts/contracts/interfaces/uniswapV2/IUniswapV2Pair.sol	1d8c09df83852e8a43df3d037545d72c
./Frgmnt_fUSD/contracts/contracts/interfaces/uniswapV2/IUniswapV2Router.sol	d7ddbc20c53dd029319ae73f313221dd
./Frgmnt_fUSD/contracts/contracts/priceAggregators/AssetHandler.sol	cec5d8db118e085dff4a1def1db61355
./Frgmnt_fUSD/contracts/contracts/priceAggregators/USDPriceAggregator.sol	ff72547fb5f9f19fb514c52232dcf584
./Frgmnt_fUSD/contracts/contracts/priceAggregators/UniV3TWAPAggregator.sol	fd4a0237b1a2ebd9ce993a724bafe387
./Frgmnt_fUSD/contracts/contracts/utils/AddressHelper.sol	228a28de1fb9b65c0ff4dc941e4adf11
./Frgmnt_fUSD/contracts/contracts/utils/CLPriceLibrary.sol	3047290cc1a3cba63b4665c83a5cf880
./Frgmnt_fUSD/contracts/contracts/utils/CallResultChecker.sol	5f66469ae206b0aee3d083d5b461ca45
./Frgmnt_fUSD/contracts/contracts/utils/DateTime.sol	d78746fa5fb399a8c1c014dfd7f0a882
./Frgmnt_fUSD/contracts/contracts/utils/DhedgeMath.sol	465faad7e624e5497016304ae1ef6bc6
./Frgmnt_fUSD/contracts/contracts/utils/FundCalculationLibrary.sol	4b94d4b3083ebdfe335e13fdf56ca0d9
./Frgmnt_fUSD/contracts/contracts/utils/MorphoChecksLib.sol	f53e76a23c3ae2dda757d6acb7d96606
./Frgmnt_fUSD/contracts/contracts/utils/MorphoCollectLib.sol	adb8301843a9864abc6112b1f924f4dc
./Frgmnt_fUSD/contracts/contracts/utils/MorphoMathLib.sol	dab8102c439ba28233afc669a19cad41
./Frgmnt_fUSD/contracts/contracts/utils/PoolLogicFlashloanAave.sol	45785cd6bdf03e7bdbc0f304940c51bb
./Frgmnt_fUSD/contracts/contracts/utils/PoolLogicFlashloanMorpho.sol	44633f3e739f6382e7e2d050711ecc84
./Frgmnt_fUSD/contracts/contracts/utils/PoolTxExecutor.sol	f55d6f98292d909f189111bd9c9488a4
./Frgmnt_fUSD/contracts/contracts/utils/PrecisionHelper.sol	85c2bf2b035ae5b4b8f684074966826a

File	Fingerprint (MD5)
./Frgmnt_fUSD/contracts/contracts/utils/ProxyAdmin.sol	618d14b77d6db0b71cc1e0bc42eab75a
./Frgmnt_fUSD/contracts/contracts/utils/SafeERC20.sol	2d0edbd739d419a7f89c6c143f683aa8
./Frgmnt_fUSD/contracts/contracts/utils/SlippageAccumulator.sol	0649e856fe790fb21e62b75f4c334c8d
./Frgmnt_fUSD/contracts/contracts/utils/SlippageAccumulatorUser.sol	34235b40ed8dbfc0fdb6a31aa0a12ea3
./Frgmnt_fUSD/contracts/contracts/utils/TxDataUtils.sol	659d83411393c22957caa46dc8dba376
./Frgmnt_fUSD/contracts/contracts/utils/UniswapV3PriceLibrary.sol	c11f0ad1f017d79d45b3c07dee71d5d6
./Frgmnt_fUSD/contracts/contracts/utils/WithdrawalPlanLib.sol	84d6b71421d11a77501b9acccb0607db
./Frgmnt_fUSD/contracts/contracts/utils/tracker/NftTrackerStorage.sol	227b86fd5d360f5927df71bb17be820c
./Frgmnt_fUSD/scripts/PoolManagerLogicRead.ts	7ed7afac329b9b6b0298cd667f42953c
./Frgmnt_fUSD/scripts/add_assets.ts	b1d704757a4d4d33ac5cf29696db429c
./Frgmnt_fUSD/scripts/add_supported_asset.ts	12688ac789336fab43681da806aaee53
./Frgmnt_fUSD/scripts/check-contract-size.ts	90d2652b0c8e7959ecec4d307bae8f6f
./Frgmnt_fUSD/scripts/check-coverage-threshold.ts	33e2a6fe560413d571f6e0ba50ce7742
./Frgmnt_fUSD/scripts/deployERC20Guard.ts	6866663dfdb580f4ee9c490f1d64f3e5
./Frgmnt_fUSD/scripts/deploy_asset_guards.ts	00e7639fc67d5ae06e87c28fd6c3834a
./Frgmnt_fUSD/scripts/deploy_contract_guard.ts	401a5872650c17dd2cbce82b0333d6a5
./Frgmnt_fUSD/scripts/deploy_core_contracts.ts	762eb62f4a0fd73a1c42914021b12206
./Frgmnt_fUSD/scripts/deploy_selective_guards.ts	d678c8adb3fdf4f8f6c74cf64d2ef2f6
./Frgmnt_fUSD/scripts/deploy_usd_aggregator.ts	db927cf2731463129c8e1db55d7a2d64
./Frgmnt_fUSD/scripts/pending_reward.ts	9a39fbbaff294921e702de9cf9fafbe1
./Frgmnt_fUSD/scripts/remediate_NftTrackerStorage.ts	aee9d4631b7acbf71e1dcf96e752d8f7
./Frgmnt_fUSD/scripts/remediate_SlippageAccumulator.ts	8bf22c8d497fb9f466b172c0b567b6e4
./Frgmnt_fUSD/scripts/setTrading.ts	37263f7211446fd0823ff24fc30cde8e
./Frgmnt_fUSD/scripts/set_Asset_Guard.ts	5c99a8989a20c5e3e570c91e41994b32
./Frgmnt_fUSD/scripts/set_Contract_Guard.ts	afbbfe8408a8f056bfc5e71b002a73c6
./Frgmnt_fUSD/scripts/setup_Token_Logic.ts	ae17af2a82e3b9fd3f8a17fefce10986

File	Fingerprint (MD5)
./Frgmnt_fUSD/scripts/transferRoles_Governance.ts	7871d293d8b7abe983535d2b6b32ef10
./Frgmnt_fUSD/scripts/transferRoles_TokenLogic.ts	4aec09a3582836a7d0195769f739ccb8
./Frgmnt_fUSD/scripts/transferRoles_poolLogic.ts	fe722265d6eb62f85a64440c14fabb79
./Frgmnt_fUSD/scripts/transferRoles_poolManagerLogic.ts	3f93f679dcbd3cd5c48c87c44b55e40e
./Frgmnt_fUSD/scripts/upgrade_attested_withdrawal.ts	806f984cf27a0eb484b41ae97f0d00e5
./Frgmnt_fUSD/scripts/upgrade_core_contracts.ts	1aa43e7dd128871c2933e2d561474b3a
./Frgmnt_fUSD/scripts/utils/ownership.ts	c9e74da956699d26de9e5a0156a4d2f0
./Frgmnt_fUSD/scripts/utils/selectiveGuards.ts	e2f5c29445e3ef21063963bd2f8af5a3
./Frgmnt_fUSD/scripts/utils/upgradePreflight.ts	4b53f31bd44709b32361b1e81e8e6fc5
./Frgmnt_fUSD/scripts/utils/validateEurUsdFeed.ts	3fb6ced5c07bd5fd441b5522dce6d9c4
./Frgmnt_fUSD/scripts/utils/withdrawalPlanBuilder.ts	6d3c2f0952a76071cff98e64e317a1c3
./Frgmnt_fUSD/services/attester/src/chain.ts	30d6bf448279d7c841f12bbd701341b0
./Frgmnt_fUSD/services/attester/src/composer.ts	902c956893b6f396283947b3ac10216d
./Frgmnt_fUSD/services/attester/src/index.ts	209b6d94348fd2528207b42725f39ff3
./Frgmnt_fUSD/services/attester/src/server.ts	c3629ffa19afcf3a05b642e2a5fe0b37
./Frgmnt_fUSD/services/attester/src/service.ts	bba1bbd893ebea1044bc6b556445c69d
./Frgmnt_fUSD/services/attester/src/signer.ts	4862536320d3ebbd4574e650aa60455d
./Frgmnt_fUSD/services/attester/src/types.ts	d037b46e1cd0cbf98a6e6a82df7e65d8

5.2 Source Lines & Risk



5.3 Capabilities

Module	Functions	Public	Restricted	Key Roles
AaveLendingPoolAssetGuard	21	14	7	owner
AaveLendingPoolGuardV3	2	2	0	-
AaveV3LendingPoolSelectiveAssetGuard	2	2	0	-
AaveV4SpokeAssetGuard	14	14	0	-
AaveV4SpokeContractGuard	1	1	0	-
AaveV4SpokeManager	6	5	1	owner
AaveV4SpokeSelectiveAssetGuard	2	2	0	-
AaveV4TokenizationAssetGuard	13	13	0	-
AaveV4TokenizationContractGuard	1	1	0	-
AaveV4TokenizationManager	6	5	1	owner
AssetHandler	9	2	7	owner
CallResultChecker	1	1	0	-
ClosedAssetGuard	4	4	0	-
ERC20Guard	6	6	0	-

Module	Functions	Public	Restricted	Key Roles
FrgmntUserActions	8	8	0	-
FundCalculationLibrary	13	13	0	-
Governance	2	0	2	owner
IAaveLendingPoolAssetGuard	2	2	0	-
IAaveProtocolDataProvider	2	2	0	-
IAaveV3Pool	20	20	0	-
IAaveV4SpokeManager	8	8	0	-
IAaveV4TokenizationManager	8	8	0	-
IAddAssetCheckGuard	2	2	0	-
IAggregatorV3Interface	2	2	0	-
IAggregatorV3InterfaceWithOwner	4	4	0	-
IAssetGuard	5	5	0	-
IAssetHandler	6	6	0	-
IComplexAssetGuard	1	1	0	-
ICompoundV3Comet	3	3	0	-
IDeficitReportingGuard	2	2	0	-
IERC20	6	6	0	-
IERC20Extended	10	10	0	-
IERC721Enumerable	1	1	0	-
IERC721VerifyingGuard	1	1	0	-
IFlashLoanReceiver	1	1	0	-
IGiverPositionManager	1	1	0	-
IGovernance	2	2	0	-
IGuard	1	1	0	-
IHasAssetInfo	4	4	0	-
IHasDaoInfo	2	2	0	-
IHasFeeInfo	4	4	0	-

Module	Functions	Public	Restricted	Key Roles
IHasGuardInfo	4	4	0	-
IHasOwnable	1	1	0	-
IHasPausable	3	3	0	-
IHasSupportedAsset	2	2	0	-
IHubBase	1	1	0	-
IIncompleteValuationGuard	2	2	0	-
IManaged	3	3	0	-
IMorphoBlueLendingPoolAssetGuard	2	2	0	-
IMorphoBlueManager	8	8	0	-
IMorphoFlashLoanCallback	1	1	0	-
IMorphoVaultV2	5	5	0	-
IMorphoVaultV2Manager	9	9	0	-
IMutableBalanceAssetGuard	2	2	0	-
IPoolFactory	9	9	0	-
IPoolLogic	18	18	0	-
IPoolManagerLogic	21	21	0	-
IPreValuedAssetGuard	2	2	0	-
ISlippageCheckingGuard	1	1	0	-
ISpoke	3	3	0	-
ISubPositionGuard	2	2	0	-
ITakerPositionManager	2	2	0	-
ITokenizationSpoke	2	2	0	-
ITxTrackingGuard	2	2	0	-
IUniswapV2Factory	1	1	0	-
IUniswapV2Pair	7	7	0	-
IUniswapV2Router	8	8	0	-
IUnwindCostAwareGuard	2	2	0	-

Module	Functions	Public	Restricted	Key Roles
IV3SwapRouter	4	4	0	-
IWETH	5	5	0	-
IWithdrawableBalanceGuard	2	2	0	-
IWithdrawalEscrow	3	3	0	-
Managed	8	8	0	-
MerklRewardClaimGuard	2	2	0	-
MorphoBlueContractGuard	2	2	0	-
MorphoBlueLendingPoolAssetGuard	19	14	5	owner
MorphoBlueLendingPoolSelectiveAssetGuard	2	2	0	-
MorphoBlueManager	6	5	1	owner
MorphoCollectLib	3	3	0	-
MorphoVaultV2AssetGuard	13	13	0	-
MorphoVaultV2ContractGuard	1	1	0	-
MorphoVaultV2Manager	7	6	1	owner
NftTrackerStorage	13	9	4	owner
PoolLogic	44	41	3	owner
PoolManagerLogic	44	44	0	-
PoolTxExecutor	2	2	0	-
SlippageAccumulator	6	4	2	owner
SlippageAccumulatorUser	1	1	0	-
TokenLogic	20	7	13	DEFAULT_ADMIN, EMERGENCY
TxDataUtils	10	10	0	-
USDPriceAggregator	2	2	0	-
UniV3TWAPAggregator	2	2	0	-
UniswapV3AssetGuard	14	9	5	admin
UniswapV3NonfungiblePositionGuard	3	3	0	-

Module	Functions	Public	Restricted	Key Roles
UniswapV3RouterGuard	1	1	0	-
UniswapV3SelectiveAssetGuard	2	2	0	-
WithdrawalEscrow	2	2	0	-
WithdrawalPlanLib	8	8	0	-

5.4 Dependencies / External Imports

Import Path	Version
@morpho-org/morpho-blue/src/interfaces/IMorpho.sol	1.0.0
@morpho-org/morpho-blue/src/libraries/MarketParamsLib.sol	1.0.0
@morpho-org/morpho-blue/src/libraries/SharesMathLib.sol	1.0.0
@morpho-org/morpho-blue/src/libraries/periphery/MorphoBalancesLib.sol	1.0.0
@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol	5.4.0
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	5.4.0
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	5.4.0
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	5.4.0
@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol	5.4.0
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol	5.4.0
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20PermitUpgradeable.sol	5.4.0
@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol	5.4.0
@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol	5.4.0
@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol	5.4.0
@openzeppelin/contracts/access/Ownable.sol	5.4.0
@openzeppelin/contracts/governance/TimelockController.sol	5.4.0
@openzeppelin/contracts/interfaces/IERC1271.sol	5.4.0
@openzeppelin/contracts/interfaces/IERC4626.sol	5.4.0
@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol	5.4.0
@openzeppelin/contracts/proxy/transparent/ProxyAdmin.sol	5.4.0

Import Path	Version
@openzeppelin/contracts/proxy/transparent/TransparentUpgradeableProxy.sol	5.4.0
@openzeppelin/contracts/token/ERC20/ERC20.sol	5.4.0
@openzeppelin/contracts/token/ERC20/IERC20.sol	5.4.0
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	5.4.0
@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol	5.4.0
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	5.4.0
@openzeppelin/contracts/token/ERC721/ERC721.sol	5.4.0
@openzeppelin/contracts/token/ERC721/IERC721.sol	5.4.0
@openzeppelin/contracts/utils/Address.sol	5.4.0
@openzeppelin/contracts/utils/ReentrancyGuard.sol	5.4.0
@openzeppelin/contracts/utils/cryptography/ECDSA.sol	5.4.0
@openzeppelin/contracts/utils/math/Math.sol	5.4.0
@openzeppelin/contracts/utils/math/SafeCast.sol	5.4.0
@uniswap/v3-core/contracts/interfaces/IUniswapV3Factory.sol	0.8
@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol	0.8
@uniswap/v3-core/contracts/interfaces/callback/IUniswapV3SwapCallback.sol	0.8
@uniswap/v3-core/contracts/libraries/FullMath.sol	0.8
@uniswap/v3-core/contracts/libraries/TickMath.sol	0.8
@uniswap/v3-periphery/contracts/interfaces/IMulticall.sol	0.8
@uniswap/v3-periphery/contracts/interfaces/INonfungiblePositionManager.sol	0.8
@uniswap/v3-periphery/contracts/libraries/LiquidityAmounts.sol	0.8
@uniswap/v3-periphery/contracts/libraries/OracleLibrary.sol	0.8
@uniswap/v3-periphery/contracts/libraries/Path.sol	0.8
@uniswap/v3-periphery/contracts/libraries/PositionValue.sol	0.8

5.5 Source Units in Scope

File	Instructions	Functions	Lines	nSLOC	Comments
PoolLogic.sol	34	65	1738	884	619
Governance.sol	2	4	87	31	39
Timelock.sol	0	0	28	10	16
Managed.sol	8	13	228	106	82
FrgmntUserActions.sol	4	9	165	119	15
PoolManagerLogic.sol	37	54	958	636	195
TokenLogic.sol	9	27	856	408	343
WithdrawalEscrow.sol	1	2	80	39	32
SlippageAccumulatorUser.sol	0	2	115	42	59
CallResultChecker.sol	0	1	32	22	4
AddressHelper.sol	0	2	47	28	16
ProxyAdmin.sol	0	0	4	2	1
DateTime.sol	0	6	62	47	6
TxDataUtils.sol	7	12	178	113	34
FundCalculationLibrary.sol	0	20	825	383	383
PoolLogicFlashloanAave.sol	0	3	86	50	18
WithdrawalPlanLib.sol	2	19	1192	721	376
SafeERC20.sol	0	6	71	46	16
SlippageAccumulator.sol	3	7	242	112	91
MorphoCollectLib.sol	0	10	378	269	49
PrecisionHelper.sol	0	1	10	7	1
PoolTxExecutor.sol	0	6	228	123	65
MorphoChecksLib.sol	0	2	62	42	11
MorphoMathLib.sol	0	6	111	63	32
CLPriceLibrary.sol	0	3	100	58	29

File	Instructions	Functions	Lines	nSLOC	Comments
DhedgeMath.sol	0	1	58	46	11
PoolLogicFlashloanMorpho.sol	0	3	23	9	10
UniswapV3PriceLibrary.sol	0	6	143	83	48
UniV3TWAPAggregator.sol	1	2	189	105	54
USDPriceAggregator.sol	1	2	38	21	14
AssetHandler.sol	8	11	257	137	80
IERC721Enumerable.sol	1	1	18	5	11
IERC20.sol	5	6	64	15	41
IHasAssetInfo.sol	4	4	13	7	2
IERC20Extended.sol	10	10	33	15	6
IAaveV4SpokeManager.sol	4	8	69	27	32
IMorphoFlashLoanCallback.sol	1	1	13	4	8
IPoolLogic.sol	17	18	186	76	84
IWithdrawalEscrow.sol	2	3	22	10	9
IAggregatorV3InterfaceWithOwner.sol	4	4	14	8	1
IHasPausable.sol	3	3	11	6	2
IHasOwnable.sol	1	1	7	4	2
IMorphoVaultV2.sol	4	5	53	14	33
IPoolManagerLogic.sol	21	21	66	24	21
IWETH.sol	5	5	15	8	2
IHasGuardInfo.sol	2	2	11	5	4
IV3SwapRouter.sol	0	4	78	47	21
IMorphoVaultV2Manager.sol	8	9	67	16	40
IMorphoBlueManager.sol	8	8	59	14	34
IPoolFactory.sol	9	9	23	12	2
IGovernance.sol	2	2	9	5	2
ITransactionTypes.sol	0	0	52	44	7

File	Instructions	Functions	Lines	nSLOC	Comments
IManaged.sol	3	3	11	6	2
IHasDaoInfo.sol	2	2	9	5	2
IHasFeeInfo.sol	4	4	14	7	3
IAggregatorV3Interface.sol	1	2	17	14	1
IAssetHandler.sol	6	6	25	16	2
IAaveV4TokenizationManager.sol	8	8	57	13	34
IHasSupportedAsset.sol	2	2	14	9	2
NftTrackerStorage.sol	6	15	301	121	148
MorphoVaultV2ContractGuard.sol	0	10	426	221	160
MorphoBlueManager.sol	6	7	232	89	106
MorphoBlueContractGuard.sol	0	9	478	358	52
AaveLendingPoolGuardV3.sol	0	10	577	301	200
MorphoVaultV2Manager.sol	6	8	210	82	95
AaveV4SpokeContractGuard.sol	0	7	305	161	110
MerklRewardClaimGuard.sol	0	3	118	51	43
AaveV4TokenizationManager.sol	6	7	172	68	74
AaveV4TokenizationContractGuard.sol	0	7	262	159	68
AaveV4SpokeManager.sol	3	7	196	83	83
ERC20Guard.sol	2	6	213	101	80
UniswapV3AssetGuard.sol	10	28	838	573	165
AaveV4TokenizationAssetGuard.sol	7	17	518	246	219
AaveLendingPoolAssetGuard.sol	14	54	1586	1061	304
AaveV3LendingPoolSelectiveAssetGuard.sol	1	5	170	120	35
MorphoBlueLendingPoolAssetGuard.sol	13	37	1195	805	242
ClosedAssetGuard.sol	2	4	62	26	31
UniswapV3SelectiveAssetGuard.sol	1	4	124	86	28
AaveV4SpokeAssetGuard.sol	7	26	848	421	351

File	Instructions	Functions	Lines	nSLOC	Comments
MorphoVaultV2AssetGuard.sol	7	16	503	218	234
MorphoBlueLendingPoolSelectiveAssetGuard.sol	1	7	196	133	43
AaveV4SpokeSelectiveAssetGuard.sol	1	3	93	66	17
UniswapV3RouterGuard.sol	0	2	239	162	45
UniswapV3NonfungiblePositionGuard.sol	1	8	595	409	118
IUniswapV2Router.sol	2	8	69	46	15
IUniswapV2Factory.sol	1	1	17	4	12
IUniswapV2Pair.sol	6	7	39	13	19
IFlashLoanReceiver.sol	0	1	18	10	7
IAaveProtocolDataProvider.sol	0	2	49	30	17
ICompoundV3Comet.sol	3	3	33	6	24
IERC721VerifyingGuard.sol	0	1	12	9	1
IPreValuedAssetGuard.sol	2	2	31	5	23
ISubPositionGuard.sol	1	2	31	12	16
ITxTrackingGuard.sol	2	2	11	6	1
IAaveLendingPoolAssetGuard.sol	1	2	17	12	1
IMorphoBlueLendingPoolAssetGuard.sol	1	2	44	12	28
ISlippageCheckingGuard.sol	1	1	14	4	9
IGuard.sol	0	1	11	8	1
IHasGuardInfo.sol	2	2	11	5	4
IComplexAssetGuard.sol	0	1	30	17	10
IAddAssetCheckGuard.sol	1	2	13	9	1
IUnwindCostAwareGuard.sol	2	2	28	5	20
IAssetGuard.sol	3	5	29	21	1
IDeficitReportingGuard.sol	2	2	29	5	21
IIncompleteValuationGuard.sol	2	2	23	5	15
IMutableBalanceAssetGuard.sol	2	2	9	5	1

File	Instructions	Functions	Lines	nSLOC	Comments
IWithdrawableBalanceGuard.sol	2	2	26	5	18
ITokenizationSpoke.sol	2	2	19	5	12
ITakerPositionManager.sol	0	2	39	15	22
ISpoke.sol	2	3	46	10	33
IGiverPositionManager.sol	0	1	29	9	19
IHubBase.sol	1	1	15	4	10
IAaveV3Pool.sol	8	20	199	87	91
DataTypes.sol	0	0	85	26	56
chain.ts	0	3	213	186	12
composer.ts	0	4	170	141	20
index.ts	0	2	64	51	9
server.ts	0	7	123	97	14
service.ts	0	0	156	128	8
signer.ts	0	1	55	44	4
types.ts	0	0	145	101	35
PoolManagerLogicRead.ts	0	1	75	51	5
add_assets.ts	0	1	81	62	6
add_supported_asset.ts	0	1	64	44	4
check-contract-size.ts	0	3	144	109	14
check-coverage-threshold.ts	0	1	83	56	16
deployERC20Guard.ts	0	1	36	19	9
deploy_asset_guards.ts	0	5	139	83	21
deploy_contract_guard.ts	0	7	238	134	56
deploy_core_contracts.ts	0	7	552	335	148
deploy_selective_guards.ts	0	4	244	192	42
deploy_usd_aggregator.ts	0	5	54	35	6
pending_reward.ts	0	1	46	30	3

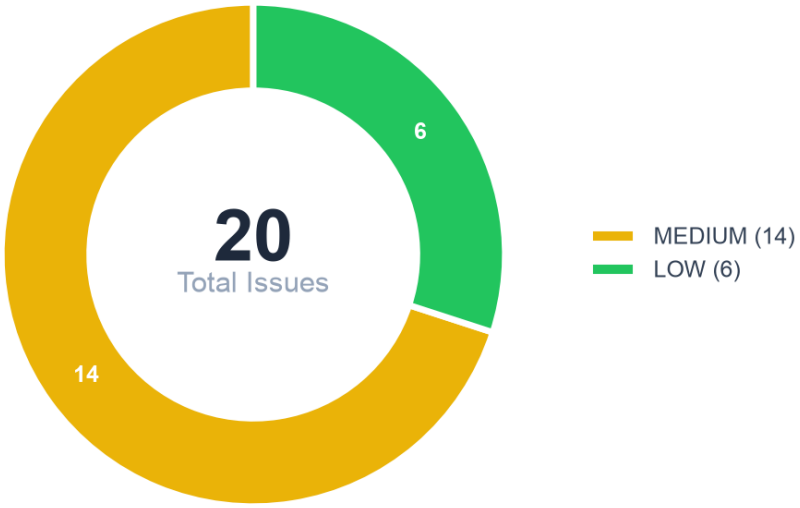
File	Instructions	Functions	Lines	nSLOC	Comments
remediate_NftTrackerStorage.ts	0	2	148	89	44
remediate_SlippageAccumulator.ts	0	2	164	108	41
setTrading.ts	0	1	42	26	5
set_Asset_Guard.ts	0	1	61	32	18
set_Contract_Guard.ts	0	1	81	52	18
setup_Token_Logic.ts	0	1	73	26	33
transferRoles_Governance.ts	0	1	290	180	89
transferRoles_TokenLogic.ts	0	2	94	56	14
transferRoles_poolLogic.ts	0	1	25	13	3
transferRoles_poolManagerLogic.ts	0	1	55	30	11
upgrade_attested_withdrawal.ts	0	2	433	251	157
upgrade_core_contracts.ts	0	4	677	380	263
ownership.ts	0	2	51	33	15
selectiveGuards.ts	0	13	493	406	50
upgradePreflight.ts	0	5	85	52	26
validateEurUsdFeed.ts	0	3	86	70	0
withdrawalPlanBuilder.ts	0	12	182	123	47
compare-with-baseline.js	0	10	147	122	15
Total	**397**	**965**	**26973**	**15676**	**7999**

6. Scope of Work

This audit covers the Frgmnt fUSD protocol on Base, including the token and vault, the integration guards, and the supporting deployment and off-chain components.

6.1 Findings Overview

Findings are classified by severity: Critical, High, Medium, Low, and Informational.



No	Title	Severity	CVSS	Status
6.2.1	Fresh-deployment script leaves the deployer as owner of all three Transparent proxy ProxyAdmins	MEDIUM	6.5	FIXED
6.2.2	execTransaction missing nonReentrant: cross-function reentrancy via callback-exempt manager	MEDIUM	5.5	FIXED
6.2.3	executeOperation missing nonReentrant: partial NAV reentrancy during Aave flashloan unwind	MEDIUM	5.3	FIXED
6.2.4	onMorphoFlashLoan missing nonReentrant: partial NAV reentrancy during Morpho flashloan unwind	MEDIUM	5.9	FIXED
6.2.5			4.3	

No	Title	Severity	CVSS	Status
	Fee unit-of-account mismatch in FundCalculationLibrary fee summation	MEDIUM		FIXED
6.2.6	Hub-wide liquidity overstatement in AaveV4SpokeAssetGuard: per-Spoke cross-pool capping absent	MEDIUM	5.3	ACKNOWLEDGED
6.2.7	withdrawalSlippageBps dual-use: spot-TWAP deviation gate and LP output slippage share one threshold	MEDIUM	5.4	FIXED
6.2.8	UniswapV3SelectiveAssetGuard inherits withdrawalSlippageBps dual-use	MEDIUM	4.3	FIXED
6.2.9	AaveLendingPoolAssetGuard: unconfigured collateral swap route causes permanent pool-wide DoS	MEDIUM	4.9	FIXED
6.2.10	UniswapV3RouterGuard does not validate amountOutMinimum on manager swaps	MEDIUM	4.4	FIXED
6.2.11	Contracts that the documents call protocol-owned are owned by the pool manager Safe, so the manager can move the pool's USDC out through a Morpho market it controls	MEDIUM	4.9	FIXED
6.2.12	The Uniswap router guard validates only the first and last token of a multi-hop path, and PoolLogic.execTransaction is unlocked, so a manager or trader can mint unbacked fees and rewards	MEDIUM	6.5	FIXED
6.2.13	The manager can act as any user on deposit, stake, unstake and immediate withdrawal by listing a contract as a callback sender	MEDIUM	5.5	FIXED
6.2.14	Liquidity provision through the Uniswap guard has no loss accounting beyond a price band of about 1 percent, so a trader can sandwich the pool's own mints repeatedly	MEDIUM	4.3	FIXED
6.2.15	Reference attester service lets the requesting user choose which assets a plan draws from	LOW	3.1	FIXED
6.2.16	A token-only plan in a pool with a debt-bearing position leaves accountedAssets below the gross NAV, so the next accrual mints a fee and rewards from the exiter's haircut	LOW	3.7	FIXED

No	Title	Severity	CVSS	Status
6.2.17	A dust queued request lets any fUSD holder block plan draws of a deposit asset in queue mode	LOW	3.1	FIXED
6.2.18	The attested-only upgrade path leaves validated protections inactive and cannot be completed by the core upgrade script	LOW	2.2	FIXED
6.2.19	A floored NAV hides a direct-leg over-draw when the deficit comes from another guard	LOW	3.1	FIXED
6.2.20	The emergency stop is not independent of the manager, or arrives after a rotation matures, in configurations the repository's own scripts produce	LOW	3.3	FIXED

6.2 Manual and Automated Vulnerability Test

6.2.1 Fresh-deployment script leaves the deployer as owner of all three Transparent proxy ProxyAdmins

MEDIUM CVSS 6.5 FIXED

DESCRIPTION

PoolLogic, PoolManagerLogic and AssetHandler are Transparent proxies, each with its own ProxyAdmin. The owner of a ProxyAdmin can replace that proxy's implementation with arbitrary code, and for PoolLogic that is unconditional control of every asset the pool holds. The client's documentation states this authority is held by GOVERNANCE_SAFE from deployment (docs/security.md, Upgrade Security) and that the deployer retains no privileged role in production (docs/deployments.md).

deploy_core_contracts.ts hands over every other role correctly. The three upgrades.deployProxy calls however pass no initialOwner in their options, and in @openzeppelin/hardhat-upgrades 3.9.1 the owner of a newly created ProxyAdmin then defaults to the deploying account. The script never transfers those ProxyAdmin contracts afterwards, and its summary prints the admin addresses but not their owners, so the gap is invisible in the script output. Whenever the deploying account differs from GOVERNANCE_SAFE, the deployer owns all three ProxyAdmins after a fresh deployment while PoolLogic.owner() correctly reports GOVERNANCE_SAFE.

The GOVERNANCE_SAFE constant is currently the old single key 0xafb9B883637f72767ADf7193Bb3B8e59C02Ea05d, which is also the recorded deployer, so nothing extra is exposed on that flow. The script's own design requires the constant to become a real Safe, and at that point one externally owned account holds every upgrade authority while the documentation says a multisig does.

Relationship to the CertiK reports: CertiK FNA-01, "Centralization Related Risks" (acknowledged), produced a detailed ownership remediation. The client's response of 7 August 2026 in that finding states that the AssetHandler ownership bug was fixed directly in deploy_core_contracts.ts and that transferRoles_Governance.ts now transfers Governance.owner, AssetHandler.owner, PoolManagerLogic.factoryOwner, PoolLogic.owner and the TokenLogic admin role to the Timelock. The three Transparent-proxy ProxyAdmin contracts created by the upgrades.deployProxy calls are absent from that inventory, so this finding is the one ownership class in the same contracts that both remediations skipped. The earlier FRG-07, "Centralized Control Of Contract Upgrade" (acknowledged), and FRG-08, "Centralization Related Risks", covered the same area without reaching the ProxyAdmins either. It is therefore not a re-report: the client's own remediation list is the evidence that these three contracts were never covered.

Impact: until the ProxyAdmin contracts are handed over, a single deployer key can take all pool assets by upgrading PoolLogic, or corrupt valuation and asset configuration by upgrading PoolManagerLogic or AssetHandler. This defeats the custody model that the FNA-01 remediation was applied to establish.

The live USD deployment is not affected. All three ProxyAdmin owners plus PoolLogic.owner, PoolManagerLogic.factoryOwner and AssetHandler.owner were read on Base on 6 October 2026 and are held by the DAO Safe 0x74aF72D91D5FB263fBa09Ed43aD1C1ea079058B3. The exposure is any fresh deployment through this script, in particular the EUR product, which this branch is meant to deploy.

Rated Medium and not High because the actor is a trusted key and the live product is unaffected. Not Low because the documented custody guarantee is silently false on every fresh deployment, the power exposed is total control of pool assets, and the earlier FNA-01 remediation covered every role except these three.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/scripts/deploy_core_contracts.ts#L330-L341

```
const poolLogic = await upgrades.deployProxy(
PoolLogic,
```

RECOMMENDATION

Pass initialOwner: GOVERNANCE_SAFE in the options of each of the three deployProxy calls, or call transferOwnership on each ProxyAdmin at the end of the script.

End the script with an assertion that reads the owner of all three ProxyAdmin contracts and fails unless it equals GOVERNANCE_SAFE.

Extend scripts/transferRoles_Governance.ts so the ProxyAdmin contracts move to the Timelock together with the other roles, and correct the Upgrade Security paragraph of docs/security.md.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/1

Commit dde19f8 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/dde19f8) passes initialOwner: GOVERNANCE_SAFE on all three deployProxy calls, asserts each ProxyAdmin owner before the deploy is treated as done, and extends transferRoles_Governance.ts so the three ProxyAdmins move to the Timelock with the other roles. docs/security.md now states that ProxyAdmin ownership is GOVERNANCE_SAFE at deploy time.

6.2.2 execTransaction missing nonReentrant: cross-function reentrancy via callback-exempt manager

MEDIUM CVSS 5.5 FIXED

DESCRIPTION

execTransaction is the sole external state-mutating function on PoolLogic that lacks nonReentrant. All 13 other external state-mutating entrypoints carry the modifier. The upstream dHEDGE V2 reference implementation includes nonReentrant on this function. The Frgmnt fork removed it during refactoring.

PoolTxExecutor.exec issues to.call(data) at line 57 and runs _checkReservedBalancesIntact() at line 62, after the external call. A callback sender registered by the manager can call execTransaction re-entrantly during this window. The PoC demonstrates a successful re-entry that transfers 50,000 USDC to an attacker-controlled address within a single outer execTransaction call.

The exploit requires the manager to deploy and register a ReentrantCallbackSender. Under the client trust model, the manager is a trusted role. However, the guard system is specifically designed to constrain what the manager can do, and the absence of nonReentrant on this function is a regression from the upstream baseline that creates a guard gap the client's scope explicitly keeps in scope.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/PoolLogic.sol

```
function execTransaction(address to, bytes calldata data) external returns (bool)
{
return _execTx(to, data);
}
```

RECOMMENDATION

Add nonReentrant to execTransaction:

function execTransaction(address to, bytes calldata data) external nonReentrant returns (bool) {

This is consistent with all 13 other external state-mutating functions and restores parity with the upstream dHEDGE V2 implementation.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit 442c8b2 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/442c8b2) adds nonReentrant to execTransaction, matching the other state-changing entrypoints. A nested execTransaction from a callback now reverts. Commit 33f1701 moved the Morpho flashloan body into PoolTxExecutor first so the modifier fit under the EIP-170 size limit.

6.2.3 executeOperation missing nonReentrant: partial NAV reentrancy during Aave flashloan unwind

MEDIUM CVSS 5.3 FIXED

DESCRIPTION

PoolLogic.executeOperation carries no nonReentrant modifier. The transaction loop inside PoolLogicFlashloanAave executes arbitrary external calls. If an ERC-777-style token is used as collateral in an Aave V3 position and the loop includes a transfer to an attacker-controlled address, the token's tokensReceived hook fires during the call. At that moment, the protocol's reentrancy slot is not locked, so harvest(), stake(), mintManagerFee(), and all other guarded entrypoints remain callable.

Re-entering harvest() during a partial debt repayment window causes _accrueYield() to run when NAV is inflated: debt is partially repaid while collateral is intact, so getBalance() reports an overstated equity figure. The compoundedRewardIndex advances at an incorrect rate, causing sfUSD stakers to be diluted by the amount the attacker subsequently claims.

ERC-777 tokens are not banned by on-chain guards. KI-SEC-29 is an operational recommendation only. The structural PoC confirms the absence of nonReentrant and the re-entry mechanism.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/utils/PoolLogicFlashloanAave.sol

```
function executeOperation(
address[] calldata assets,
uint256[] calldata amounts,
uint256[] calldata premiums,
address initiator,
bytes calldata params
) external override returns (bool) {
return _executeAaveFlashloan(assets, amounts, premiums, initiator, params);
}
```

RECOMMENDATION

Add nonReentrant to executeOperation:

function executeOperation(...) external override nonReentrant returns (bool) {

Aave's own swap adapters implement executeOperation with nonReentrant. Alternatively, ensure PoolLogic's reentrancy lock is set before delegating into the flashloan library.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit 77d900c (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/77d900c) does not put nonReentrant directly on executeOperation: that callback is only reached nested inside an already locked outer call, and a direct modifier would revert every Aave debt unwind. The callback now reverts with FlashloanCallbackOutsideGuardedCall unless that outer lock is already held.

6.2.4 onMorphoFlashLoan missing nonReentrant: partial NAV reentrancy during Morpho flashloan unwind

MEDIUM CVSS 5.9 FIXED

DESCRIPTION

The Morpho Blue flashloan path has the same nonReentrant gap as the Aave path (F-PLFA-001). onMorphoFlashLoan is external override with no nonReentrant modifier. The transaction loop inside PoolLogicFlashloanMorpho executes arbitrary external calls under the same conditions. An ERC-777 token's tokensReceived hook can re-enter harvest() during partial debt repayment, corrupting compoundedRewardIndex against an inflated NAV snapshot.

Morpho Blue transfers assets to PoolLogic before calling the callback (Morpho.sol:429), so the re-entry window is open for the full duration of the unwind loop. The four structural tests confirm the mechanism and the distinction from KI-SEC-05 and KI-INT-IMPL-20, which cover user functions and the WithdrawalPlanLib loop respectively, not Morpho flashloan callbacks.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/utils/PoolLogicFlashloanMorpho.sol

```
function onMorphoFlashLoan(uint256 assets, bytes calldata params) external
override {
_executeMorphoFlashloan(assets, params);
}
function _getPoolManagerLogic()
internal
view
```

RECOMMENDATION

Add nonReentrant to onMorphoFlashLoan:

function onMorphoFlashLoan(uint256 assets, bytes calldata data) external override nonReentrant {

This finding should be remediated together with F-PLFA-001 (Aave path) to close the reentrancy gap in all three flashloan callback entry points.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit d7327f2 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/d7327f2) applies the same outer-lock assertion to onMorphoFlashLoan. A direct nonReentrant modifier would have reverted every Morpho debt unwind for the same reason as M-02.

6.2.5 Fee unit-of-account mismatch in FundCalculationLibrary fee summation

MEDIUM CVSS 4.3 FIXED

DESCRIPTION

calculateManagementFee at line 61 computes the fee as a fraction of the raw fUSD token supply:

managementFee = (_totalFusd * _managementFeeNumerator * dt) / _feeDenominator / 365 days

calculatePerformanceFee at line 36 returns a value denominated in USD-18 (the incremental NAV dollar value). Both outputs are summed at PoolLogic.sol:748 and passed to ITokenLogic.mintFromPool. When NAV per fUSD diverges from $1.00, which is the normal state for any pool that has earned yield, the management fee is sized against the token count instead of the USD AUM value. At NAV = $1.05, the management fee is under-minted by approximately 4.76% per cycle. At NAV = $0.95 it is over-minted by approximately 5.26%.

The error compounds across accrual cycles. For a $10M AUM pool with a 0.5% annual management fee and sustained 5% positive yield, the systematic under-payment accumulates to approximately $23K per year.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/utils/FundCalculationLibrary.sol#L36-L61andPoolLogic.sol:748

```
function calculatePerformanceFee(
uint256 _totalValue,
uint256 _accountedAssets,
uint256 _performanceFeeNumerator,
uint256 _feeDenominator
) internal pure returns (uint256 performanceFee, uint256 netYield) {
if (_totalValue <= _accountedAssets) {
return (0, 0);
```

```
}
uint256 incrementalYield = _totalValue - _accountedAssets;
performanceFee = (incrementalYield * _performanceFeeNumerator) / _feeDenominator;
netYield = incrementalYield - performanceFee;
```

RECOMMENDATION

Convert the management fee output to the same unit as the performance fee before summing. The management fee should be expressed in fUSD tokens at the current NAV-per-token rate:

managementFeeUSD = (_totalFusd * NAVperToken * _managementFeeNumerator * dt) / _feeDenominator / 365 days / 1e18;

where NAVperToken = totalFundValue() / _totalFusd. Alternatively, convert the performance fee to token units using the same rate before summing.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit 93ab052 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/93ab052) sizes the management fee against the pool's USD NAV (totalValue) instead of the raw fUSD supply, so it stays in the same unit as the performance fee.

6.2.6 Hub-wide liquidity overstatement in AaveV4SpokeAssetGuard: per-Spoke cross-pool capping absent

MEDIUM CVSS 5.3 ACKNOWLEDGED

DESCRIPTION

Re-verification note (final review, 12 October): this section consolidates the former Aave V4 Spoke TOCTOU finding (validated as F-AV4-001) with the per-call-frame ledger gap (F-005). Re-reading the guard source shows the FNA-07 liquidity ledger is applied inside withdrawProcessing and dedupes shared (hub, assetId) pairs even across Spoke instances within one call, and the share-value cap (previewRemoveByShares) is block-static, so within a single transaction the planned transfer amount always matches what the Spoke can deliver. The forced short-delivery scenario of the original PoC is therefore not reachable in same-transaction flows. What remains reachable: PoolLogic calls each asset's guard separately, so two pool assets backed by the same Hub asset plan in separate call frames, each against the full Hub liquidity. Their combined planned legs exceed what the Hub can deliver, and at execution the later leg reverts at Hub.remove's liquidity check, reverting the entire withdrawal. The impact is a liveness freeze (withdrawal DoS) for that configuration, not a silent deficit, which is why the consolidated finding is Medium rather than High.

The HubLiquidityLedger introduced to resolve FNA-07 deduplicates Hub liquidity claims only within a single call frame. Each call to getWithdrawableBalance(pool, spoke) creates a fresh HubLiquidityLedger memory ledger in stack scope. A second pool calling the same function independently reads the same IHubBase.getAssetLiquidity(assetId) value without any cross-pool coordination.

When Hub is under-liquid and two pools both independently compute that they can withdraw X from the same (hub, assetId), their combined demand exceeds the Hub's capacity. The pool whose withdrawProcessing() executes second receives a revert from ISpoke.withdraw(), blocking all withdrawals for that pool's holders until Hub liquidity recovers.

The NatSpec at lines 56-60 explicitly acknowledges the cross-guard-TYPE residual gap (different guard types drawing from the same Hub) but does not address the cross-pool gap for pools using the same guard type.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/guards/assetGuards/AaveV4SpokeAssetGuard.sol#L356-420

```
function getWithdrawableBalance(
address pool,
address spoke
) external view override returns (uint256 balanceUsd18) {
address poolManagerLogic = IPoolLogic(pool).poolManagerLogic();
uint256[] memory reserveIds = IAaveV4SpokeManager(aaveV4SpokeManager)
.getTrackedPoolReserves(pool, spoke);
HubLiquidityLedger memory ledger = _newHubLiquidityLedger(reserveIds.length);
for (uint256 i = 0; i < reserveIds.length; ++i) {
(uint256 value, ) = _reserveWithdrawableValueUsd(
pool,
spoke,
```

RECOMMENDATION

Coordinate Hub liquidity reservations across pool addresses. One approach is to maintain an on-chain mapping that records the amount already committed to pending withdrawals for each (hub, assetId) pair in the current block or block window, and cap each pool's computed withdrawable balance against the remaining uncommitted liquidity.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Accepted residual, no contract behavior change. Commits f3df464 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/f3df464) and 8e77298 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/8e77298) on pull request #5. Re-review found no safe on-chain change: the pro-rata path degrades gracefully, the queued path is backstopped by the pool's idle balance, and the attested path's

staleness window is already bounded by the attester-signed minValueOutBps. The reference attester already applies liquidityBufferBps against shared Hub liquidity; the follow-up documents that buffer and corrects an earlier inaccurate justification.

6.2.7 withdrawalSlippageBps dual-use: spot-TWAP deviation gate and LP output slippage share one threshold

MEDIUM CVSS 5.4 FIXED

DESCRIPTION

withdrawalSlippageBps (default 100 bps, max 2000 bps) is used for two independent purposes. First, it gates the TWAP oracle check: if spot deviates from the 10-minute TWAP by more than withdrawalSlippageBps, the withdrawal reverts. Second, it sets the minimum output amounts for decreaseLiquidity: the pool accepts up to withdrawalSlippageBps percent below the expected LP output.

These two uses have opposite pressure on the parameter. A tight threshold (e.g., 100 bps) provides strong oracle manipulation protection but causes frequent DoS for volatile pairs during normal market moves. A loose threshold (e.g., 500 bps) keeps withdrawals live during volatility but opens a sandwich window where MEV bots can extract up to 5% per LP withdrawal without triggering the oracle gate.

At the default of 100 bps, any spot/TWAP divergence exceeding 1%, routine for ETH/USDC or WBTC/ETH during news events, blocks all LP withdrawals for affected pools until prices reconverge.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/guards/assetGuards/UniswapV3AssetGuard.sol#L793-797

```
uint256(spotSqrtPriceX96) * 1e18,
Q192
);
uint256 twapPrice = FullMath.mulDiv(
uint256(twapSqrtPriceX96),
uint256(twapSqrtPriceX96) * 1e18,
Q192
);
uint256 diff = spotPrice > twapPrice ? spotPrice - twapPrice : twapPrice -
spotPrice;
uint256 deviationBps = FullMath.mulDiv(diff, BPS_DENOMINATOR, twapPrice);
require(
deviationBps <= withdrawalSlippageBps,
```

RECOMMENDATION

Split into two independent parameters:

uint256 public withdrawalTwapDeviationBps = 100; // oracle guard threshold

uint256 public withdrawalSlippageBps = 100; // LP decreaseLiquidity output floor

Expose separate admin setters for each. Typical production deployments use 50-200 bps for oracle deviation gates and 50-500 bps for LP output slippage depending on pool volatility profile.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit a3b0401 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/a3b0401) splits withdrawalTwapDeviationBps (oracle gate) from withdrawalSlippageBps (LP output floor), each with its own admin setter and the same default and bounds.

6.2.8 UniswapV3SelectiveAssetGuard inherits withdrawalSlippageBps dual-use

MEDIUM CVSS 4.3 FIXED

DESCRIPTION

UniswapV3SelectiveAssetGuard inherits _checkSpotPriceDeviation from UniswapV3AssetGuard without overriding it, inheriting the same dual-use parameter conflict described in F-014. Selective withdrawal paths are equally exposed. Because this is a separately deployable contract with its own guard registry entry, the fix to F-014 does not automatically remediate this contract.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/guards/assetGuards/UniswapV3SelectiveAssetGuard.sol

```
contract UniswapV3SelectiveAssetGuard is UniswapV3AssetGuard, ISubPositionGuard {
error InvalidPositionId();
error PositionsNotAscending();
error SubsetBadPortion();
error SubsetToZero();
error UnexpectedTransaction();
```

RECOMMENDATION

Apply the same parameter split as F-014 to UniswapV3SelectiveAssetGuard, either by overriding the relevant functions or by ensuring any base-contract fix propagates through inheritance correctly.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit 3366b16 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/3366b16). UniswapV3SelectiveAssetGuard inherits the M-06 split and does not reimplement the dual-use parameter. Regression tests confirm the inherited gates move independently.

6.2.9 AaveLendingPoolAssetGuard: unconfigured collateral swap route causes permanent pool-wide DoS

MEDIUM CVSS 4.9 FIXED

DESCRIPTION

Re-verification note (final review, 12 October): reclassified from High. The trigger is a trusted-role configuration gap (a manager adds an Aave collateral asset while the owner has not configured its swap route, with any debt outstanding), the impact is a liveness freeze with no value loss, and one owner call (setUniV3Fee or setUniV3PathExactIn) clears it. Under the engagement's actor-trust rubric that combination caps at Medium, matching CertiK's Medium rating for the adjacent FNA-04 valuation class.

_netRealizableBalance iterates over the collateral assets in the Aave position to compute their swap cost. For each collateral it calls _routeFeeCollateralToSettlement(underlying, settlementToken) at line 326. That function reads uniV3Fee[collateral][settlement] and checks uniV3PathExactIn[collateral][settlement]. If both are unset, it executes:

require(fee != 0, "Frgmnt: missing fee/path collateral->settle");

No try/catch exists anywhere in the call chain from getNetRealizableBalance through FundCalculationLibrary.guardNetRealizableBalance. The unconditional revert propagates to every caller: all withdrawal processing paths that invoke NAV computation with this guard, every deposit completeness check, and every fee accrual cycle.

Any pool whose operator adds a new Aave V3 collateral asset before calling setUniV3Fee enters a frozen state. There is no on-chain protection preventing deployment or asset addition without pre-configured routes. The PoC confirms: without a configured WETH-to-USDC route, getNetRealizableBalance reverts. After calling setUniV3Fee(weth, usdc, 3000) the revert is eliminated.

Impact: Pool-wide freeze on all withdrawals, NAV computation, and deposit completeness checks until the operator configures the missing route. All depositors are locked out. For a leveraged Aave V3 pool carrying millions of USD in TVL, the freeze duration is unbounded until the operator acts. The freeze is admin-fixable with a single call but there is no on-chain guarantee of timely action.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/guards/assetGuards/AaveLendingPoolAssetGuard.sol#L1290-1306

```
function _routeFeeCollateralToSettlement(
address tokenIn,
```

```
address settlementToken
) internal view returns (uint24 fee) {
bytes memory forwardPath = uniV3PathExactIn[tokenIn][settlementToken];
if (forwardPath.length > 0) {
(address pathIn, address pathOut, uint24 routeFee) =
_decodeUniV3Path(forwardPath);
require(
pathIn == tokenIn && pathOut == settlementToken,
"Frgmnt: path endpoints mismatch"
);
return routeFee;
```

RECOMMENDATION

Two mitigations are complementary. First, wrap the _routeFeeCollateralToSettlement call in a try/catch inside _netRealizableBalance and assign a worst-case swap cost (or zero) when the route is missing, rather than reverting. Second, add a check in addAssetCheck or the asset guard initializer to require that all configured collateral tokens have a registered swap route before accepting the guard registration.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit eeb0b1d (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/eeb0b1d) degrades NAV to a conservative value when an Aave collateral swap route is unconfigured, instead of reverting every valuation. This guard is not a proxy. The fix takes effect on the live pool only after the guard is redeployed and re-registered with setAssetGuard. Pull request #5 does not send that transaction.

6.2.10 UniswapV3RouterGuard does not validate amountOutMinimum on manager swaps

MEDIUM CVSS 4.4 FIXED

DESCRIPTION

All four branches of txGuard (exactInput, exactInputSingle, exactOutput, exactOutputSingle) validate that the destination asset is supported and the recipient is the pool, but impose no check on params.amountOutMinimum (or params.amountInMaximum for exact-output variants). A manager can submit swaps with amountOutMinimum = 0, which Uniswap V3 accepts, allowing the swap to execute at any price.

The SlippageAccumulator records value loss post-execution and can revert the transaction if cumulative slippage exceeds maxCumulativeSlippage, but this is a reactive circuit breaker, not a preventive control. A sufficiently sized single swap may not trip the cumulative cap if prior slippage has decayed. The guard's purpose is specifically to constrain swap parameters before execution.

While the manager must initiate the swap, the guard is designed to constrain even the manager. External MEV bots extract value from pool users through any unconstrained swap the manager submits during routine operations.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/guards/contractGuards/uniswapV3/UniswapV3RouterGuard.sol#L67-166

```
} else if (method == IV3SwapRouter.exactInputSingle.selector) {
IV3SwapRouter.ExactInputSingleParams memory params = abi.decode(
getParams(data),
(IV3SwapRouter.ExactInputSingleParams)
);
require(
poolManagerLogicAssets.isSupportedAsset(params.tokenOut),
"Frgmnt: unsupported destination asset"
);
require(poolLogic == params.recipient, "Frgmnt: recipient is not pool");
intermediateSwapData[msg.sender] = SlippageAccumulator.SwapData({
srcAsset: params.tokenIn,
```

RECOMMENDATION

In each branch of txGuard, require a non-trivial minimum output:

require(params.amountOutMinimum > 0, "UniswapV3RouterGuard: amountOutMinimum must be > 0");

For exactOutput variants, require params.amountInMaximum < type(uint256).max. Optionally, derive a TWAP-based floor consistent with the existing pattern in the Aave/Morpho unwind guards.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit a9b860c (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/a9b860c) requires a non-zero, bounded per-swap output floor in UniswapV3RouterGuard. The live guard must be redeployed and re-registered with setContractGuard. Pull request #5 does not send that transaction.

6.2.11 Contracts that the documents call protocol-owned are owned by the pool manager Safe, so the manager can move the pool's USDC out through a Morpho market it controls

MEDIUM CVSS 4.9 FIXED

DESCRIPTION

docs/contracts/MorphoBlueManager.md states that the owner is the protocol owner (intended to be the Timelock) and describes the market allowlist as a second, independent, protocol-owner-only approval step that stops a manager from registering exposure to an unvetted market. The guard relies on it. MorphoBlueContractGuard checks only that the market id is on the allowlist, that the loan token is a supported asset and that onBehalf and the receiver are the pool. It never reads the market's oracle, IRM or LLTV, and the document says so (the market parameters "are not re-validated here").

On the live deployment the allowlist is not protocol-owned. MorphoBlueManager.owner(), MorphoBlueLendingPoolAssetGuard.owner() and PoolManagerLogic.manager() all return the same address, the manager Safe 0x18f84Ea9D3351081780d817e5edd554112CF81Ab. The same Safe owns the SlippageAccumulator and an NftTrackerStorage, and the retired deployer key 0xafb9B883637f72767ADf7193Bb3B8e59C02Ea05d (no code) owns the Aave V3 asset guard and is the admin() of the Uniswap V3 asset guard. The DAO Safe owns everything else (PoolLogic, PoolManagerLogic, Governance, AssetHandler and the ProxyAdmin contracts). With one Safe holding both keys, the second key does not exist.

Evidence. Ownership was read with owner() on Base for every contract in docs/deployments.md (table in section 8). The theft path was run on a fork of live state at block 52,168,988 with the unmodified live contracts and the real Morpho Blue and USDC. A negative control first shows the live guard rejecting an unlisted market. The owner then lists a market created with an attacker-controlled oracle (createMarket is permissionless, and the AdaptiveCurve IRM and 86 percent LLTV are already enabled on Morpho), the pool withdraws about 40.6k USDC from a legitimate market and supplies it to the new one, and a second address borrows all of it. 40,578.89 USDC reached the attacker wallet. The pool NAV was 162,295.40 USD before and 162,295.40 USD after, because the pool's supply shares stay valued at par until bad debt is realized, so deposits, fee accrual and redemptions continue at full value while the funds are gone. The test impersonates the Safe, a real run needs two of its three signatures. No Safe guard or module restricts the calls.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/guards/contractGuards/MorphoBlueManager.sol#L100-L114

```
function setPoolMarkets(address pool, Id[] calldata markets) external onlyOwner {
require(pool != address(0), "Invalid pool address");
Id[] memory oldMarkets = poolMarkets[pool];
```

RECOMMENDATION

Have the current owner call transferOwnership on MorphoBlueManager and the Morpho asset guard (and, per the ownership table, also SlippageAccumulator, NftTrackerStorage and the Aave V3 asset guard) to the DAO Safe, or to the Timelock once it owns the other roles, then repeat the owner() checks. Add every one of these contracts to scripts/transferRoles_Governance.ts, add the handover

to scripts/deploy_contract_guard.ts, and stop scripts/utils/selectiveGuards.ts from handing the new selective guard to old.owner(). A stricter on-chain option is a check in setPoolMarkets that the caller is neither the pool manager nor a trader.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit ebd47bb (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/ebd47bb) makes MorphoBlueManager.setPoolMarkets reject a pool whose manager is the contract owner. Ownership of the affected live contracts still has to be transferred off the manager Safe. Pull request #5 does not send that transaction.

6.2.12 The Uniswap router guard validates only the first and last token of a multi-hop path, and PoolLogic.execTransaction is unlocked, so a manager or trader can mint unbacked fees and rewards

MEDIUM CVSS 6.5 FIXED

DESCRIPTION

A manager or trader sends an exactOutput swap whose path runs through a token it controls (USDC, HOP, USDC). The guard checks that the destination is a supported asset and the recipient is the pool, and it never inspects the middle token. Uniswap pays the output to the pool first, then the nested hop calls the HOP token's transfer hook while the pool holds the output but has not paid the input. At that moment the pool NAV is overstated by the whole swap size. The hook calls an accrual entry point (on the branch TokenLogic.deposit, which calls the unlocked checkpointFeesForDeposit, and on the live implementation stake, unstake or harvest, which are nonReentrant but whose lock is not held during execTransaction). _accrueYield reads the overstated NAV as yield, mints a performance fee to the manager as unbacked fUSD, accrues the rest to stakers through the reward index, and ratchets accountedAssets up to the inflated NAV. The hook tops up exactly the swap shortfall, so the pool's USDC balance is the same before and after and the slippage accumulator records nothing.

Why it is latent today. The live router guard (0xcAE75F063Ef5b432F4ad3140960c888a0795d5DC) accepts a path with an unsupported middle token (a simulated txGuard call returned success, while an unsupported destination reverts). It is bound to a SlippageAccumulator whose factory is PoolLogic instead of PoolManagerLogic (the CertiK finding that scripts/remediate_SlippageAccumulator.ts remediates), so afterTxGuard reverts with CallbackSenderNotAllowed() and no router swap works through the live pool. The remediation script's second phase registers the repository UniswapV3RouterGuard, which has no per-hop check. Applying it as written opens this path.

Evidence. poc-s2-auth-multihop-reentrancy-fork.test.ts runs on a Base fork with the real Uniswap V3 factory and pools, SwapRouter02 and USDC and the real Frgmnt contracts (a mock price feed at 1 USD). A 100,000 USDC swap mints 20,000 fUSD of fee, lifts accountedAssets from 200,000 to 301,005, and an honest depositor of 200,000 can then redeem 180,991 (a loss of 19,009 that equals the manager-side gain). A 20,000,000 USDC swap leaves her 9,088.65 (a 95.5 percent loss). A trader acting alone gets the same haircut. poc-s2-auth-live-multihop-fork.test.ts applies the remediation on the fork and runs the attack from the live trader EOA against the live contracts. With a 1,000,000 USDC swap and a hook that calls harvest() with a 20,000 stake, accountedAssets went from 182,294 to 1,182,237, the manager Safe was minted 199,989 unbacked fUSD and redeemed all of the pool's USDC at par, and a real 38,700 sfUSD staker could no longer redeem (without the attack she redeems 38,724 USDC). A trader acting alone, after the 24 hour cooldown, redeemed 89,386 USDC through staking rewards. About 812,000 USDC of swap size drains the whole live pool, and a free Morpho Blue flash loan (about 233 million USDC) pays for the attacker's liquidity. The 0.13 USDC idle balance does not block it because the hook donates the shortfall. The live implementation is worse than the branch because it has no solvency haircut.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/guards/contractGuards/uniswapV3/UniswapV3RouterGuard.sol#L67-L89

```
if (method == IV3SwapRouter.exactInput.selector) {
IV3SwapRouter.ExactInputParams memory params = abi.decode(
getParams(data),
(IV3SwapRouter.ExactInputParams)
);
(address srcAsset, address dstAsset) = _decodePath(params.path);
require(
poolManagerLogicAssets.isSupportedAsset(dstAsset),
"Frgmnt: unsupported destination asset"
);
require(poolLogic == params.recipient, "Frgmnt: recipient is not pool");
intermediateSwapData[msg.sender] = SlippageAccumulator.SwapData({
```

RECOMMENDATION

Require every hop token of a multi-hop path to be a registered or supported asset, not only the path endpoints: _decodePath already walks the tokens, so each hop can be validated with the same isSupportedAsset call the destination gets. On the pool side, assert in PoolTxExecutor.exec that accountedAssets is unchanged by a guarded call, which closes every callback-class reentry across all guards at once ((see also remapped) for the execTransaction lock and M-11 for the missing per-swap floor on the same router). Give the replacement SlippageAccumulator to the DAO rather than the manager Safe.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit b518063 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/b518063) requires every intermediate hop of a multi-hop Uniswap path to be a supported asset. The headline reentrancy impact no longer reproduces because M-01 already locks execTransaction; softstack closed issue #3 (https://github.com/frgmnt-dao/Frgmnt_fUSD/issues/3) on that basis. The replacement router guard must be redeployed and re-registered. Pull request #5 does not send that transaction.

6.2.13 The manager can act as any user on deposit, stake, unstake and immediate withdrawal by listing a contract as a callback sender

MEDIUM CVSS 5.5 FIXED

DESCRIPTION

Both _actionSender helpers replace msg.sender with IUserActionSender(sender).actionUser() whenever the sender is on the pool's callback allowlist. The intended member is FrgmntUserActions, which sets actionUser to its own caller. Nothing restricts the list to that router, and setAllowedCallbackSender accepts any non-zero address from the manager. A contract the manager deploys and lists can return any address from actionUser(), and the pool and the token then run the user paths as that address. stake pulls the named user's fUSD allowance and sends the entry fee to the manager. unstake burns the user's sfUSD and needs no allowance. withdrawCashImmediate burns the user's fUSD with burnFrom and charges the exit fee to the manager. deposit pulls the user's collateral allowance and re-sets the weighted cooldown on the whole balance.

Evidence. poc-s2-tok-callback-impersonation.test.ts and poc-s2-auth-callback-sender.test.ts on the real stack. With the entry fee at its 1 percent cap, 100 forced unstake and stake cycles moved 63.4 percent of a 10,000 sfUSD staker's principal to the manager (3,660.32 left, the manager gained exactly the loss), and 60 cycles took 45 percent in the second run. A forced withdrawal charges the exit fee to the victim. Forced dust deposits reset the victim's cooldown to a full day and a second deposit just before expiry extends it, which blocks cash exits and fUSD transfers and needs no fee increase. A forced unstake needs no allowance at all. A user whose allowances were consumed by their own transaction is protected from the fee variants.

Impact. Theft of staker principal through repeated entry fees, forced exits and forced unstakes. At the audited block 158,990 sfUSD of 160,919 fUSD outstanding (98.8 percent) is staked and exposed.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/PoolManagerLogic.sol#L886-L892

```
function setAllowedCallbackSender(address caller, bool allowed) external
onlyManager {
require(caller != address(0), "caller=0");
allowedCallbackSenders[caller] = allowed;
```

```
    emit AllowedCallbackSenderSet(caller, allowed);
    }
    function setNftMembershipCollectionAddress(address addr) external onlyManager {
```

RECOMMENDATION

Restrict setAllowedCallbackSender so the manager cannot admit arbitrary senders: allow only the single FrgmntUserActions address whose value is set by the factoryOwner, or only addresses with a registered contract guard (the flash loan callbacks already require exactly this). If third-party routers must be admitted, bind user consent per action with an EIP-712 authorization or require msg.sender == user unless the sender is the factoryOwner-set router. The existing allowlist members that legitimately need the callback gate keep working under both options.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit d756cfa (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/d756cfa) limits user-action impersonation to a router set by the factoryOwner. setUserActionsRouter must be called as part of the upgrade. Pull request #5 does not send that transaction.

6.2.14 Liquidity provision through the Uniswap guard has no loss accounting beyond a price band of about 1 percent, so a trader can sandwich the pool's own mints repeatedly

MEDIUM CVSS 4.3 FIXED

DESCRIPTION

The guard lets the manager or the trader mint a position or add liquidity for the pool when the pool's spot price is inside a band around the Chainlink fair price. The band is 0.5 percent of the square root price, about 1 percent of the price, for fee tiers below 0.5 percent, and 1.5 times the fee for higher tiers (about 3 percent for the 1 percent tier). Nothing else limits what the pool pays. The attacker pushes the pool's spot price to just inside the band with its own capital, calls execTransaction on the position manager with the pool as recipient and zero minimums, and then swaps back. The reversion walks through the pool's new range, so the pool buys the appreciated token above the market price or sells it below. The difference is the pool's loss and the attacker's gain, less swap fees. A range that lies entirely on one side of the skewed price needs only one token, so no rebalancing swap is needed, and alternating an up skew with a USDC-only range and a down skew with a WETH-only range repeats the cycle. The NAV prices the new position at the fair price, so the loss shows up in the NAV at once, but no code compares the value spent with the value created. The SlippageAccumulator, which caps router swaps at 5 percent per 6 hours, is attached to the router guard only and never sees these operations.

Evidence. poc-s2-uni-sandwich-mint.test.ts runs on the real stack with the real Uniswap V3 factory, pools and position manager, real tokens and real Chainlink feeds (five cases). In the real WETH/USDC 0.05 percent pool, a position of about 1,000,000 USD minted with the spot pushed 90 ticks (0.9 percent) above fair costs the pool 3,855.83 USD (0.39 percent of the position) and nets the attacker 3,444.68 USD after fees, against 1.03 USD for an honest control. Eight alternating single-sided cycles against a pool with 1,207,876.73 USD of NAV took the NAV down to 1,147,194.99, a loss of 60,681.74 USD (5.02 percent), for an attacker gain of 56,755.02 USD, and the accumulator's accumulatedSlippage and lastTradeTimestamp stayed at 0 throughout. In a thinner pool (USDC/USDT 0.05 percent) pushing the price costs under 15 USD. A negative control shows a skew of 130 ticks reverting with the band message and 95 ticks accepted. The position of about 100,000 USD in the deepest pool is barely profitable for the attacker (30.26 USD), so the cycle needs a larger position or a thinner pool.

Impact. A rogue or compromised trader or manager can bleed the pool by about 0.4 to 0.65 percent of the capital deployed per cycle, and nothing in the code limits the number of cycles. The pool's loss is the attacker's gain to within the swap fees.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/guards/contractGuards/uniswapV3/UniswapV3NonfungiblePositionGuard.sol#L137-L150

```
if (method == INonfungiblePositionManager.mint.selector) {
INonfungiblePositionManager.MintParams memory param = abi.decode(
getParams(data),
(INonfungiblePositionManager.MintParams)
);
require(
poolManagerLogicAssets.isSupportedAsset(param.token0),
"Frgmnt: unsupported token0"
);
require(
poolManagerLogicAssets.isSupportedAsset(param.token1),
"Frgmnt: unsupported token1"
```

RECOMMENDATION

Add loss accounting to liquidity provision: record the valued entry state of each pool-minted or increased position and compare it when liquidity is later decreased or collected, so a repeated sandwich cycle is caught as a loss instead of passing as fresh noise inside the price band. Route mint and increaseLiquidity through the SlippageAccumulator so LP operations share the cumulative slippage budget with swaps, and split the spot-price band from the output-slippage floor as recommended in M-08 so tightening one does not disable the other.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/5

Commit 689e40d (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/689e40d) accounts the tokens a mint or increaseLiquidity actually consumes against the fair-priced value of the resulting liquidity. Follow-up d0a8729 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/d0a8729) shares that loss budget with router swaps through the SlippageAccumulator. The guard must be redeployed with the accumulator address and re-registered. Pull request #5 does not send that transaction.

6.2.15 Reference attester service lets the requesting user choose which assets a plan draws from

LOW CVSS 3.1 FIXED

DESCRIPTION

Section 3.1 of the design document explains why a user must not choose the asset mix of a withdrawal: each rational user takes the healthiest and most liquid assets first and leaves the impaired or illiquid ones to whoever exits later, moving loss from early leavers to remaining holders. A dedicated attester exists to choose the composition instead. The reference service accepts an optional assets list in the request body and composes the plan only from the named assets, so the requester controls the composition. The contract cannot tell the two cases apart because it only checks the plan against the signed value band, so the protection against adverse selection exists only if the service enforces it.

Impact: remaining holders absorb the loss. It scales with the gap between oracle and market price on an allowed asset, the plan size and the pool volume cap, and it is reduced by the usage surcharge. The contract itself is unaffected.

Any user can send the request if the service is exposed as written, and a pricing gap on an allowed asset (a depeg, a stale feed, an illiquid yield-bearing position) is the only precondition. Rated Low because it is off-chain reference code, the gain is bounded by the cap and surcharge, and the fix is local. Not Informational because it contradicts a stated security property of the feature and moves real money between holders.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/services/attester/src/composer.ts#L82-L93

```
const requested = request.assets?.map((a) => getAddress(a));
if (requested) {
if (requested.length === 0 || new Set(requested).size !== requested.length) {
throw new RefusalError(
'ASSET_NOT_ALLOWED',
'assets must be a non-empty list without repeats',
);
}
for (const a of requested) {
```

```
if (!allowed.has(a)) throw new RefusalError('ASSET_NOT_ALLOWED', `asset ${a} is
not allowed`);
}
}
```

RECOMMENDATION

Derive the asset set only from server configuration and ignore the request-level assets field. If users must be able to exclude assets, have the service compute the same pro-rata composition and apply a haircut for any deviation. If the behavior is intended, say so in the design document and docs/security.md and rely on the cap and surcharge, in which case this becomes a documentation finding.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/1

Commit 5f3d97b (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/5f3d97b) removes the request-level assets field. composePlan derives the asset set only from ServiceConfig.allowedAssets, and an assets field in a raw HTTP body is ignored.

6.2.16 A token-only plan in a pool with a debt-bearing position leaves accountedAssets below the gross NAV, so the next accrual mints a fee and rewards from the exiter's haircut

LOW CVSS 3.7 FIXED

DESCRIPTION

accountedAssets is the high-water mark used to recognize yield. A plan withdrawal lowers it by the value taken out plus the leaving claims' share of the existing overhang (the FNA-42 remediation), computed through computeAccountedAssetsReduction on the net-realizable basis, meaning after the modeled unwind cost of debt-bearing Aave V3 and Morpho positions. The accrual step ratchets accountedAssets up to the gross NAV instead. When a plan draws only plain tokens, which is all the reference attester composes, the gross NAV falls by the value taken while accountedAssets falls by that value plus the exiter's share of the unwind cost. The next accrual reads that difference as yield and mints a performance fee to the manager and rewards to stakers with no gain in assets.

The design document sections 3.10 and 3.11 say the bookkeeping keeps accountedAssets equal to NAV with no gap the next accrual would read as yield. That holds on the net basis only. The validated queued path (finalizeCashWithdraw) has the same effect, so this is a gap in the FNA-42 fix rather than a new behavior introduced by the plan path alone.

Relationship to the CertiK reports: two September-report findings sit in this family. FNA-26, "Haircut Cash Withdrawal Understates Accounted Assets And Mints Phantom Yield" (Medium, resolved in commit 667536d), fixed the queued claim subtracting the full pre-haircut amount from accountedAssets after the reserve was already excluded from the active baseline. FNA-42, "Underwater Withdrawals Overstate The Remaining Accounting Baseline" (Minor, resolved in

commit 0c60703), added the overhang retirement that CertiK confirmed with the words "Both withdrawal paths now also retire the exiting claim's share of any outstanding overhang". This finding is not a re-report of either. The FNA-26 root cause no longer exists, and the FNA-42 reduction does run on the plan path. The residual defect is that the reduction is computed on the net-realizable basis while the accrual ratchets to the gross NAV, so an exit that leaves the modeled unwind cost behind reopens a gap between the two bases that both fixes left unhandled. The effect is one to two orders of magnitude smaller than FNA-26's, which is why this is rated Low rather than Medium.

Impact per plan is n * U / T where n is the fUSD burned, T the total claims and U the modeled unwind cost. With 2,000,000 of leveraged equity in 10,000,000 of claims at a modeled cost of 0.75 percent, a plan of 1,000,000 recognizes 1,500 of phantom yield, of which 300 becomes the manager fee at a 20 percent performance fee. No principal leaves the pool and the total is bounded by the volume cap times U / T.

Preconditions: the pool holds a debt-bearing Aave V3 or Morpho position (the live pool's Aave account is empty today), the attester composes token-only plans, and the feature is enabled. Rated Low because the amount is a fraction of a percent of plan volume, no principal is lost, and the pro-rata path is unaffected. Not Informational because it contradicts a stated invariant, mints fUSD without assets behind it, and repeats on every honest token-only plan once a leveraged position exists.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/PoolLogic.sol#L1188-L1203

```
The plan-path reduction at [https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/
b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/PoolLogic.sol#L1188-
L1203](https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/
b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/PoolLogic.sol#L1188-
L1203) inside withdrawCashImmediateWithPlan ([L1170](https://github.com/frgmnt-
dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/
PoolLogic.sol#L1170)):
uint256 reduction = FundCalculationLibrary.computeAccountedAssetsReduction(
result.netFusd,
result.totalClaims,
accountedAssets,
result.completeFundValue,
result.valueDelta
);
if (accountedAssets < reduction) revert InvalidFundValue();
accountedAssets -= reduction;
```

RECOMMENDATION

Compute the plan-path reduction against the gross active NAV that the accrual uses, that is the gross NAV drop plus n * max(accountedAssets - grossNavBefore, 0) / T. This can live entirely in WithdrawalPlanLib, which also frees PoolLogic bytecode (the contract has 150 bytes of headroom under the size limit). Apply the same change to finalizeCashWithdraw, and correct sections 3.10 and 3.11 to state which basis they describe.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/1

Commit 6ad0be9 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/6ad0be9) sizes the accountedAssets reduction against the gross active NAV on both the attested-plan path and finalizeCashWithdraw, which is the basis _accrueYield compares against.

6.2.17 A dust queued request lets any fUSD holder block plan draws of a deposit asset in queue mode

LOW CVSS 3.1 FIXED

DESCRIPTION

A plan may not draw an asset that has Pending queued withdrawal requests, so a plan cannot starve finalization. The check is unconditional: any non-zero pendingCashWithdrawCount[asset] makes every plan that includes that asset revert with AssetHasPendingWithdrawRequests. requestCashWithdraw has no minimum size and requests cannot be cancelled. In queue mode any fUSD holder past the exit cooldown can queue 1e12 wei of fUSD (0.000001 fUSD) against a deposit asset and block plan draws of that asset. Only the manager can clear it, one request per finalizeCashWithdraw call, and the same holder can open another immediately.

A second variant needs no repeat. A one-unit request on a 6-decimal asset becomes unfinalizable after a price move of one basis point, because the asset amount rounds to zero and finalizeCashWithdraw reverts with ZeroAmount, so the block cannot be cleared until the price moves back.

Relationship to the CertiK reports: the March-report finding FRG-48, "Small Withdraw Requests Can Lock FUSD Forever" (Minor, resolved), and FRG-72, "Rounding To Zero In Immediate Withdrawal Causes Irrecoverable FUSD Loss" (Minor, resolved), established the producer: requestCashWithdraw accepts any amount above zero. The audited commit still rejects only amount == 0, so no minimum request size or cancel exists at commit b464064. This finding is not a re-report of either, because both concerned losses to the requester on paths that already existed. The new unconditional earmark check in WithdrawalPlanLib._validateAllocation turns the same producer into a permissionless block on the new plan path, a primitive neither report could have covered because the plan path did not exist at the time.

Impact: the plan path, which the design document calls the only immediate exit in queue mode, cannot draw the deposit assets, which are typically the most liquid ones. Users fall back to the queued flow, which is the behavior before this feature. No funds are lost. The cost to the attacker is dust fUSD plus gas per asset.

Preconditions: the pool must be in queue mode when the request is made, or hold a Pending request from an earlier queue window. On the live USD pool immediate withdrawals are enabled and lastRequestId() is 0 (both re-read on chain on 6 October 2026), so it is not exploitable there today. Rated Low because only the availability of one new path is affected and the manager can clear or switch modes. Not Medium because the live pool is not in queue mode and the queued fallback still works, though Medium would be justified if queue mode were the normal operating state.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/utils/WithdrawalPlanLib.sol#L838-L844

```
if (guard != alloc.guard) revert IPoolLogic.GuardMismatch();
if (IPoolLogic(address(this)).pendingCashWithdrawCount(asset) != 0) {
revert IPoolLogic.AssetHasPendingWithdrawRequests();
}
```

RECOMMENDATION

Protect only what finalization needs: require that after the plan the pool still holds reserved + sum(pending amounts) of the asset, which needs a per-asset pending sum. Alternatively add a minimum request size and a manager cancelCashWithdraw, and let finalizeCashWithdraw cancel a request whose asset amount rounds to zero. Both options touch PoolLogic, which has 150 bytes of headroom, so any new code there needs an equal amount moved into a library.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/1

Commit a1657b6 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/a1657b6) replaces the unconditional pending-request block with a check that, after the plan draws an asset, the pool still holds enough to cover the reserved balance plus every pending request for that asset.

6.2.18 The attested-only upgrade path leaves validated protections inactive and cannot be completed by the core upgrade script

LOW CVSS 2.2 FIXED

DESCRIPTION

Two scripts can upgrade the live PoolLogic. The core script upgrades PoolLogic, TokenLogic, AssetHandler and PoolManagerLogic in one atomic Safe batch of nine calls. The attested script upgrades PoolLogic only and is state aware: it reads live state and drops initializer calls that already ran. The core script is not state aware. It always emits the PoolLogic upgrade together with initializeAutoCompounding, initializeWithdrawalEscrow and, when an attester is configured, initializeAttestedWithdrawal. The documents cover the order that works and the case where both batches are generated before either executes, but not the opposite order, which is the natural one if the attested feature ships first.

After the attested-only batch has executed, a freshly generated core batch reverts on its calls 6, 7 and 8 with InvalidInitialization, EscrowAlreadySet and InvalidInitialization, so the AssetHandler, PoolManagerLogic and TokenLogic upgrades, the EUR/USD aggregator lock, the sequencer feed and the deposit cap cannot be applied with the shipped tooling. A hand-trimmed batch of calls 1 to 4 and 9 works.

The attested-only state is also weaker than it looks. Deposits, staking, withdrawals, queued requests and plans all keep working on it, because FundCalculationLibrary falls back to totalFundValue() when the old PoolManagerLogic lacks totalFundValueWithCompleteness(). That fallback reports every NAV as complete, so the withholding of yield and fees on an incomplete NAV that CertiK FNA-04, "Failed External Position Valuation Is Consumed As Complete NAV" (Medium, resolved), introduced is inert. The old TokenLogic never calls checkpointFeesForDeposit(), so the pre-deposit fee settlement that resolved CertiK FNA-22, "New FUSD Deposits Are Retroactively Included In The Management-Fee Base" (Minor, resolved, confirmed fixed with the checkpoint reverting a deposit on failure), is inert as well. There is also no deposit cap, no sequencer check (the protection whose disabled deployment was FNA-50, "Base Sequencer Uptime Protection Is Deployed In A Disabled State", Minor, resolved) and no EUR/USD aggregator lock from FNA-40, "Global EUR Conversion Mode Changes Desynchronize Pool Accounting" (Minor, resolved), until the core upgrade lands. The interim state simply equals the live state today, which is why this is rated Low and not higher: the finding is about the tooling failing to complete the step that activates these protections.

Impact: a delayed rollout, a wasted signing round and pressure to hand-edit a batch for a production ProxyAdmin. No funds are lost. Rated Low because there is no on-chain exposure beyond the interim state, which equals the live state today, and a hand-trimmed batch recovers it. Not Informational because the step that fails is the one that applies several validated protections and the documents only cover the order that works.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/scripts/upgrade_core_contracts.ts#L391-L439

RECOMMENDATION

Make the core script state aware in the same way as the attested script: skip the PoolLogic upgrade and the three initializers when compoundedRewardIndex(), withdrawalEscrow() and withdrawalAttester() are already set. Alternatively state in the documents that the attested script is a dead end for the full stack and remove it from the offered paths. Add the ordering constraints to the runbook: TokenLogic after PoolLogic, and the PoolLogic initializers in the same batch as its upgrade.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/1

Commit 49957dc (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/49957dc) makes upgrade_core_contracts.ts read live state and emit each PoolLogic post-upgrade initializer, and the escrow deployment, only when that step has not already run.

6.2.19 A floored NAV hides a direct-leg over-draw when the deficit comes from another guard

LOW CVSS 3.1 FIXED

DESCRIPTION

completeFundValue is the sum of the guard balances minus the sum of the guard deficits, floored at zero. The plan's two NAV bounds are measured on it before and after the allocation loop. When the plan's own draw takes the floored NAV to zero, valueDelta is only completeBefore, however much real value left the pool. The receipt-side check values only legs delivered as a token. Legs that a guard pays through its own transactions (the Aave V4 Spoke, Uniswap V3, Morpho without debt and Aave V3 without debt) are invisible to it. The client's accepted residual KI-ACC-02 justifies this because direct-leg guards refuse debt, so no deficit is involved, but the deficit is a pool-level sum over every guard: a leveraged position in another guard can be underwater by d while the direct-leg guard holds a debt-free position.

In the basic form the last claimant draws the debt-free position beyond their NAV entitlement by up to d. In the wider form, when the floored NAV is at or below the 1e15 dust tolerance, any holder of the 0.01 fUSD minimum can draw the whole debt-free position, because the NAV cannot fall further so the band is met, and the volume breaker meters the 0.01 fUSD burned and not the value released.

Relationship to the CertiK reports: FNA-05, "Underwater Immediate Redemptions Create A First-Redeemer Bank Run" (Medium, resolved), and FNA-30, "Temporary Illiquidity Is Converted Into A Permanent Redemption Loss" (Minor, resolved), are the closest September-report findings, and the mechanism here is deliberately the same family, an exit favored by the accounting while the pool is impaired. It is not a re-report of either. Those findings concerned the haircut sizing of the pre-existing pro-rata path and were resolved by the haircut and overhang accounting (FNA-42 among them). This finding is about the new plan path only: the band is measured on a NAV floored at zero, and the

receipt-side check cannot see direct legs, so the premise of the accepted residual (that no deficit is involved in a direct-leg draw) does not hold when the deficit sits in another guard. None of the September fixes bound a floored-NAV direct-leg draw.

Preconditions: a catastrophic leveraged loss in a deficit-reporting guard that cancels most or all of the pool, a debt-free position behind a direct-leg guard, and a compromised or careless attester key that signs a plan with a whole-asset direct leg at 100 percent. The reference attester service composes token legs only and cannot produce that plan, and on the live pool the Aave account is empty and the pool is essentially one Morpho position, so the conditions are not met today. Rated Low because it needs a pool-wide leveraged loss and a compromised attester. Not Medium only for that reason, though Medium is arguable since the attester key can defeat the one bound the trust model says always holds.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/utils/WithdrawalPlanLib.sol#L525-L531

```
if (completeBefore < completeAfter) revert IPoolLogic.InvalidFundValue();
result.valueDelta = completeBefore - completeAfter;
if (result.valueDelta > target + DUST_TOLERANCE) {
revert IPoolLogic.ValueConservationViolated();
}
uint256 minAllowed = target - (target * plan.minValueOutBps) / 10_000;
if (result.valueDelta < minAllowed) revert IPoolLogic.ValueConservationViolated();
{
uint256 delivered;
for (uint256 i = 0; i < result.outAssets.length; ++i) {
delivered += IPoolManagerLogic(input.poolManagerLogic).assetValue(
result.outAssets[i],
```

RECOMMENDATION

When the post-loop completeAfter is zero and the plan has a direct leg, revert with ValueConservationViolated, because the outflow of a direct leg cannot be measured once the NAV is floored. It is one line placed right after the completeBefore < completeAfter check and it lives in the library, so it needs no PoolLogic bytecode. The cost is that a last claimant who legitimately empties the pool through a direct leg falls back to the pro-rata path. An alternative is to measure the plan on the gross - deficit figure before the floor.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/1

Commit eac8d5f (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/eac8d5f) reverts with ValueConservationViolated when the post-loop NAV is exactly zero and the plan has a direct leg. A last claimant who empties the pool through a direct leg falls back to the pro-rata path.

6.2.20 The emergency stop is not independent of the manager, or arrives after a rotation matures, in configurations the repository's own scripts produce

LOW CVSS 3.3 FIXED

DESCRIPTION

The factoryOwner's latched stop is the only control of the attested-withdrawal feature that does not depend on the manager (design document section 3.9). It works only when the factoryOwner is a different actor from the manager and can act faster than attesterRotationDelay. Three configurations show where that fails.

Configuration A, fresh deployment: deploy_core_contracts.ts sets the manager and the factoryOwner to the same address, so setAttestedWithdrawEnabled takes the manager branch for that address, the factoryOwner branch is unreachable and nothing ever latches. The address can disable, re-enable and rotate at will. This is the state of every fresh pool, including the EUR product, until the manager role is moved.

Configuration B, the planned lock-down: transferRoles_Governance.ts moves PoolManagerLogic.factoryOwner behind the 48-hour Timelock. A stop then needs a scheduled operation plus 48 hours, while a manager rotation needs only attesterRotationDelay, where 24 hours is both the floor and the script default. The stop executes about 25 hours after the rotation has matured, it does not revoke an attester that is already active, the factoryOwner has no function to propose or revoke an attester, and changeManager is manager-only.

Configuration C, the live custody: the manager Safe (2 of 3) and the factoryOwner Safe (3 of 4) share two signers, and those two signers meet the manager Safe threshold on their own. All addresses below were re-read on chain on 6 October 2026 and match. CertiK's FNA-01 response states the same baseline fact from the client's side: the Timelock "was deployed alongside the rest of the stack but never actually wired as owner of anything", and the planned wiring is exactly the transferRoles_Governance.ts run that configuration B analyzes. This finding is therefore not a re-report of FNA-01 (acknowledged, centralization). FNA-01 documents that the roles exist and are powerful. This finding identifies the specific latency and role-assignment configurations, produced by the repository's own scripts, in which the one control designed not to depend on the manager does not function.

PoolManagerLogic.manager() is 0x18f84Ea9D3351081780d817e5edd554112CF81Ab, a Safe with a 2 of 3 threshold. factoryOwner(), PoolLogic.owner() and the three ProxyAdmin owners are 0x74aF72D91D5FB263fBa09Ed43aD1C1ea079058B3, a Safe 1.4.1 with a 3 of 4 threshold.

Impact: during the exposure window a colluding manager and attester can steer asset composition but not magnitude, bounded by the volume cap, the 1 percent band and the surcharge ceiling. The independent control the design relies on does not exist, or arrives late. Rated Low because it needs a trusted role to misbehave and the damage is composition skew, not loss of funds. Medium is justified if governance confirms it will run transferRoles_Governance.ts as written without raising the rotation delay.

CODE LOCATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/blob/b4640647afe7761183257d4a729ab28663ea05ac/contracts/contracts/PoolLogic.sol#L1070-L1085

```
function setAttestedWithdrawEnabled(bool enabled) external {
if (msg.sender != _manager()) {
if (msg.sender != IPoolManagerLogic(poolManagerLogic).factoryOwner()) {
revert OnlyManager();
}
attestedWithdrawOwnerStopped = !enabled;
if (enabled) return;
} else if (enabled && attestedWithdrawOwnerStopped) {
revert AttestedWithdrawOwnerStopActive();
}
isAttestedWithdrawEnabled = enabled;
if (!enabled) {
```

RECOMMENDATION

Add a preflight to both upgrade scripts that reads factoryOwner() and manager(), refuses when they are equal, and, when the factoryOwner is a contract exposing getMinDelay(), requires ATTESTER_ROTATION_DELAY_SECONDS to exceed that delay plus a monitoring margin. Or make the stop executable by a fast guardian role separate from the Timelock-owned factoryOwner, and let a factoryOwner stop also revoke the active attester. Document the invariant in section 3.9.

MITIGATION

https://github.com/frgmnt-dao/Frgmnt_fUSD/pull/1

Commit 01a0851 (https://github.com/frgmnt-dao/Frgmnt_fUSD/commit/01a0851) makes a factoryOwner stop revoke the active attester, prints a required follow-up when a fresh deploy leaves manager and factoryOwner as the same address, and refuses to move factoryOwner behind a Timelock whose delay does not exceed attesterRotationDelay.

7. Executive Summary

Two independent softstack experts performed an unbiased and isolated audit of the Frgmnt fUSD smart contracts provided by GRANIT FINANCE SAS. The main objective of the audit was to verify the security and functionality of the protocol. The audit process was a manual code review together with automated security testing.

The audit identified 20 issues:

- No critical issues were found
- No high severity issues were found
- 14 medium severity issues were found
- 6 low severity issues were found
- No informational issues were found

Update (20.10.2026): 19 findings are Fixed. M-05 is Acknowledged as an accepted residual.

Token Deployments

Network	Contract Address	Explorer
Base (fUSD)	`0xeB82611A2B2dC9FBEAF5903d5decDf801765B759`	View on Explorer
Base (sfUSD)	`0x704c56974e0CA4BF8ff8fe8acc51FBF1E053878E`	View on Explorer

8. About the Auditor

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

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

Underpinned by a team of industry veterans possessing robust technical knowledge in the Web3 domain, softstack offers industry-leading smart contract audit services. Committed to evolving with their clients' ever-changing business needs, softstack's approach is as dynamic and innovative as the industry it serves.

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

9. Glossary

Term	Definition
Aave	Lending protocol integrated through dedicated contract and asset guards
Attester	Off-chain signer that chooses the asset mix of an attested withdrawal plan
Base	Ethereum layer-2 network where the protocol is deployed (chain id 8453)
CVSS	Common Vulnerability Scoring System
Chainlink	Oracle network used for USD prices and the Base sequencer uptime feed
ERC-20	Ethereum token standard for fungible tokens
Guard	Contract that validates a vault transaction or values a position before it executes
Morpho	Morpho Blue lending market integrated through dedicated guards and a market allowlist
NAV	Net asset value of the vault, used to size yield, fees and withdrawals
OZ	OpenZeppelin, the Solidity library used for upgradeability, access control and reentrancy protection
ProxyAdmin	Owner contract of an OpenZeppelin Transparent proxy; it can replace the implementation
SlippageAccumulator	Contract that caps cumulative value loss on guarded swaps
Timelock	DAO contract that delays privileged calls, documented at 48 hours
UUPS	Universal Upgradeable Proxy Standard, the upgrade pattern used by TokenLogic
Uniswap V3	Decentralized exchange used for swaps and liquidity positions
fUSD	Frgmnt USD, the protocol's collateral-backed stablecoin
sfUSD	Staked fUSD, the non-transferable vault receipt that accrues yield