Velvet Capital

Core v3 Contracts

SMART CONTRACT AUDIT

13.07.2024

Made in Germany by Softstack.io

Table of contents

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 Velvet Solutions, Inc.. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (16.05.2024)	Layout
0.4 (28.05.2024)	Automated Security Testing Manual Security Testing
0.5 (05.06.2024)	Verify Claims and Test Deployment
0.6 (07.06.2024)	Testing SWC Checks
0.9 (21.06.2024)	Summary and Recommendation
1.0 (21.06.2024)	Final document
1.1 (21.06.2024)	Re-check 849629b1aacf32d84634d8c4ef1378527bce3bb3
1.2 (09.07.2024)	Re-check 2dcca2bd610f847dc88b2b5692e19edc72f92d68

2. About the Project and Company

Company address:

Velvet Solutions, Inc.
251 Little Falls Drive
Wilmington, DE 19808
USA

Website: https://www.velvet.capital

LinkedIn: https://www.linkedin.com/company/velvetcapital

Twitter (X): https://twitter.com/velvet_capital

Discord: https://discord.gg/GkEwgezVMR

Telegram: https://t.me/velvetcapital

2.1 Project Overview

Velvet Capital is an innovative platform revolutionizing DeFi asset management by providing a suite of tools aimed at facilitating the management and launch of on-chain funds, DeFi structured products, and tokenized portfolios. Backed by Binance Labs, Velvet Capital integrates intent-based architecture and account abstraction to allow for seamless trade execution, yield farming, and strategy automation across multiple blockchain ecosystems.

Addressing key pain points faced by professional investors, such as lack of liquidity, complex compliance requirements, and the need for advanced security measures, Velvet Capital offers superior execution, smart routing, and multi-chain asset management capabilities. The platform’s API layer supports the automation of trading strategies, while its non-custodial vaults maintain asset safety without compromising on management flexibility.

Additionally, Velvet Capital extends its reach to real-world assets and plans to incorporate on-chain derivatives and borrowing functionalities. Security audits and real-time monitoring ensure the integrity of operations and asset safety. As a community-driven ecosystem, Velvet also involves its users in governance through the Velvet DAO, which influences the platform's evolution and future developments.

In essence, Velvet Capital stands out as a comprehensive, professional-grade DeFI operating system designed to meet the needs of both seasoned professionals and DeFi enthusiasts, facilitating the efficient management of diverse assets and enhancing the adoption of decentralized finance.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to softstack to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to softstack describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

File	Fingerprint (MD5)
./handler/ExternalSwapHandler/EnsoHandlerBundled.sol	7dd9f6ada37416c0d072b6ddff37b4b5
./handler/ExternalSwapHandler/EnsoHandler.sol	d629b2fc89f706de6bce2dac41cc5f8d
./handler/IIntentHandler.sol	9315295ccecd0bc36c21f5105dd547b8
./core/token/PortfolioToken.sol	6144b3532050bda8205854595f4f9cbd
./core/access/AccessModifiers.sol	82137cd7f6465391a06527dd6be16039
./core/cooldown/CooldownManager.sol	f47d1f8e4b2fba79502431c3e3f51131
./core/config/VaultConfig.sol	6382a622ab1eb13deb5fb47bef9aab36
./core/config/Dependencies.sol	d58865c7a72f93f3fa14ff425c11bfc8
./core/checks/ChecksAndValidations.sol	9fc2c2259d92abde30cd906a7ecc7521
./core/management/FeeManager.sol	ecd65153646233a488ef5a6f541bb1dd
./core/management/TokenExclusionManager.sol	1c6253e8b92f84c1b482aa332925fc78
./core/management/VaultManager.sol	49adde0a00b83cdaceb70323c1c11abe
./core/user/UserManagement.sol	8d162ef078bde53120cdf073d519bbe8
./core/calculations/VaultCalculations.sol	7e99162b9b82241018a2d45f5e46ee9f
./core/calculations/MathUtils.sol	9df11927c17f2641f212ea8b7d8ae627
./core/calculations/TokenBalanceLibrary.sol	ce47ea42d913ca0e85015c65bfebdc0e
./core/calculations/TokenCalculations.sol	f8704df1646ee3c2bd6c1dd78bc6f737

./core/Portfolio.sol	e28c113292734096c0b6bac641bbc47b
./core/interfaces/ITokenExclusionManager.sol	8009ff7ed774265c22cccab19e92c836
./core/interfaces/IGnosisSafe.sol	258c9c2ad863f87f06e1d94d78606175
./core/interfaces/IMultiSend.sol	8db2058d224335c1ba179a1b95ec6992
./core/interfaces/IAllowanceTransfer.sol	5b082b61b1b7eec3aa87ff8d5987641d
./core/interfaces/IEIP712.sol	3a4b1b6ad9f2b9ea39d752ee63847e6a
./core/interfaces/IPortfolio.sol	e34197a092dc2ecb007c9cd0bea6f485
./fee/FeeConfig.sol	dd49e3223ef99b4ddf60a503988bf012
./fee/FeeEvents.sol	fb7075bad92a050554b090f546daaf16
./fee/FeeCalculations.sol	2012644ffc110e773f95cff706ae68ae
./fee/FeeModule.sol	f45605c7a2ef6acf387956b7e569a4be
./fee/IFeeModule.sol	67ac1cd9092e8229bb0f91c858d2d9d1
./access/IAccessController.sol	7ba40fd6b9933adf7a52a934d5e31ecb
./access/AccessRoles.sol	e240f081410d1e6651a0381446aee3f3
./access/AccessController.sol	4491ed6782e6a55580df1ea439b96566
./config/assetManagement/TreasuryManagement.sol	9eae328858b7935f6ffac65a614159ee
./config/assetManagement/IAssetManagementConfig.sol	66708c0a2f2cf3d64be9a19fba8990b4
./config/assetManagement/AssetManagerCheck.sol	8ac6bbe1a074ec39ea040b410a1003f3
./config/assetManagement/FeeManagement.sol	48856478b3bd82048fd631f0b837f524
./config/assetManagement/TokenWhitelistManagement.sol	68622d8bbb65db2342591b60520daba4
./config/assetManagement/AssetManagementConfig.sol	19ed1963ec9a6f0796d1facb3c13d931
./config/assetManagement/UserWhitelistManagement.sol	20ad91ecc9663a5e1c1c81a511cf5912
./config/assetManagement/PortfolioSettings.sol	08891efc47f85cdad03e0f8cc9211f54
./config/protocol/OracleManagement.sol	cd2bd1d91623be33eab398441292ea74
./config/protocol/ProtocolTreasuryManagement.sol	e7c26cd2724412d326f397f7f7db8658
./config/protocol/OwnableCheck.sol	62f9f2eff640ae58dd8da08c2a85f9c1
./config/protocol/SystemSettings.sol	b93ac2b085c7185f7a70fb1aab0614bb

./config/protocol/ProtocolConfig.sol	0206bc8cd1d17113d67367c57ddc80f5
./config/protocol/IProtocolConfig.sol	3846025d869c643a725a8b18213f5a22
./config/protocol/SolverManagement.sol	e0a8db36f7798ee3806b0f41a11791e4
./config/protocol/ProtocolFeeManagement.sol	2f36a95349e52c6c62c312a415d116ea
./config/protocol/TokenManagement.sol	e0ed945bd05ceb11935cdc4c2b985bea
./front-end-helpers/IPortfolioFactory.sol	bb93ca8664429040c5f680c993ee226c
./front-end-helpers/IUniswapV2Router02.sol	50edefb5a2696ccb2b2baf146aeb926b
./front-end-helpers/PortfolioCalculations.sol	c57b8fece8f1f997ecdca0889d48cd4f
./oracle/PriceOracle.sol	8f5864cbfd1ec309c4e4ab434e60af5d
./oracle/IPriceOracle.sol	04fc5fe1eecdeac2c2db0395c56a2139
./oracle/PriceOracleL2.sol	87de0a227e28a90829e23ac67d5d53f8
./oracle/PriceOracleAbstract.sol	79e1ef40076f38f2ffe88b8d26784f4d
./bundle/WithdrawBatch.sol	f87deff7fba8ca770b5e50bcd2031254
./bundle/DepositBatch.sol	5e488176607796500c2d2b0d595402fe
./library/GnosisDeployer.sol	2137568a9a35197d76649e016804b0de
./library/ErrorLibrary.sol	69a4575ad12c9b17f8a813b420644bd8
./rebalance/RebalancingConfig.sol	05f40aebf36df72a71302e59bc35025a
./rebalance/Rebalancing.sol	8edde8438bf6b48ecab4226fbbb4b809
./rebalance/IRebalancing.sol	8253074668be1c2b50719d7ec08a1812
./PortfolioFactory.sol	b19ff20c00f32989702bb5584c4223a1
./FunctionParameters.sol	77653be287f88b547c80f3a576d57ab5
./vault/VelvetSafeModule.sol	303cf547ef20807192424f7ed21f9d07
./vault/IVelvetSafeModule.sol	f279fc6f540af3f8530b6cec27701908

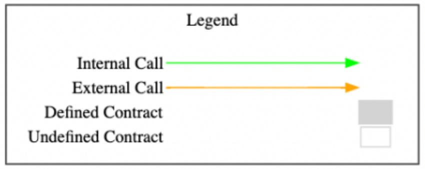
Latest contract files from re-check 2dcca2bd610f847dc88b2b5692e19edc72f92d68

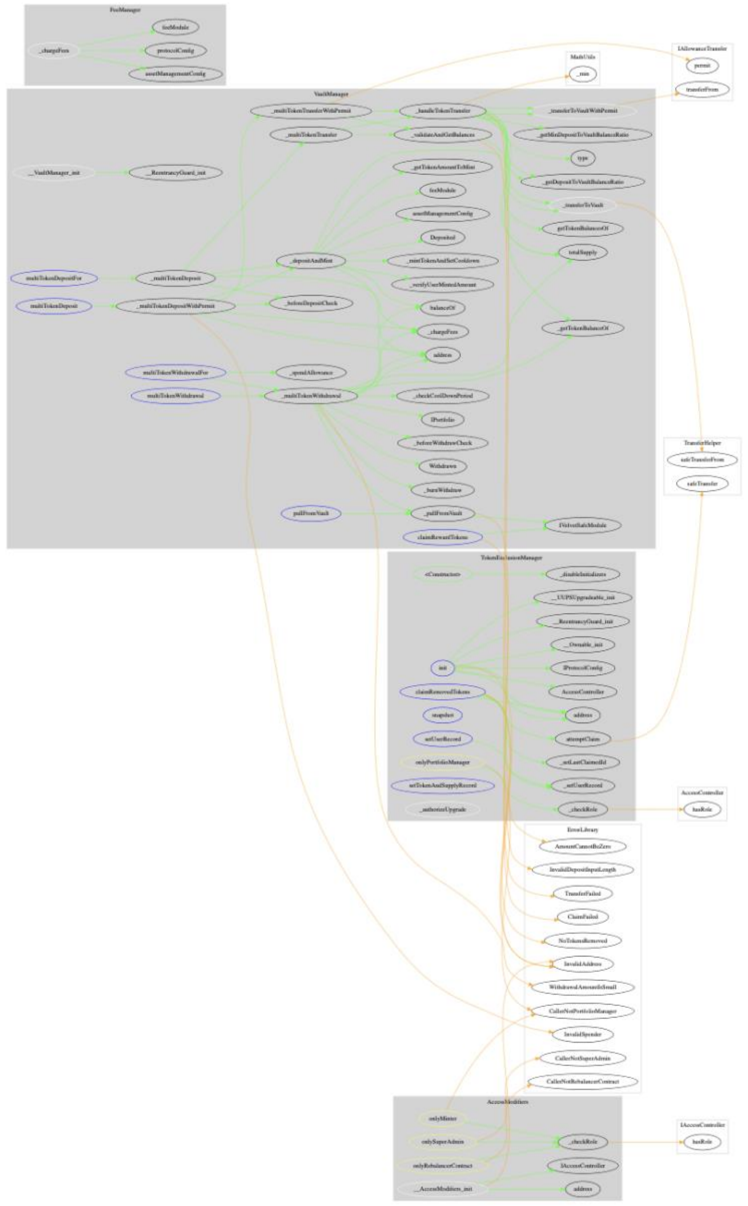
File	Fingerprint (MD5)
./contracts/handler/ExternalSwapHandler/EnsoHandlerBundled.sol	2c7ea33dd56e300dba22167cdf0d12fa
./contracts/handler/ExternalSwapHandler/EnsoHandler.sol	2b31c0294e97ef69c8f4e77a3096e4e8
./contracts/handler/IIntentHandler.sol	9315295ccecd0bc36c21f5105dd547b8
./contracts/core/token/PortfolioToken.sol	6144b3532050bda8205854595f4f9cbd
./contracts/core/access/AccessModifiers.sol	82137cd7f6465391a06527dd6be16039
./contracts/core/cooldown/CooldownManager.sol	f47d1f8e4b2fba79502431c3e3f51131
./contracts/core/config/VaultConfig.sol	585670960e3b1e0ad8a4ceb0e51e818c
./contracts/core/config/Dependencies.sol	d58865c7a72f93f3fa14ff425c11bfc8
./contracts/core/checks/ChecksAndValidations.sol	3dfe0130eccbfa2642e143b69f25b5c5
./contracts/core/management/FeeManager.sol	2a78393f6c3d20ad349ca6aa52f6500b
./contracts/core/management/TokenExclusionManager.sol	2269bd23a637db5f9c887c6b3b154039
./contracts/core/management/VaultManager.sol	b4b4ced36aa070fa40950e611dce42c6
./contracts/core/user/UserManagement.sol	8d162ef078bde53120cdf073d519bbe8
./contracts/core/calculations/VaultCalculations.sol	7e99162b9b82241018a2d45f5e46ee9f
./contracts/core/calculations/MathUtils.sol	98ff199247dfe9d1aa6f7555459eb725
./contracts/core/calculations/TokenBalanceLibrary.sol	ce47ea42d913ca0e85015c65bfebdc0e
./contracts/core/calculations/TokenCalculations.sol	f8704df1646ee3c2bd6c1dd78bc6f737
./contracts/core/Portfolio.sol	e28c113292734096c0b6bac641bbc47b
./contracts/core/interfaces/IPortfolioFactory.sol	97806311c68327f8c4d27228b084d747
./contracts/core/interfaces/ITokenExclusionManager.sol	cadbea8fe5b143a3eaf805bb0da7a4cd
./contracts/core/interfaces/IGnosisSafe.sol	ff46830b31b174ce84dcae2e770ebe17
./contracts/core/interfaces/IMultiSend.sol	2a33e5121134814994ccd6f234fa95e1
./contracts/core/interfaces/IAllowanceTransfer.sol	29e4d7e3c3c88be4ffe408ab58f94ff2
./contracts/core/interfaces/IEIP712.sol	9dd483ec7af5cd782057d9b793ea0798
./contracts/core/interfaces/IPortfolio.sol	e34197a092dc2ecb007c9cd0bea6f485

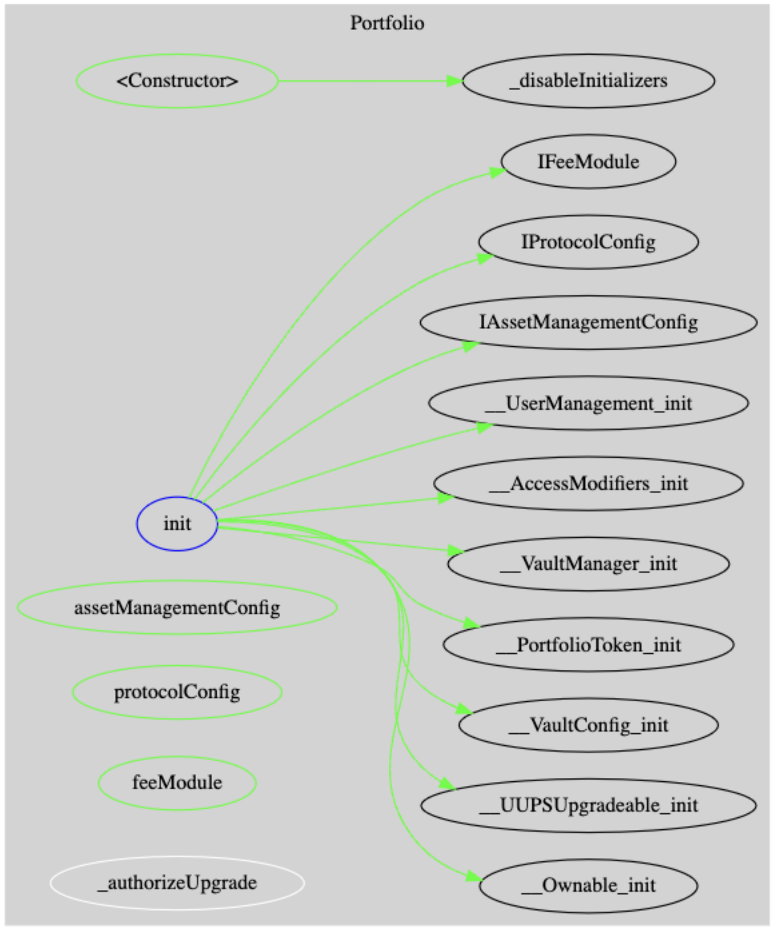
./contracts/fee/FeeConfig.sol	e4c6c7d9b4c4c7f8a7530bc2126fa0fd
./contracts/fee/FeeEvents.sol	fb7075bad92a050554b090f546daaf16
./contracts/fee/FeeCalculations.sol	2012644ffc110e773f95cff706ae68ae
./contracts/fee/FeeModule.sol	eff770df66cbc799396c066d75faf7e3
./contracts/fee/IFeeModule.sol	da804d87bc2ad9649e9d3bc30ea4b28d
./contracts/access/IAccessController.sol	7ba40fd6b9933adf7a52a934d5e31ecb
./contracts/access/AccessRoles.sol	e240f081410d1e6651a0381446aee3f3
./contracts/access/AccessController.sol	4491ed6782e6a55580df1ea439b96566
./contracts/config/assetManagement/TreasuryManagement.sol	9eae328858b7935f6ffac65a614159ee
./contracts/config/assetManagement/IAssetManagementConfig.sol	66708c0a2f2cf3d64be9a19fba8990b4
./contracts/config/assetManagement/AssetManagerCheck.sol	8ac6bbe1a074ec39ea040b410a1003f3
./contracts/config/assetManagement/FeeManagement.sol	0538b143e13242184923934b1ad5cc14
./contracts/config/assetManagement/TokenWhitelistManagement.sol	34a78d41c8207ea1db8ddd1a44cd5e5d
./contracts/config/assetManagement/AssetManagementConfig.sol	5bd3e5085c9f814fb9dfd16c3bff15ec
./contracts/config/assetManagement/UserWhitelistManagement.sol	4c702450335d0af75c1e7b5bbccc2ccf
./contracts/config/assetManagement/PortfolioSettings.sol	92261575e2555e557e941c8306f8e9ce
./contracts/config/protocol/OracleManagement.sol	cd2bd1d91623be33eab398441292ea74
./contracts/config/protocol/ProtocolTreasuryManagement.sol	e7c26cd2724412d326f397f7f7db8658
./contracts/config/protocol/OwnableCheck.sol	62f9f2eff640ae58dd8da08c2a85f9c1
./contracts/config/protocol/SystemSettings.sol	23906a7d9f05847b02c0ed3cfe47fcf0
./contracts/config/protocol/ProtocolConfig.sol	d319570c722e724378e97d7502b18577
./contracts/config/protocol/IProtocolConfig.sol	2a66cd7a62ef57676e84bbfe027aca35
./contracts/config/protocol/SolverManagement.sol	e0a8db36f7798ee3806b0f41a11791e4

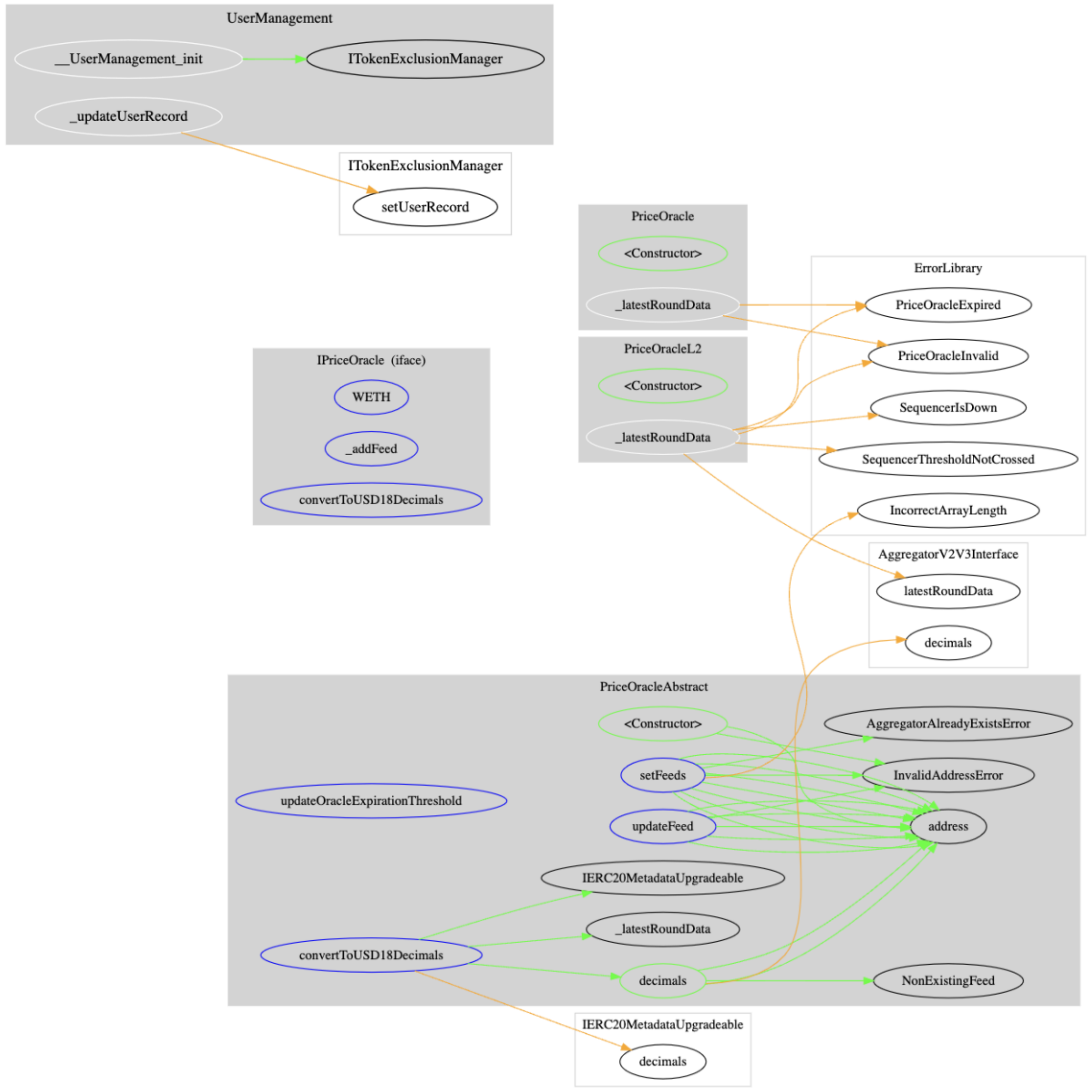
./contracts/config/protocol/ProtocolFeeManagement.sol	2f36a95349e52c6c62c312a415d116ea
./contracts/config/protocol/RewardTargetManagement.sol	0f4bf3f80e2cf76efbb7bcf6996809c5
./contracts/config/protocol/TokenManagement.sol	f086a79a347fd7f0b3cf027e6c6afa59
./contracts/front-end-helpers/IUniswapV2Router02.sol	50edefb5a2696ccb2b2baf146aeb926b
./contracts/front-end-helpers/PortfolioCalculations.sol	2e7df079437ff35f752e71517781f011
./contracts/oracle/PriceOracle.sol	8f5864cbfd1ec309c4e4ab434e60af5d
./contracts/oracle/IPriceOracle.sol	04fc5fe1eecdeac2c2db0395c56a2139
./contracts/oracle/PriceOracleL2.sol	87de0a227e28a90829e23ac67d5d53f8
./contracts/oracle/PriceOracleAbstract.sol	79e1ef40076f38f2ffe88b8d26784f4d
./contracts/bundle/WithdrawBatch.sol	e5cb3b5c3c12e4224ef460322cdd884b
./contracts/bundle/IWithdrawBatch.sol	ad20cbbf93f591bc4b7e71a6f5f65a41
./contracts/bundle/TargetWhitelisting.sol	9e55ad994814b1e06bb22599ac6537db
./contracts/bundle/WithdrawManager.sol	6f329e163960f1427778f4f9b3b0221c
./contracts/bundle/DepositBatch.sol	ad6bdf46670d382a71826a13a4af8335
./contracts/bundle/IDepositBatch.sol	69b38e7dec0603531edfc7781d7a0242
./contracts/bundle/DepositManager.sol	3ebaa5e06a86b0ee806e78f710d5636c
./contracts/library/GnosisDeployer.sol	2137568a9a35197d76649e016804b0de
./contracts/library/ErrorLibrary.sol	310a02342f82f32e172727e58bb05761
./contracts/rebalance/RebalancingConfig.sol	05f40aebf36df72a71302e59bc35025a
./contracts/rebalance/Rebalancing.sol	92faee9303113ef5e57b55478a6e1f55
./contracts/rebalance/IRebalancing.sol	6bd60edc6f9c24b3cb32f431247e7489
./contracts/PortfolioFactory.sol	0681b3f3a0f9c40186bca9736f4aa430
./contracts/FunctionParameters.sol	fc1ac7e508a482dd3ae2641f3f1e4858
./contracts/vault/TokenRemovalVault.sol	287df9ee8d8e3e38d53f0ee41c15c1d4
./contracts/vault/VelvetSafeModule.sol	303cf547ef20807192424f7ed21f9d07
./contracts/vault/ITokenRemovalVault.sol	fadcbbb2faf537a3ccbc1618ffc371cb
./contracts/vault/IVelvetSafeModule.sol	f279fc6f540af3f8530b6cec27701908

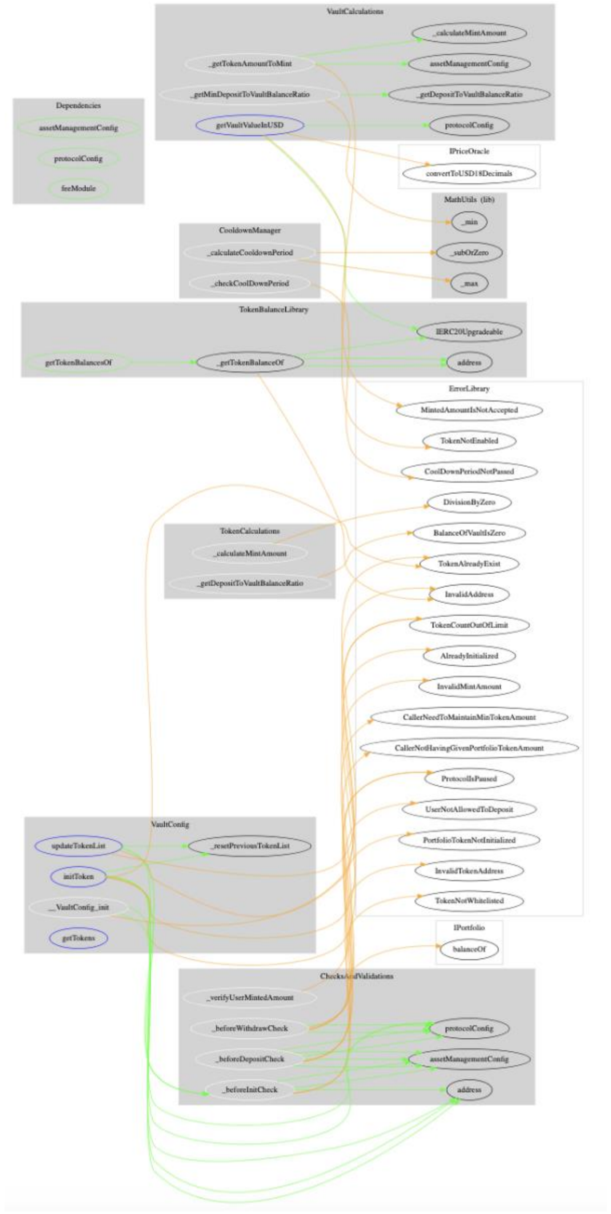
5.2 CallGraph



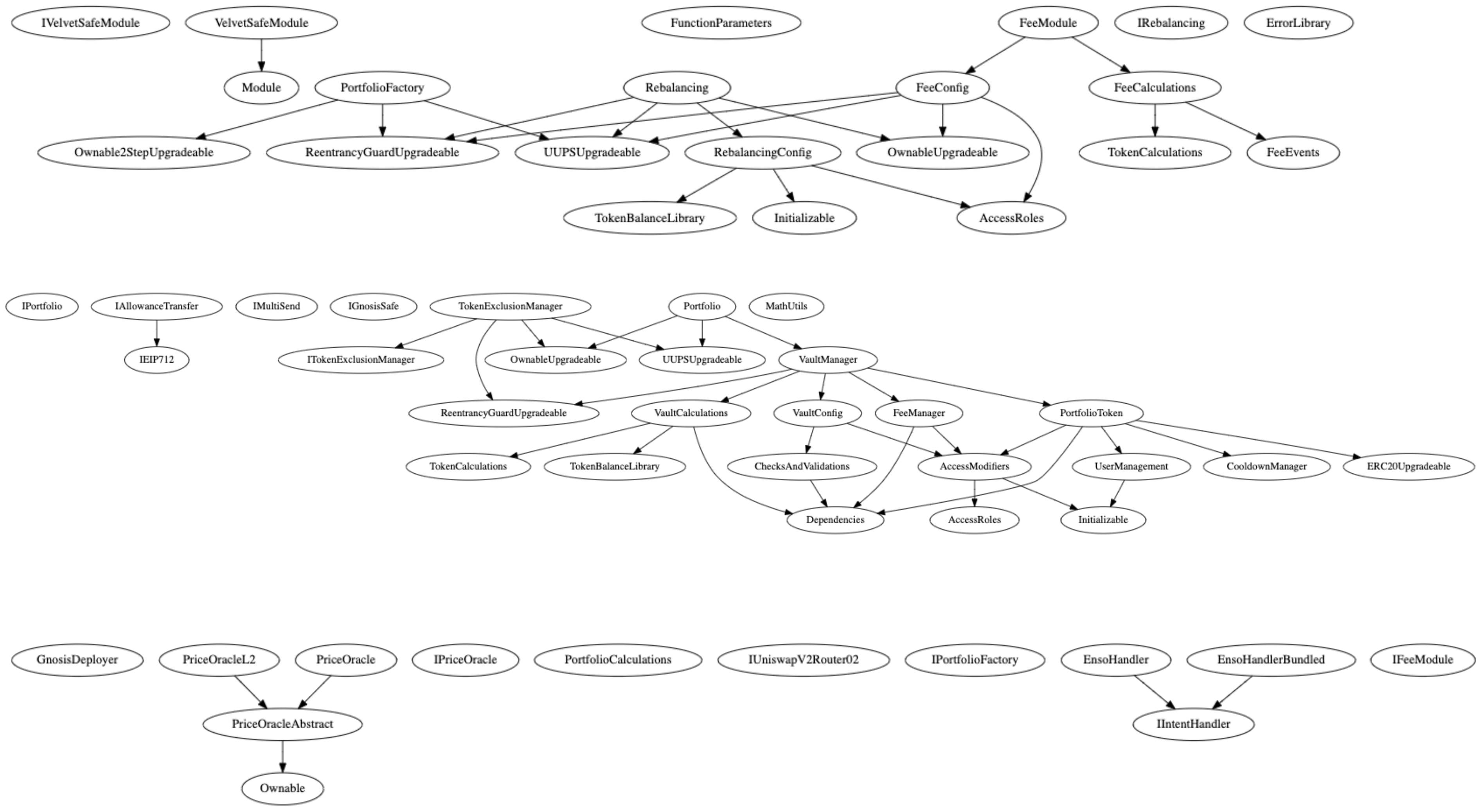




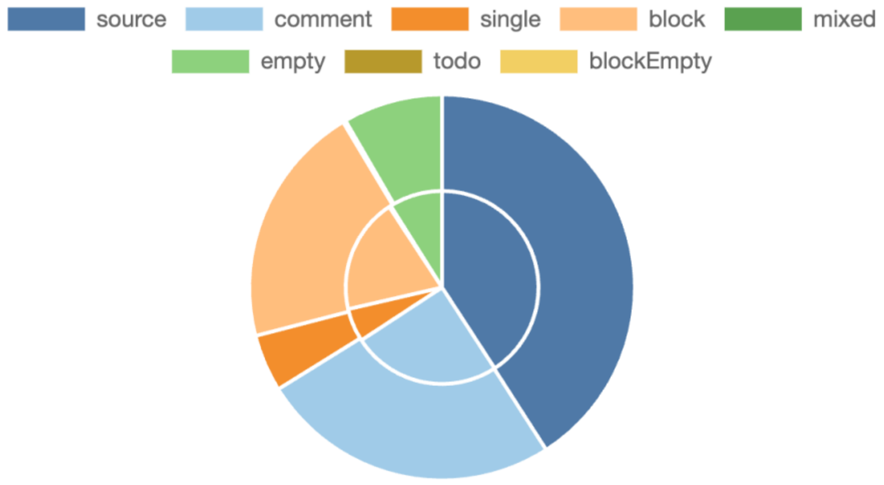

UserManagement
__UserManagement_init
ITokenExclusionManager
_updateUserRecord
ITokenExclusionManager
setUserRecord
PriceOracle
<Constructor>
_latestRoundData
PriceOracleL2
<Constructor>
_latestRoundData
ErrorLibrary
PriceOracleExpired
PriceOracleInvalid
SequencerIsDown
SequencerThresholdNotCrossed
IncorrectArrayLength
IPriceOracle (iface)
WETH
_addFeed
convertToUSD18Decimals
AggregatorV2V3Interface
latestRoundData
decimals
PriceOracleAbstract
<Constructor>
AggregatorAlreadyExistsError
setFeeds
InvalidAddressError
updateOracleExpirationThreshold
updateFeed
address
IERC20MetadataUpgradeable
_latestRoundData
convertToUSD18Decimals
decimals
NonExistingFeed
IERC20MetadataUpgradeable
decimals

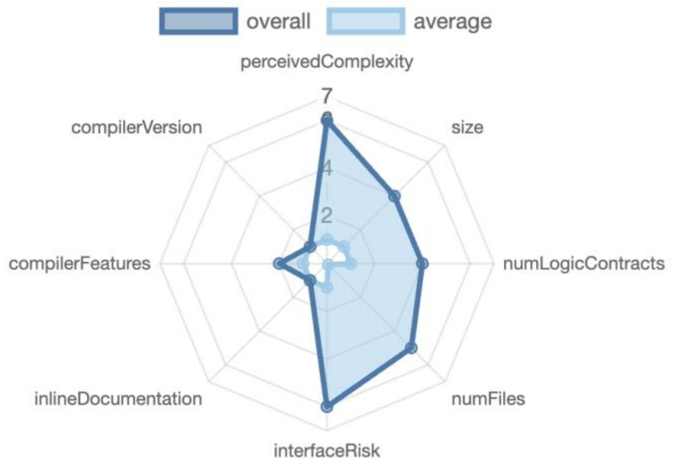

VaultCalculations
_calculateMintAmount
_getTokenAmountToMint
assetManagementConfig
_getMinDepositToVaultBalanceRatio
_getDepositToVaultBalanceRatio
getVaultValueInUSD
protocolConfig
Dependencies
assetManagementConfig
protocolConfig
feeModule
IPriceOracle
convertToUSD18Decimals
MathUtils (lib)
_min
CooldownManager
_calculateCooldownPeriod
_subOrZero
_checkCoolDownPeriod
_max
TokenBalanceLibrary
IERC20Upgradeable
getTokenBalancesOf
_getTokenBalanceOf
address
ErrorLibrary
MintedAmountIsNotAccepted
TokenNotEnabled
CoolDownPeriodNotPassed
DivisionByZero
TokenCalculations
BalanceOfVaultIsZero
_calculateMintAmount
TokenAlreadyExist
_getDepositToVaultBalanceRatio
InvalidAddress
TokenCountOutOfLimit
AlreadyInitialized
InvalidMintAmount
CallerNeedToMaintainMinTokenAmount
CallerNotHavingGivenPortfolioTokenAmount
ProtocolIsPaused
VaultConfig
UserNotAllowedToDeposit
updateTokenList
_resetPreviousTokenList
PortfolioTokenNotInitialized
initToken
InvalidTokenAddress
__VaultConfig_init
TokenNotWhitelisted
getTokens
IPortfolio
balanceOf
ChecksAndValidations
_verifyUserMintedAmount
_beforeWithdrawCheck
protocolConfig
_beforeDepositCheck
assetManagementConfig
_beforeInitCheck
address

5.3 Inheritance Graph



5.4 Source Lines & Risk





5.5 Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
0.8.17 ^0.8.17 ^0.8.0		yes		

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
		yes	yes		yes → NewContract:ERC1967Proxy → NewContract:AccessController

TryCatch	Σ Unchecked
	yes

Exposed Functions
This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

Public	Payable
249	7

External	Internal	Private	Pure	View
238	362	3	23	91

StateVariables

Total	🌐Public
123	83

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
24	4	16	23

5.6 Dependencies / External imports

Dependency / Import Path	Source
@chainlink/contracts/src/v0.8/Denominations.sol	https://github.com/smartcontractkit/chainlink/blob/develop/contracts/src/v0.8/Denominations.sol
@chainlink/contracts/src/v0.8/interfaces/AggregatorV2V3Interface.sol	https://github.com/smartcontractkit/chainlink/blob/develop/contracts/src/v0.8/interfaces/AggregatorV2V3Interface.sol
@gnosis.pm/safe-contracts/contracts/GnosisSafe.sol	https://github.com/safe-global/safe-smart-account/blob/v1.1.0/contracts/GnosisSafe.sol
@gnosis.pm/safe-contracts/contracts/proxies/GnosisSafeProxyFactory.sol	https://github.com/safe-global/safe-smart-account/blob/v1.1.0/contracts/proxies/ProxyFactory.sol
@gnosis.pm/zodiac/contracts/core/Module.sol	https://github.com/gnosisguild/zodiac/blob/master/contracts/core/Module.sol
@openzeppelin/contracts-upgradeable-4.9.6/access/Ownable2StepUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts
@openzeppelin/contracts-upgradeable-4.9.6/access/OwnableUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts
@openzeppelin/contracts-upgradeable-4.9.6/interfaces/IERC20Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts
@openzeppelin/contracts-upgradeable-4.9.6/proxy/utils/Initializable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts

Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable-4.9.6/proxy/utils/UUPSUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts
@openzeppelin/contracts-upgradeable-4.9.6/security/ReentrancyGuardUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts/security/ReentrancyGuardUpgradeable.sol
@openzeppelin/contracts-upgradeable-4.9.6/token/ERC20/ERC20Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts/token/ERC20/ERC20Upgradeable.sol
@openzeppelin/contracts-upgradeable-4.9.6/token/ERC20/IERC20Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts/token/ERC20/IERC20Upgradeable.sol
@openzeppelin/contracts-upgradeable-4.9.6/token/ERC20/extensions/IERC20MetadataUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts/token/ERC20/extensions/IERC20MetadataUpgradeable.sol
@openzeppelin/contracts-upgradeable-4.9.6/token/ERC20/utils/SafeERC20Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.9.6/contracts/token/ERC20/IERC20Upgradeable.sol
@openzeppelin/contracts/access/AccessControl.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.2/contracts/access/AccessControl.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.2/contracts/access/Ownable.sol

Dependency / Import Path	Source
@openzeppelin/contracts/proxy/Clones.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.2/contracts/proxy/Clones.sol
@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.2/contracts/proxy/ERC1967/ERC1967Proxy.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.2/contracts/token/ERC20/utils/SafeERC20.sol
@uniswap/lib/contracts/libraries/TransferHelper.sol	https://github.com/Uniswap/v3-core/blob/main/contracts/libraries/TransferHelper.sol

5.7 Source Unites in Scope

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/vault/IVelvetSafeModule.sol		1	15	5	3	3	7
contracts/vault/VelvetSafeModule.sol	1		50	47	24	18	18
contracts/FunctionParameters.sol	1		203	203	88	107	1
contracts/PortfolioFactory.sol	1		518	473	325	104	276
contracts/rebalance/IRebalancing.sol		1	48	26	9	21	13
contracts/rebalance/Rebalancing.sol	1		388	353	198	102	187

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/rebalance/RebalancingConfig.sol	1		150	141	88	33	78
contracts/bundle/DepositBatch.sol	1		109	104	68	25	66
contracts/bundle/WithdrawBatch.sol	1		83	75	42	24	28
contracts/oracle/PriceOracleAbstract.sol	1		178	161	89	53	60
contracts/oracle/PriceOracleL2.sol	1		64	61	29	24	16
contracts/oracle/IPriceOracle.sol		1	19	7	4	1	7
contracts/oracle/PriceOracle.sol	1		43	40	15	19	9
contracts/access/AccessController.sol	1		98	87	42	32	39
contracts/access/AccessRoles.sol	1		38	38	15	15	25
contracts/access/IAccessController.sol		1	24	12	4	5	9
contracts/library/ErrorLibrary.sol	1		160	160	77	80	1
contracts/library/GnosisDeployer.sol	1		67	65	55	2	25
contracts/front-end-helpers/PortfolioCalculations.sol	1		284	234	176	20	97
contracts/front-end-helpers/IUniswapV2Router02.sol		1	420	13	3	225	61
contracts/front-end-helpers/IPortfolioFactory.sol		1	65	26	14	29	11
contracts/handler/IIntentHandler.sol		1	25	21	3	17	3

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/handler/ExternalSwapHandler/EnsoHandler.sol	1		79	76	37	28	29
contracts/fee/FeeConfig.sol	1		138	129	70	42	45
contracts/handler/ExternalSwapHandler/EnsoHandlerBundled.sol	1		63	60	29	24	26
contracts/fee/FeeEvents.sol	1		28	28	23	1	1
contracts/fee/IFeeModule.sol		1	52	12	3	29	13
contracts/fee/FeeModule.sol	1		172	158	104	35	59
contracts/fee/FeeCalculations.sol	1		199	166	79	72	17
contracts/config/assetManagement/PortfolioSettings.sol	1		153	136	76	43	51
contracts/config/assetManagement/UserWhitelistManagement.sol	1		61	55	34	14	39
contracts/config/assetManagement/AssetManagementConfig.sol	1		110	96	63	15	33
contracts/config/assetManagement/TokenWhitelistManagement.sol	1		63	59	33	19	27
contracts/config/assetManagement/FeeManagement.sol	1		213	200	110	61	92

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/config/assetManagement/AssetManagerCheck.sol	1		40	37	12	22	7
contracts/config/assetManagement/IAssetManagementConfig.sol		1	112	15	4	71	33
contracts/config/assetManagement/TreasuryManagement.sol	1		51	47	19	21	17
contracts/config/protocol/TokenManagement.sol	1		76	74	34	30	36
contracts/config/protocol/ProtocolFeeManagement.sol	1		57	53	36	15	24
contracts/config/protocol/SolverManagement.sol	1		52	50	21	21	20
contracts/config/protocol/IProtocolConfig.sol		1	203	15	3	130	61
contracts/config/protocol/ProtocolConfig.sol	1		71	60	40	13	31
contracts/config/protocol/SystemSettings.sol	1		148	137	80	43	60
contracts/config/protocol/OwnableCheck.sol	1		32	24	8	19	4
contracts/config/protocol/ProtocolTreasuryManagement.sol	1		49	45	22	15	18
contracts/config/protocol/OracleManagement.sol	1		42	42	20	15	18
contracts/core/Portfolio.sol	1		91	77	41	25	31
contracts/core/calculations/TokenCalculations.sol	1		52	46	14	26	6

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/core/calculations/TokenBalanceLibrary.sol	1		50	44	17	28	26
contracts/core/calculations/MathUtils.sol	1		47	47	18	26	4
contracts/core/calculations/VaultCalculations.sol	1		88	76	43	27	29
contracts/core/user/UserManagement.sol	1		44	44	15	22	10
contracts/core/management/VaultManager.sol	1		513	444	230	166	135
contracts/core/management/TokenExclusionManager.sol	1		277	255	132	90	69
contracts/core/management/FeeManager.sol	1		38	38	14	20	12
contracts/core/checks/ChecksAndValidations.sol	1		98	91	53	33	41
contracts/core/interfaces/IPortfolio.sol		1	207	38	20	98	63
contracts/core/interfaces/IEIP712.sol		1	6	5	3	1	3
contracts/core/interfaces/IAllowanceTransfer.sol		1	187	108	58	74	19
contracts/core/interfaces/IMultiSend.sol		1	6	5	3	1	6
contracts/core/interfaces/IGnosisSafe.sol		1	6	5	3	1	3
contracts/core/interfaces/ITokenExclusionManager.sol		1	28	5	3	1	15
contracts/core/config/Dependencies.sol	1		40	21	6	23	7
contracts/core/config/VaultConfig.sol	1		133	128	76	36	60

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/core/cooldown/CooldownManager.sol	1		83	77	40	30	12
contracts/core/access/AccessModifiers.sol	1		75	70	33	30	21
contracts/core/token/PortfolioToken.sol	1		154	134	74	46	65
Totals	**51**	**16**	**7536**	**5884**	**3227**	**2561**	**2435**

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...

6. Scope of Work

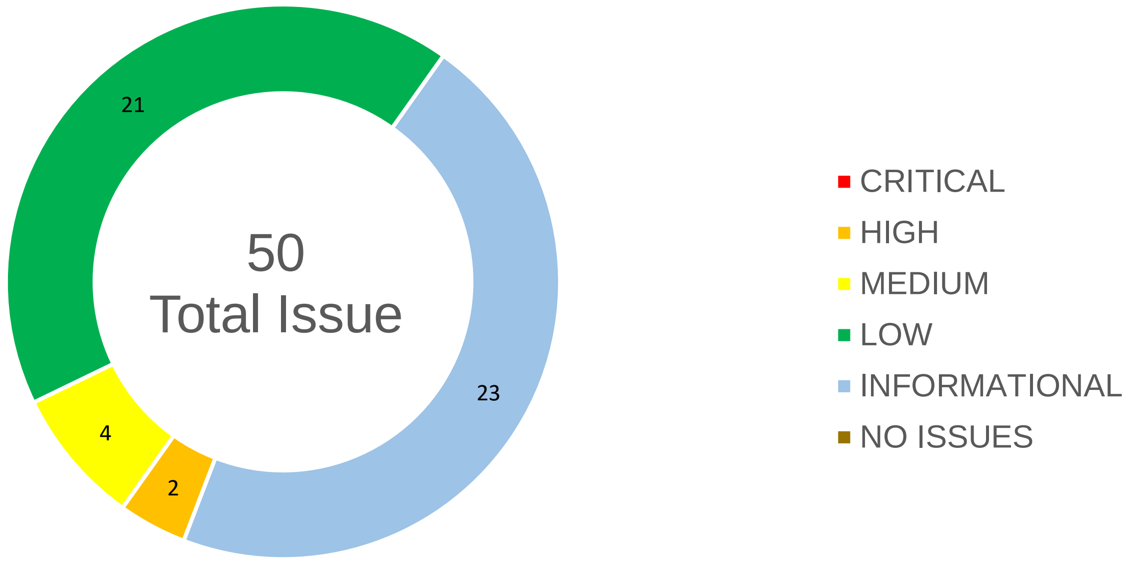
The Velvet Capital Team provided us with the files that needs to be tested. The scope of the audit are the core v3 contracts.

The team put forward the following assumptions regarding the security, usage of the contracts:

1. Compliance with Best Practices: The audit should ensure that the contracts adhere to smart contract best practices, including checking for common vulnerabilities such as reentrancy attacks and overflow/underflow issues.
2. Effective Role-Based Access Control: The audit confirms proper assignment and management of roles like `SUPER_ADMIN`, `ASSET_MANAGER`, and `WHITELIST_MANAGER`, ensuring only authorized entities execute privileged functions. The `AccessController` contract's role setup and management functions should be evaluated for correct implementation and security.
3. Secure Token Transfer Functions: Token transfer functions are audited to ensure resilience against common vulnerabilities, safeguarding against unauthorized transfers and balance manipulation.
4. Accurate Fee Calculation and Charging Mechanisms: The audit validates the accuracy and security of fee calculation and charging mechanisms in the `FeeModule`, `FeeCalculations`, and `VaultCalculations` contracts. This includes ensuring the correct implementation of management fees, performance fees, entry/exit fees, and their correct application during deposits and withdrawals.
5. Correct and Secure Initialization Processes: The audit verifies that the initialization processes in contracts are correctly implemented and securely executed.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client’s request.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Incorrect Parameter Order in Function Call	HIGH	FIXED
6.2.2	Incorrect updatedAt Value Retrieval	HIGH	FIXED
6.2.3	Division by Zero in PortfolioCalculations Contract	MEDIUM	FIXED
6.2.4	Incorrect Calculation Due to Mismatched Array Lengths in PortfolioCalculations Contract	MEDIUM	FIXED
6.2.5	Division Before Multiplication in FeeCalculations and PortfolioCalculations Contracts	MEDIUM	FIXED
6.2.6	Potential Issue with UUPSUpgradeable in _upgrade Function Due to Outdated OpenZeppelin Library	MEDIUM	FIXED
6.2.7	Division by Zero in PortfolioCalculations Contract	LOW	FIXED

6.2.8	Use Ownable2Step for Safer Ownership Transfer in VelvetSafeModule	LOW	FIXED
6.2.9	Potential Denial of Service (DoS) in whitelistUser Function Due to Lack of User Limit	LOW	FIXED
6.2.10	Missing Event Emission in PriceOracleAbstract.updateOracleExpirationThreshold Function	LOW	FIXED
6.2.11	Deprecated Library Function Usage in AccessController	LOW	FIXED
6.2.12	Incompatibility with Solidity 0.8.20 Due to PUSH0 Opcode	LOW	FIXED
6.2.13	Missing Storage Gap in TokenBalanceLibrary for Upgradeable Contracts	LOW	ACKNOWLEDGED
6.2.14	Non-compliance with Solidity Function Order in TokenBalanceLibrary	LOW	FIXED
6.2.15	Potential Division by Zero and Rounding Issues in calculateMintAmount and getDepositToVaultBalanceRatio Functions	LOW	FIXED
6.2.16	Inconsistent Handling of Deadlines in CooldownManager Contract	LOW	FIXED
6.2.17	Public Functions Not Used Internally Should Be Marked as External	LOW	FIXED
6.2.18	Events Missing indexed Fields in Multiple Contracts	LOW	FIXED
6.2.19	Empty Blocks in _authorizeUpgrade Functions	LOW	FIXED
6.2.20	Internal Functions Called Only Once Should Be Inlined	LOW	FIXED
6.2.21	Unused Custom Errors in Multiple Contracts	LOW	FIXED
6.2.22	Missing Event Emission in _updatePriceOracle Function OracleManagement contract	LOW	FIXED

6.2.23	Missing Check for Identical Protocol Fee in updateProtocolFee and updateProtocolStreamingFee Functions in ProtocolFeeManagement contract	LOW	FIXED
6.2.24	Missing Check for Identical Treasury Address in updateVelvetTreasury Function in ProtocolTreasuryManagement contract	LOW	FIXED
6.2.25	Missing Minimum Cooldown Period Check in setCoolDownPeriod Function	LOW	FIXED
6.2.26	Prevent Division by Zero in getDepositToVaultBalanceRatio function	LOW	FIXED
6.2.27	Missing Cooldown Period Check for Deposit	LOW	ACKNOWLEDGED
6.2.28	Missing inheritance	INFORMATIONAL	FIXED
6.2.29	Too Many Digits	INFORMATIONAL	FIXED
6.2.30	Unused Imports	INFORMATIONAL	FIXED
6.2.31	Non-compliance with Solidity Style Guide in AssetManagerCheck and OwnableCheck Contracts	INFORMATIONAL	FIXED
6.2.32	Missing NatSpec Comments in enableSolverHandler Function	INFORMATIONAL	FIXED
6.2.33	Non-compliance with Solidity Naming Convention in MathUtils Library	INFORMATIONAL	FIXED
6.2.34	Redundant Return Statement in getTokenBalancesOf Function	INFORMATIONAL	FIXED
6.2.35	Use Enums for Array Indices Instead of Numeric Literals in PortfolioCalculations Contract	INFORMATIONAL	ACKNOWLEDGED
6.2.36	Potential Issues with abi.encodeWithSignature in GnosisDeployer Contract	INFORMATIONAL	FIXED
6.2.37	Use bytes.concat Instead of abi.encodePacked in GnosisDeployer Contract	INFORMATIONAL	FIXED

6.2.38	Superfluous Initialization of Loop Variable in getTokenBalancesOf Function	INFORMATIONAL	FIXED
6.2.39	Optimization Opportunity in _disableInitializers Function	INFORMATIONAL	FIXED
6.2.40	Typographical Error in Smart Contract File Name PortolfioCalculations.sol	INFORMATIONAL	FIXED
6.2.41	Unnecessary Variable Usage in _updateWeights Function	INFORMATIONAL	FIXED
6.2.42	Missing Function Parameters in NatSpec Documentation	INFORMATIONAL	FIXED
6.2.43	Missing NatSpec Documentation for Interfaces	INFORMATIONAL	FIXED
6.2.44	Incorrect Documentation	INFORMATIONAL	FIXED
6.2.45	Inconsistent Implementation of updateTokens Function in IRebalancing and Rebalancing Contracts	INFORMATIONAL	FIXED
6.2.46	Duplicate NatSpec Documentation for getUserAmountToDeposit Function	INFORMATIONAL	FIXED
6.2.47	Gas Optimization for setEmergencyPause Function	INFORMATIONAL	FIXED
6.2.48	Use of Constant ONE_ETH_IN_WEI for Gas Efficiency	INFORMATIONAL	FIXED
6.2.49	Unused Named Returns	INFORMATIONAL	FIXED
6.2.50	Assignment of tokensLength Variable Earlier in initToken Function	INFORMATIONAL	FIXED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, softstack‘s experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, softstack‘s experts found **2 High issues** in the code of the smart contract.

6.2.1 Incorrect Parameter Order in Function Call

Severity: HIGH
Status: FIXED
File(s) affected: FeeModule.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/224

Attack / Description	In the call to the calculateProtocolAndManagementFeesToMint function, the parameters lastChargedProtocolFee and lastChargedManagementFee are swapped. This can lead to incorrect fee calculations, impacting the contract's financial operations.

Code	Line 87 - 97 (FeeModule.sol):
	`(`
	`   uint256 assetManagerFeeToMint,`
	`   uint256 protocolFeeToMint`
	` ) = _calculateProtocolAndManagementFeesToMint(`
	`   _managementFee,`
	`   _protocolFee,`
	`   _protocolStreamingFee,`
	`   _totalSupply,`

	```
	    lastChargedManagementFee,
	    lastChargedProtocolFee,
	    block.timestamp
	  );
	```

Result/Recommendation	Swap the lastChargedProtocolFee and lastChargedManagementFee parameters to match the expected order in the function definition.
	```
	(
	  uint256 assetManagerFeeToMint,
	  uint256 protocolFeeToMint
	) = calculateProtocolAndManagementFeesToMint(
	    _managementFee,
	    _protocolFee,
	    _protocolStreamingFee,
	    _totalSupply,
	    lastChargedManagementFee, // Correct order
	    lastChargedProtocolFee, // Correct order
	    block.timestamp
	  );
	```

6.2.2 Incorrect updatedAt Value Retrieval

Severity: HIGH
Status: FIXED
File(s) affected: PriceOracleL2.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/224

Attack / Description	In the latestRoundData function where the latest price data for a token pair is retrieved, the updatedAt value is expected to be the second to last value returned by the latestRoundData

	function from the AggregatorV2V3Interface. The current code incorrectly retrieves updatedAt as the last value.

Code	Line 51 - 54 (PriceOracleL2.sol):
	`// Retrieve the latest price data for the given token pair`
	`(, int256 price, , uint256 updatedAt, ) = tokenPairToAggregator[base]`
	`  .aggregators[quote]`
	`  .latestRoundData();`

Result/Recommendation	Adjust the code to correctly retrieve the updatedAt value as the second to last value returned by the latestRoundData function.
	`// Retrieve the latest price data for the given token pair`
	`(, int256 price, , uint256 updatedAt, ) = tokenPairToAggregator[base]`
	`  .aggregators[quote]`
	`  .latestRoundData();`

MEDIUM ISSUES

During the audit, softstack‘s experts found 4 Medium issues in the code of the smart contract.

6.2.3 Division by Zero in PortfolioCalculations Contract

Severity: MEDIUM
Status: FIXED
File(s) affected: PortfolioCalculations.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The PortolfioCalculations contract contains a function getUserAmountToDeposit that calculates the lowest amount possible to deposit to get the exact ratio of token amounts in a portfolio. However, if

	any of the vaultBalance values are zero, the division operations in the function will result in a division by zero error, causing the transaction to revert. **Impact** Division by zero errors can disrupt the functionality of the contract and prevent users from obtaining the correct deposit amounts. This can lead to a poor user experience and potential financial losses if users are unable to interact with the contract as intended.
Code	Line 22 - 63 (PortfolioCalculations.sol):

```
function getUserAmountToDeposit(
    uint256[] memory userAmounts,
    address _portfolio
  ) external view returns (uint256[] memory, uint256 _desiredShare) {
    IPortfolio portfolio = IPortfolio(_portfolio);

    uint256[] memory vaultBalance = portfolio.getTokenBalancesOf(
      portfolio.getTokens(),
      portfolio.vault()
    );
    uint256 vaultTokenLength = vaultBalance.length;

    // Validate that none of the vault balances are zero
    for (uint256 i = 0; i < vaultTokenLength; i++) {
      if (vaultBalance[i] == 0) revert ErrorLibrary.BalanceOfVaultIsZero();
    }

    // Validate that the lengths of the input arrays match
```

	```
	  if (userAmounts.length != vaultTokenLength)
	    revert ErrorLibrary.InvalidLength();

	  uint256[] memory newAmounts = new uint256[](vaultTokenLength);
	  uint256 leastPercentage = (userAmounts[0] * ONE_ETH_IN_WEI) /
	    vaultBalance[0];
	  _desiredShare =
	    (userAmounts[0] * ONE_ETH_IN_WEI) /
	    (vaultBalance[0] + userAmounts[0]);
	  for (uint256 i = 1; i < vaultTokenLength; i++) {
	    uint256 tempPercentage = (userAmounts[i] * ONE_ETH_IN_WEI) /
	      vaultBalance[i];
	    if (leastPercentage > tempPercentage) {
	      leastPercentage = tempPercentage;
	      _desiredShare =
	        (userAmounts[i] * ONE_ETH_IN_WEI) /
	        (vaultBalance[i] + userAmounts[i]);
	    }
	  }
	  for (uint256 i; i < vaultTokenLength; i++) {
	    newAmounts[i] = (vaultBalance[i] * leastPercentage) / ONE_ETH_IN_WEI;
	  }
	  return (newAmounts, _desiredShare);
	}
	```

Result/Recommendation	Implement checks to ensure that none of the values in the vaultBalance array are zero before performing division operations. This will prevent division by zero errors and ensure the function operates correctly.
	
	`function getUserAmountToDeposit(`
	`    uint256[] memory userAmounts`
	`) external view returns (uint256[] memory, uint256 _desiredShare) {`
	`    uint256[] memory vaultBalance = portfolio.getTokenBalancesOf(`
	`      portfolio.getTokens(),`
	`      portfolio.vault()`
	`    );`
	`    uint256 vaultTokenLength = vaultBalance.length;`
	
	`    // Validate that none of the vault balances are zero`
	`    for (uint256 i = 0; i < vaultTokenLength; i++) {`
	`        require(vaultBalance[i] != 0, "Vault balance cannot be zero");`
	`    }`
	
	`    uint256[] memory newAmounts = new uint256[](vaultTokenLength);`
	`    uint256 leastPercentage = (userAmounts[0] * ONE_ETH_IN_WEI) /`
	`      vaultBalance[0];`
	`    _desiredShare =`
	`      (userAmounts[0] * ONE_ETH_IN_WEI) /`
	`      (vaultBalance[0] + userAmounts[0]);`
	`    for (uint256 i = 1; i < vaultTokenLength; i++) {`
	`      uint256 tempPercentage = (userAmounts[i] * ONE_ETH_IN_WEI) /`
	`        vaultBalance[i];`
	`      if (leastPercentage > tempPercentage) {`
	`        leastPercentage = tempPercentage;`
	`        _desiredShare =`
	`          (userAmounts[i] * ONE_ETH_IN_WEI) /`
	`          (vaultBalance[i] + userAmounts[i]);`

	```
	    }
	  }
	  for (uint256 i = 0; i < vaultTokenLength; i++) {
	    newAmounts[i] = (vaultBalance[i] * leastPercentage) / ONE_ETH_IN_WEI;
	  }
	  return (newAmounts, _desiredShare);
	}
	```

6.2.4 Incorrect Calculation Due to Mismatched Array Lengths in PortfolioCalculations Contract

Severity: MEDIUM
Status: FIXED
File(s) affected: PortfolioCalculations.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The PortfolioCalculations contract contains a function getUserAmountToDeposit that calculates the lowest amount possible to deposit to get the exact ratio of token amounts in a portfolio. The function assumes that the userAmounts array and the vaultBalance array have the same length and corresponding indices. If this assumption is violated, the calculations will be incorrect. Impact If the lengths of the userAmounts array and the vaultBalance array do not match, the function will produce incorrect results. This can lead to users receiving incorrect deposit amounts, which can cause financial losses and disrupt the intended functionality of the contract.

Code	Line 22 - 62 (PriceOracleL2.sol):
	```
	function getUserAmountToDeposit(
	  uint256[] memory userAmounts,
	  address _portfolio
	```

```
) external view returns (uint256[] memory, uint256 _desiredShare) {
  IPortfolio portfolio = IPortfolio(_portfolio);

  uint256[] memory vaultBalance = portfolio.getTokenBalancesOf(
    portfolio.getTokens(),
    portfolio.vault()
  );
  uint256 vaultTokenLength = vaultBalance.length;

  // Validate that none of the vault balances are zero
  for (uint256 i = 0; i < vaultTokenLength; i++) {
    if (vaultBalance[i] == 0) revert ErrorLibrary.BalanceOfVaultIsZero();
  }

  // Validate that the lengths of the input arrays match
  if (userAmounts.length != vaultTokenLength)
    revert ErrorLibrary.InvalidLength();

  uint256[] memory newAmounts = new uint256[](vaultTokenLength);
  uint256 leastPercentage = (userAmounts[0] * ONE_ETH_IN_WEI) /
    vaultBalance[0];
  _desiredShare =
    (userAmounts[0] * ONE_ETH_IN_WEI) /
    (vaultBalance[0] + userAmounts[0]);
  for (uint256 i = 1; i < vaultTokenLength; i++) {
```

	```
	    uint256 tempPercentage = (userAmounts[i] * ONE_ETH_IN_WEI) /
	     vaultBalance[i];
	    if (leastPercentage > tempPercentage) {
	     leastPercentage = tempPercentage;
	     _desiredShare =
	      (userAmounts[i] * ONE_ETH_IN_WEI) /
	      (vaultBalance[i] + userAmounts[i]);
	    }
	   }
	   for (uint256 i; i < vaultTokenLength; i++) {
	    newAmounts[i] = (vaultBalance[i] * leastPercentage) / ONE_ETH_IN_WEI;
	   }
	   return (newAmounts, _desiredShare);
	  }
	```

Result/Recommendation	Add input validation to ensure that the userAmounts array and the vaultBalance array have the same length. This will prevent incorrect calculations and ensure the function operates as intended.
	```
	function getUserAmountToDeposit(
	    uint256[] memory userAmounts
	) external view returns (uint256[] memory, uint256 _desiredShare) {
	    uint256[] memory vaultBalance = portfolio.getTokenBalancesOf(
	      portfolio.getTokens(),
	      portfolio.vault()
	    );
	    uint256 vaultTokenLength = vaultBalance.length;

	    // Validate that the lengths of the input arrays match
	```

```
  require(userAmounts.length == vaultTokenLength, "Input array lengths do not match");

  uint256[] memory newAmounts = new uint256[](vaultTokenLength);
  uint256 leastPercentage = (userAmounts[0] * ONE_ETH_IN_WEI) /
    vaultBalance[0];
  _desiredShare =
    (userAmounts[0] * ONE_ETH_IN_WEI) /
    (vaultBalance[0] + userAmounts[0]);
  for (uint256 i = 1; i < vaultTokenLength; i++) {
    uint256 tempPercentage = (userAmounts[i] * ONE_ETH_IN_WEI) /
      vaultBalance[i];
    if (leastPercentage > tempPercentage) {
      leastPercentage = tempPercentage;
      _desiredShare =
        (userAmounts[i] * ONE_ETH_IN_WEI) /
        (vaultBalance[i] + userAmounts[i]);
    }
  }
  for (uint256 i = 0; i < vaultTokenLength; i++) {
    newAmounts[i] = (vaultBalance[i] * leastPercentage) / ONE_ETH_IN_WEI;
  }
  return (newAmounts, _desiredShare);
}
```

6.2.5 Division Before Multiplication in FeeCalculations and PortfolioCalculations Contracts

Severity: MEDIUM
Status: FIXED
File(s) affected: FeeCalculations.sol, PortfolioCalculations.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The FeeCalculations and PortolfioCalculations contracts contain functions that perform a multiplication on the result of a division. This can lead to precision loss and incorrect calculations due to the inherent limitations of integer arithmetic in Solidity. Precision Loss Performing division before multiplication can lead to significant precision loss, especially when dealing with large numbers or numbers with many decimal places. This can result in incorrect fee calculations and user deposit amounts. Incorrect Calculations: The precision loss can cause the functions to return incorrect values, which can affect the overall functionality and reliability of the platform. For example, users may receive incorrect amounts of tokens, and fees may be miscalculated.

Code	Line 190 - 195 (FeeCalculations.sol):
	
	`uint256 performanceIncrease = _currentPrice - _highWaterMark;`
	`    uint256 performanceFee = (performanceIncrease *`
	`      _totalSupply *`
	`      _feePercentage) /`
	`      TOTAL_WEIGHT /`
	`      ONE_ETH_IN_WEI;`
	
	Line 36 - 37 (PortfolioCalculations.sol):
	
	`uint256 leastPercentage = (userAmounts[0] * ONE_ETH_IN_WEI) /`
	`      vaultBalance[0];`
	
	Line 52 (PortfolioCalculations.sol):
	
	`newAmounts[i] = (vaultBalance[i] * leastPercentage) / ONE_ETH_IN_WEI;`

Result/Recommendation	Reorder Operations Reorder the operations to perform multiplication before division to minimize precision loss. This ensures that the calculations are as accurate as possible. Use SafeMath Library Consider using a safe math library, such as OpenZeppelin's SafeMath, to handle arithmetic operations and prevent overflow/underflow issues.

6.2.6 Potential Issue with UUPSUpgradeable in _upgrade Function Due to Outdated OpenZeppelin Library

Severity: MEDIUM
Status: FIXED
File(s) affected: PortfolioFactory.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/224

Attack / Description	The _upgrade function in the PortfolioFactory contract uses the UUPSUpgradeable contract to upgrade proxies. However, the current implementation may encounter issues due to the visibility of the upgradeTo and upgradeToAndCall functions in the UUPSUpgradeable contract from an older version of the OpenZeppelin library. These functions were previously marked as external, which can cause problems when invoked using the super keyword in derived contracts. This issue has been addressed in a newer version of the OpenZeppelin library where these functions are now marked as public.

Code	Line 390 - 406 (PortfolioFactory.sol):
	
	`function _upgrade(`
	`    address[] calldata _proxy,`
	`    address _newImpl`
	`  ) internal virtual onlyOwner {`

	```
	  if (!IProtocolConfig(protocolConfig).isProtocolPaused()) {
	    revert ErrorLibrary.ProtocolNotPaused();
	  }
	  if (_newImpl == address(0)) {
	    revert ErrorLibrary.InvalidAddress();
	  }
	  uint256 proxyLength = _proxy.length;
	  for (uint256 i; i < proxyLength; i++) {
	    address proxyAddress = _proxy[i];
	    if (proxyAddress == address(0)) revert ErrorLibrary.InvalidAddress();
	    UUPSUpgradeable(_proxy[i]).upgradeTo(_newImpl);
	  }
	}
	```
Result/Recommendation	Impose a maximum limit on the number of users that can be whitelisted in a single transaction. This can be achieved by adding a require statement to check the length of the users array against a predefined maximum limit.

LOW ISSUES

During the audit, softstack‘s experts found **21 Low issues** in the code of the smart contract

6.2.7 Division by Zero in PortfolioCalculations Contract

Severity: LOW
Status: FIXED

File(s) affected: PortfolioCalculations.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The PortolfioCalculations contract contains a function getUserAmountToDeposit that calculates the lowest amount possible to deposit to get the exact ratio of token amounts in a portfolio. However, if any of the vaultBalance values are zero, the division operations in the function will result in a division by zero error, causing the transaction to revert. **Impact** Division by zero errors can disrupt the functionality of the contract and prevent users from obtaining the correct deposit amounts. This can lead to a poor user experience and potential financial losses if users are unable to interact with the contract as intended.

Code	Line 22 - 63 (PortfolioCalculations.sol):
	
	`function getUserAmountToDeposit(`
	`    uint256[] memory userAmounts,`
	`    address _portfolio`
	`  ) external view returns (uint256[] memory, uint256 _desiredShare) {`
	`    IPortfolio portfolio = IPortfolio(_portfolio);`
	
	`    uint256[] memory vaultBalance = portfolio.getTokenBalancesOf(`
	`      portfolio.getTokens(),`
	`      portfolio.vault()`
	`    );`
	`    uint256 vaultTokenLength = vaultBalance.length;`
	
	`    // Validate that none of the vault balances are zero`
	`    for (uint256 i = 0; i < vaultTokenLength; i++) {`

```
  if (vaultBalance[i] == 0) revert ErrorLibrary.BalanceOfVaultIsZero();
}

// Validate that the lengths of the input arrays match
if (userAmounts.length != vaultTokenLength)
  revert ErrorLibrary.InvalidLength();

uint256[] memory newAmounts = new uint256[](vaultTokenLength);
uint256 leastPercentage = (userAmounts[0] * ONE_ETH_IN_WEI) /
  vaultBalance[0];
_desiredShare =
  (userAmounts[0] * ONE_ETH_IN_WEI) /
  (vaultBalance[0] + userAmounts[0]);
for (uint256 i = 1; i < vaultTokenLength; i++) {
  uint256 tempPercentage = (userAmounts[i] * ONE_ETH_IN_WEI) /
    vaultBalance[i];
  if (leastPercentage > tempPercentage) {
    leastPercentage = tempPercentage;
    _desiredShare =
      (userAmounts[i] * ONE_ETH_IN_WEI) /
      (vaultBalance[i] + userAmounts[i]);
  }
}
for (uint256 i; i < vaultTokenLength; i++) {
  newAmounts[i] = (vaultBalance[i] * leastPercentage) / ONE_ETH_IN_WEI;
```

	```
	  }
	  return (newAmounts, _desiredShare);
	 }
	```

Result/Recommendation	Implement checks to ensure that none of the values in the vaultBalance array are zero before performing division operations. This will prevent division by zero errors and ensure the function operates correctly.
	```
	function getUserAmountToDeposit(
	    uint256[] memory userAmounts
	) external view returns (uint256[] memory, uint256 _desiredShare) {
	    uint256[] memory vaultBalance = portfolio.getTokenBalancesOf(
	      portfolio.getTokens(),
	      portfolio.vault()
	    );
	    uint256 vaultTokenLength = vaultBalance.length;

	    // Validate that none of the vault balances are zero
	    for (uint256 i = 0; i < vaultTokenLength; i++) {
	        require(vaultBalance[i] != 0, "Vault balance cannot be zero");
	    }

	    uint256[] memory newAmounts = new uint256[](vaultTokenLength);
	    uint256 leastPercentage = (userAmounts[0] * ONE_ETH_IN_WEI) /
	      vaultBalance[0];
	    _desiredShare =
	      (userAmounts[0] * ONE_ETH_IN_WEI) /
	      (vaultBalance[0] + userAmounts[0]);
	    for (uint256 i = 1; i < vaultTokenLength; i++) {
	      uint256 tempPercentage = (userAmounts[i] * ONE_ETH_IN_WEI) /
	        vaultBalance[i];
	```

	```
	    if (leastPercentage > tempPercentage) {
	      leastPercentage = tempPercentage;
	      _desiredShare =
	        (userAmounts[i] * ONE_ETH_IN_WEI) /
	        (vaultBalance[i] + userAmounts[i]);
	    }
	  }
	  for (uint256 i = 0; i < vaultTokenLength; i++) {
	    newAmounts[i] = (vaultBalance[i] * leastPercentage) / ONE_ETH_IN_WEI;
	  }
	  return (newAmounts, _desiredShare);
	}
	```

6.2.8 Use Ownable2Step for Safer Ownership Transfer in VelvetSafeModule

Severity: LOW
Status: FIXED
File(s) affected: VelvetSafeModule.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/161

Attack / Description	The VelvetSafeModule contract currently uses the transferOwnership function from the Ownable contract, which transfers ownership in a single step. This approach is less secure compared to the two-step ownership transfer provided by Ownable2Step. The two-step process ensures that the new owner explicitly accepts ownership, reducing the risk of accidental or malicious ownership transfers.

Code	Line 31 (VelvetSafeModule.sol):
	
	`transferOwnership(_portfolio);`

Result/Recommendation	Replace the Ownable contract with Ownable2Step and use the transferOwnership and acceptOwnership functions provided by Ownable2Step for a safer ownership transfer process. https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol

6.2.9 Potential Denial of Service (DoS) in whitelistUser Function Due to Lack of User Limit
Severity: LOW
Status: FIXED
File(s) affected: UserWhitelistManagement.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/224

Attack / Description	The whitelistUser function in the UserWhitelistManagement contract allows an arbitrary number of users to be whitelisted in a single transaction. This can lead to a potential Denial of Service (DoS) attack due to high gas consumption when processing a large array of users. To mitigate this risk, it is recommended to impose a maximum limit on the number of users that can be whitelisted in a single transaction.

Code	Line 24 (VelvetSafeModule.sol):
	
	`uint256 len = users.length;`

Result/Recommendation	Impose a maximum limit on the number of users that can be whitelisted in a single transaction. This can be achieved by adding a require statement to check the length of the users array against a predefined maximum limit.

6.2.10 Missing Event Emission in PriceOracleAbstract.updateOracleExpirationThreshold Function

Severity: LOW
Status: FIXED
File(s) affected: PriceOracleAbstract.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/224

Attack / Description	The PriceOracleAbstract contract contains a function updateOracleExpirationThreshold that updates the oracleExpirationThreshold state variable. However, this function does not emit an event when the state variable is updated. Emitting events for state changes is a best practice in Solidity to ensure transparency and traceability of state changes.

Code	Line 125 - 130 (PriceOracleAbstract.sol):
	
	`function updateOracleExpirationThreshold(`
	`    uint256 _newTimeout`
	`  ) external onlyOwner {`
	`    oracleExpirationThreshold = _newTimeout;`
	`    emit OracleExpirationThresholdUpdated(oracleExpirationThreshold);`
	`  }`

Result/Recommendation	Emit an Event Add an event emission in the updateOracleExpirationThreshold function to log the change in the oracleExpirationThreshold state variable. This will enhance transparency, auditability, and monitoring capabilities.

6.2.11 Deprecated Library Function Usage in AccessController

Severity: LOW
Status: FIXED
File(s) affected: AccessController.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/160

Attack / Description	The AccessController contract uses the deprecated _setupRole function from the OpenZeppelin AccessControl library.
Code	The following instances of _setupRole usage have been identified: contracts/access/AccessController.sol#20 contracts/access/AccessController.sol#39 contracts/access/AccessController.sol#49 contracts/access/AccessController.sol#51 contracts/access/AccessController.sol#58 contracts/access/AccessController.sol#59 contracts/access/AccessController.sol#60 contracts/access/AccessController.sol#61 contracts/access/AccessController.sol#63 contracts/access/AccessController.sol#64 contracts/access/AccessController.sol#66 contracts/access/AccessController.sol#78 The _setupRole function has been marked as deprecated in favor of _grantRole as per the OpenZeppelin pull request #2568. https://github.com/OpenZeppelin/openzeppelin-contracts/pull/2568
Result/Recommendation	Replace all instances of _setupRole with _grantRole to ensure compatibility with future versions of the OpenZeppelin library and to adhere to best practices.

6.2.12 Incompatibility with Solidity 0.8.20 Due to PUSH0 Opcode

Severity: LOW
Status: FIXED
File(s) affected: N/A
Update: https://github.com/Velvet-Capital/v3-contract/pull/161

Attack / Description	The contracts are compiled with Solidity version 0.8.23, which defaults to targeting the Shanghai EVM version. This includes the new PUSH0 opcode introduced in Solidity 0.8.20. The PUSH0

	opcode is not yet supported on all Layer 2 (L2) solutions, such as Arbitrum, leading to deployment failures on these chains.
Code	N/A
Result/Recommendation	To ensure compatibility across all intended chains, including those that do not yet support the PUSH0 opcode, it is recommended to either: 1. Downgrade the Solidity Compiler Version: Use Solidity version 0.8.19 or earlier, which does not include the PUSH0 opcode. 2. Specify an Earlier EVM Version: Configure the compiler to target an earlier EVM version, such as paris, which does not include the PUSH0 opcode.

6.2.13 Missing Storage Gap in TokenBalanceLibrary for Upgradeable Contracts

Severity: LOW
Status: ACKNOWLEDGED
File(s) affected: TokenBalanceLibrary.sol
Update: We intentionally did not add storage gaps to this contract because we believe it is unnecessary. This contract functions are primarily for calculations, similar to a library.

Attack / Description	The TokenBalanceLibrary contract does not include a storage gap, which is essential for maintaining storage compatibility in upgradeable contracts. According to the OpenZeppelin documentation, a storage gap should be included to allow for future state variable additions without compromising the storage layout. This is crucial for ensuring that the contract can be safely extended or integrated with other projects that may use upgradeable patterns.
Code	N/A

Result/Recommendation	Add a storage gap to the TokenBalanceLibrary contract to ensure future compatibility with upgradeable patterns. This can be done by including an empty reserved space in storage, typically an array of 50 slots.

6.2.14 Non-compliance with Solidity Function Order in TokenBalanceLibrary

Severity: LOW
Status: FIXED
File(s) affected: TokenBalanceLibrary.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/225

Attack / Description	The TokenBalanceLibrary contract does not follow the recommended function order as specified in the Solidity style guide. According to the style guide, functions should be laid out in the following order: constructor, receive, fallback, external, public, internal, and private. The current implementation of the TokenBalanceLibrary contract does not adhere to this order, which can lead to confusion and reduce code readability.
Code	N/A
Result/Recommendation	Reorder the functions in the TokenBalanceLibrary contract to comply with the Solidity style guide. This will improve code readability and maintainability by providing a consistent structure for developers.

6.2.15 Potential Division by Zero and Rounding Issues in calculateMintAmount and getDepositToVaultBalanceRatio Functions

Severity: LOW
Status: FIXED
File(s) affected: TokenCalculations.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The calculateMintAmount and getDepositToVaultBalanceRatio functions in the contract perform divisions without checking for zero values in the input parameters. This can lead to the functions reverting when zero is passed as an argument. Additionally, there are potential rounding issues due to Solidity not supporting fractions, which may result in the division yielding zero when dealing with large numbers. The use of magic numbers in the division operations also reduces code readability and maintainability.

Code	Line 26 - 31 (TokenCalculations.sol):
	
	`function _calculateMintAmount(`
	`    uint256 _userShare,`
	`    uint256 _totalSupply`
	`  ) internal pure returns (uint256) {`
	`    uint256 remainingShare = ONE_ETH_IN_WEI - _userShare;`
	`    if (remainingShare == 0) revert ErrorLibrary.DivisionByZero();`
	`    return (_userShare * _totalSupply) / remainingShare;`
	`  }`
	
	Line 43 - 51 (TokenCalculations.sol):
	
	`function _getDepositToVaultBalanceRatio(`
	`    uint256 depositAmount,`
	`    uint256 tokenBalance`
	`  ) internal pure returns (uint256) {`
	`    if (tokenBalance == 0) revert ErrorLibrary.BalanceOfVaultIsZero();`
	
	`    // Calculate the deposit ratio to 18 decimal precision`

	```
	  return (depositAmount * ONE_ETH_IN_WEI) / tokenBalance;
	}
	```
Result/Recommendation	1. Add Zero-Value Checks: Ensure that the input parameters are checked for zero values before performing the division to prevent the functions from reverting. 2. Use Constants for Magic Numbers: Define constants for magic numbers to improve code readability and maintainability. 3. Handle Rounding Issues: Consider requiring a minimum amount for the numerator to ensure it is always larger than the denominator or use a different approach to handle potential rounding issues.

6.2.16 Inconsistent Handling of Deadlines in CooldownManager Contract

Severity: LOW
Status: FIXED
File(s) affected: CooldownManager.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The CooldownManager contract uses a timestamp comparison to enforce cooldown periods. According to EIP-2612, signatures used on exactly the deadline timestamp should be allowed. While the signature may or may not be used for the exact EIP-2612 use case (transfer approvals), for consistency's sake, all deadlines should follow this semantic. The current implementation does not allow actions at the exact timestamp of the deadline, which may lead to unexpected reverts. The comparison userCoolDownPeriod < block.timestamp does not allow actions at the exact timestamp of the deadline, which is inconsistent with the behavior recommended by EIP-2612.

Code	Line 72 - 82 (CooldownManager.sol):
	```
	function _checkCoolDownPeriod(address _user) internal view {
	  uint256 userCoolDownPeriod = userLastDepositTime[_user] +
	    userCooldownPeriod[_user];
	```

	uint256 remainingCoolDown = userCoolDownPeriod <= block.timestamp
	? 0
	: userCoolDownPeriod - block.timestamp;
	
	if (remainingCoolDown > 0) {
	revert ErrorLibrary.CoolDownPeriodNotPassed();
	}
	}

Result/Recommendation	Modify the comparison to allow actions at the exact timestamp of the deadline. This can be achieved by changing the comparison from < to <=.

6.2.17 Public Functions Not Used Internally Should Be Marked as External

Severity: LOW
Status: FIXED
File(s) affected: Portfolio.sol, VaultConfig.sol, UniSwapV2Handler.sol, FeeModuleV3_2.sol, PortfolioV3_2.sol, TokenExclusionManagerV3_2.sol, VaultConfigV3_4.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	In the codebase, several functions are marked as public but are not used internally within their respective contracts. According to best practices, functions that are not called internally should be marked as external instead of public. This change can lead to gas savings and better clarity in the contract's interface.
Code	The following instances have been identified where public functions can be marked as external: contracts/core/Portfolio.sol Line 57: function assetManagementConfig()

	Line 67: function protocolConfig() Line 77: function feeModule() public view override(Dependencies) returns (IFeeModule) contracts/core/config/VaultConfig.sol Line 92: function updateTokenList() contracts/mock/UniSwapV2Handler.sol Line 41: function getSwapAddress() public view returns (address) Line 45: function swapTokensToETH() Line 65: function swapTokenToTokens() Line 97: function swapETHToTokens() contracts/mock/upgradeability/FeeModuleV3_2.sol Line 27: function init() contracts/mock/upgradeability/PortfolioV3_2.sol Line 69: function assetManagementConfig() Line 79: function protocolConfig() Line 89: function feeModule() public view override(Dependencies) returns (IFeeModule) contracts/mock/upgradeability/TokenExclusionManagerV3_2.sol Line 108: function snapshot() public onlyPortfolioManager returns (uint256) Line 120: function claimRemovedTokens(address user) public nonReentrant Line 201: function setUserRecord() Line 226: function setTokenAndSupplyRecord() Line 263: function _incrementedSnapshot() public contracts/mock/upgradeability/changedDependencies/v3_4/VaultConfigV3_4.sol Line 86: function updateTokenList() contracts/rebalance/Rebalancing.sol Line 153: function removePortfolioToken() Line 183: function removePortfolioTokenPartially()

Result/Recommendation	Change the visibility of the identified functions from public to external to adhere to best practices and potentially save on gas costs.

6.2.18 Events Missing indexed Fields in Multiple Contracts

Severity: LOW
Status: FIXED
File(s) affected: PortfolioFactory.sol, FeeManagement.sol, PortfolioSettings.sol, TokenWhitelistManagement.sol, TreasuryManagement.sol, UserWhitelistManagement.sol, ProtocolFeeManagement.sol, ProtocolTreasuryManagement.sol, SolverManagement.sol, SystemSettings.sol, TokenManagement.sol, VaultConfig.sol, IPortfolio.sol, TokenExclusionManager.sol, FeeEvents.sol, TokenExclusionManagerV3_2.sol, VaultConfigV3_4.sol, PriceOracleAbstract.sol, Rebalancing.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/224

Attack / Description	Several events across multiple contracts are missing indexed fields. Indexing event fields makes them more quickly accessible to off-chain tools that parse events. Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.
Code	The following instances have been identified where events are missing indexed fields: contracts/PortfolioFactory.sol Line 57: event PortfolioInfo Line 62: event PortfolioCreationState(bool state) Line 63: event UpgradePortfolio(address newImplementation) Line 64: event UpgradeAssetManagerConfig(address newImplementation) Line 65: event UpgradeFeeModule(address newImplementation) Line 66: event UpgradeRebalance(address newImplementation) Line 67: event UpdateGnosisAddresses contracts/config/assetManagement/FeeManagement.sol Line 37: event ProposeManagementFee(uint256 newManagementFee) Line 38: event ProposePerformanceFee(uint256 newPerformanceFee)

	Line 39: event ProposeEntryAndExitFee(uint256 newEntryFee, uint256 newExitFee) Line 40: event UpdateManagementFee(uint256 newManagementFee) Line 41: event UpdatePerformanceFee(uint256 newPerformanceFee) Line 42: event UpdateEntryAndExitFee(uint256 newEntryFee, uint256 newExitFee) contracts/config/assetManagement/PortfolioSettings.sol Line 31: event TransferabilityUpdated(bool _transferable, bool _publicTransfers) Line 33: event MinPortfolioTokenHoldingAmountUpdated Line 36: event InitialPortfolioAmountUpdated(uint256 _newInitialPortfolioAmount) contracts/config/assetManagement/TokenWhitelistManagement.sol Line 21: event TokenWhitelisted(address[] tokens) Line 22: event TokensRemovedFromWhitelist(address[] tokens) contracts/config/assetManagement/TreasuryManagement.sol Line 19: event TreasuryUpdated(address newTreasury) ccontracts/config/assetManagement/UserWhitelistManagement.sol Line 14: event UserWhitelisted(address[] users) Line 15: event UserRemovedFromWhitelist(address[] users) contracts/config/protocol/OracleManagement.sol Line 18: event OracleUpdated(address newOracle) contracts/config/protocol/ProtocolFeeManagement.sol Line 22: event ProtocolFeeUpdated(uint256 newProtocolFee) Line 23: event ProtocolStreamingFeeUpdated(uint256 newProtocolStreamingFee) contracts/config/protocol/ProtocolTreasuryManagement.sol Line 17: event TreasuryUpdated(address newTreasury) contracts/config/protocol/SolverManagement.sol Line 16: event SolverHandlerEnabled(address handler)

Line 17: event SolverHandlerDisabled(address handler)

contracts/config/protocol/SystemSettings.sol
Line 21: event ProtocolPaused(bool paused)
Line 23: event MinPortfolioTokenHoldingAmountUpdated(uint256 newAmount)
Line 24: event CooldownPeriodUpdated(uint256 newPeriod)
Line 25: event MinInitialPortfolioAmountUpdated(uint256 newAmount)

contracts/config/protocol/TokenManagement.sol
Line 23: event TokensEnabled(address[] tokens)
Line 26: event TokenDisabled(address token)

contracts/core/config/VaultConfig.sol
Line 39: event Deposited
Line 44: event Withdrawn contracts/core/interfaces/IAllowanceTransfer.sol
Line 42: event Lockdown(address indexed owner, address token, address spender)

contracts/core/interfaces/IPortfolio.sol
Line 41: event Transfer(address indexed from, address indexed to, uint256 value)
Line 47: event Approval(address indexed owner, address indexed spender, uint256 value)

contracts/core/management/TokenExclusionManager.sol
Line 53: event UserRecordUpdated
Line 59: event TokenRecordUpdated
Line 66: event SnapShotCreated(uint256 snapshotId)
Line 68: event UserClaimedToken(address user, uint256 claimedTill)

contracts/fee/FeeEvents.sol
Line 5: event FeesToBeMinted
Line 12: event ManagementFeeCalculated
Line 18: event EntryExitFeeCharged
Line 23: event PerformanceFeeCalculated

	contracts/mock/upgradeability/TokenExclusionManagerV3_2.sol Line 54: event UserRecordUpdated Line 60: event TokenRecordUpdated Line 67: event SnapShotCreated(uint256 snapshotId) Line 69: event UserClaimedToken(address user, uint256 claimedTill) contracts/mock/upgradeability/changedDependencies/v3_4/VaultConfigV3_4.sol Line 38: event Deposited Line 43: event Withdrawn contracts/oracle/PriceOracleAbstract.sol Line 37: event FeedAdded Line 42: event FeedUpdated contracts/rebalance/IRebalancing.sol Line 20: event FeeCharged(uint256 charged, address token, uint256 amount) Line 21: event UpdatedWeights(uint256 updated, uint96[] newDenorms) Line 22: event UpdatedTokens contracts/rebalance/Rebalancing.sol Line 27: event UpdatedTokens(address[] newTokens) Line 28: event PortfolioTokenRemoved
Result/Recommendation	Add indexed to the appropriate fields in the events to make them more accessible to off-chain tools. Ensure that each event uses three indexed fields if there are three or more fields, and index all fields if there are fewer than three.

6.2.19 Empty Blocks in _authorizeUpgrade Functions

Severity: LOW
Status: FIXED
File(s) affected: PortfolioFactory.sol, AssetManagementConfig.sol, ProtocolConfig.sol, Portfolio.sol, TokenExclusionManager.sol, FeeConfig.sol, PortfolioV3_2.sol, PortfolioV3_4.sol, TokenExclusionManagerV3_2.sol, FeeConfigV3_2.sol, Rebalancing.sol

Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	Several _authorizeUpgrade functions across multiple contracts contain empty blocks. Empty blocks do not contribute to the functionality of the contract and can be removed to improve code readability and maintainability.
Code	The following instances have been identified where _authorizeUpgrade functions contain empty blocks: contracts/PortfolioFactory.sol Line 512: function _authorizeUpgrade contracts/config/assetManagement/AssetManagementConfig.sol Line 99: function _authorizeUpgrade contracts/config/protocol/ProtocolConfig.sol Line 62: function _authorizeUpgrade contracts/core/Portfolio.sol Line 86: function _authorizeUpgrade contracts/core/management/TokenExclusionManager.sol Line 263: function _authorizeUpgrade contracts/fee/FeeConfig.sol Line 130: function _authorizeUpgrade contracts/mock/upgradeability/PortfolioV3_2.sol Line 98: function _authorizeUpgrade contracts/mock/upgradeability/PortfolioV3_4.sol Line 101: function _authorizeUpgrade

	contracts/mock/upgradeability/TokenExclusionManagerV3_2.sol Line 268: function _authorizeUpgrade contracts/mock/upgradeability/changedDependencies/fee_v3_2/FeeConfigV3_2.sol Line 127: function _authorizeUpgrade contracts/rebalance/Rebalancing.sol Line 322: function _authorizeUpgrade
Result/Recommendation	Remove the empty blocks from the _authorizeUpgrade functions to improve code readability and maintainability. If the function is required by an interface or abstract contract, consider adding a comment to indicate that the function is intentionally left empty.

6.2.20 Internal Functions Called Only Once Should Be Inlined

Severity: LOW
Status: FIXED
File(s) affected: VaultCalculations.sol, CooldownManager.sol, FeeCalculations.sol, RebalancingConfig.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	Several internal functions across multiple contracts are called only once. Instead of separating the logic into a separate function, consider inlining the logic into the calling function. This can reduce the number of function calls and improve readability.
Code	The following instances have been identified where internal functions are called only once and can be inlined: contracts/core/calculations/VaultCalculations.sol Line 65: function getUSDValueOfTokenBalance

	contracts/core/cooldown/CooldownManager.sol Line 72: function getRemainingCoolDown(address _user) internal view returns (uint256) contracts/fee/FeeCalculations.sol Line 70: function calculateStreamingFee contracts/library/GnosisDeployer.sol Line 62: function generateByteCode contracts/oracle/PriceOracleAbstract.sol Line 63: function getAggregator contracts/rebalance/RebalancingConfig.sol Line 75: function verifyCompleteTokenSale Line 91: function verifyNewTokenList Line 120: function verifyZeroBalanceForRemovedTokens Line 148: function validateSolver(address _handler) internal view Line 184: function createNewTokenList
Result/Recommendation	Inline the logic of the identified functions into the calling functions to reduce the number of function calls and improve readability.

6.2.21 Unused Custom Errors in Multiple Contracts

Severity: LOW
Status: FIXED
File(s) affected: IAllowanceTransfer.sol, ErrorLibrary.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	Several custom errors are defined but not used in the codebase. It is recommended to remove these unused custom errors to improve code readability and maintainability. Unused code can lead to confusion and potential errors in the future.
Code	The following instances have been identified where custom errors are defined but not used:

	contracts/core/interfaces/IAllowanceTransfer.sol
	Line 12: error AllowanceExpired(uint256 deadline);
	Line 16: error InsufficientAllowance(uint256 amount);
	Line 19: error ExcessiveInvalidation();
	
	contracts/library/ErrorLibrary.sol
	Line 12: error ContractPaused();
	Line 34: error TokenWhitelistingNotEnabled();
	Line 38: error InvalidToken();

Result/Recommendation	Remove the unused custom errors from the codebase to improve readability and maintainability. This will also help in reducing the potential for confusion and errors in the future.

6.2.22 Missing Event Emission in _updatePriceOracle Function OracleManagement contract

Severity: LOW
Status: FIXED
File(s) affected: OracleManagement.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The _updatePriceOracle function in the contract updates the priceOracle address but does not emit an event to log this change. Emitting events is a best practice in Solidity as it provides transparency and allows external observers to track changes in the contract state. The absence of an event emission makes it difficult to track changes to the priceOracle address.

Code	Line 38 - 41 (OracleManagement.sol):
	
	`function _updatePriceOracle(address _newOracle) internal {`
	`    if (_newOracle == address(0)) revert ErrorLibrary.InvalidOracleAddress();`
	`    oracle = IPriceOracle(_newOracle);`

	}

Result/Recommendation	Add an event emission to the _updatePriceOracle function to log the old and new priceOracle addresses. This will improve transparency and allow external observers to track changes to the priceOracle address.
	
	`event UpdatePriceOracle(address indexed oldOracle, address indexed newOracle);`
	`function _updatePriceOracle(address _newOracle) internal {`
	`    if (_newOracle == address(0)) revert ErrorLibrary.InvalidOracleAddress();`
	`address _oldOracle = address(oracle);`
	`oracle = IPriceOracle(_newOracle);`
	`    emit UpdatePriceOracle(_oldOracle, _newOracle);`
	`}`

6.2.23 Missing Check for Identical Protocol Fee in updateProtocolFee and updateProtocolStreamingFee Functions in ProtocolFeeManagement contract

Severity: LOW
Status: FIXED
File(s) affected: ProtocolFeeManagement.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/224

Attack / Description	The updateProtocolFee and updateProtocolStreamingFee functions in the contract update the protocol fee and protocol streaming fee, respectively. However, these functions do not check if the new fee is the same as the current fee. Adding a check to ensure that the new fee is different from the current fee can prevent unnecessary state changes and gas consumption.

Code	Line 37 - 44 (ProtocolFeeManagement.sol):
	
	`function updateProtocolFee(`
	`    uint256 _newProtocolFee`

	```
	  ) external onlyProtocolOwner {
	    if (_newProtocolFee > 5_000 || _newProtocolFee == protocolFee)
	      revert ErrorLibrary.InvalidProtocolFee();
	    protocolFee = _newProtocolFee;
	    emit ProtocolFeeUpdated(_newProtocolFee);
	  }
	```
	 Line 46 - 56 (ProtocolFeeManagement.sol):
	```
	function updateProtocolStreamingFee(
	    uint256 _newProtocolStreamingFee
	  ) external onlyProtocolOwner {
	    if (
	      _newProtocolStreamingFee > 100 ||
	      _newProtocolStreamingFee == protocolStreamingFee
	    ) revert ErrorLibrary.InvalidProtocolStreamingFee();

	    protocolStreamingFee = _newProtocolStreamingFee;
	    emit ProtocolStreamingFeeUpdated(_newProtocolStreamingFee);
	  }
	```

Result/Recommendation	Add a check to ensure that the new fee is different from the current fee before updating the state. This can be achieved by adding a require statement to compare the new fee with the current fee.

6.2.24 Missing Check for Identical Treasury Address in updateVelvetTreasury Function in ProtocolTreasuryManagement contract

Severity: LOW
Status: FIXED
File(s) affected: ProtocolTreasuryManagement.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/224

Attack / Description	The updateVelvetTreasury function in the contract updates the velvetTreasury address but does not check if the new address is the same as the current address. Adding a check to ensure that the new address is different from the current address can prevent unnecessary state changes and gas consumption.

Code	Line 34 - 39 (ProtocolTreasuryManagement.sol)
	
	`function updateVelvetTreasury(`
	`    address _newVelvetTreasury`
	`  ) external onlyProtocolOwner {`
	`    _updateVelvetTreasury(_newVelvetTreasury);`
	`    emit TreasuryUpdated(_newVelvetTreasury);`
	`  }`

Result/Recommendation	Add a check to ensure that the new treasury address is different from the current treasury address before updating the state. This can be achieved by adding a require statement to compare the new address with the current address.

6.2.25 Missing Minimum Cooldown Period Check in setCoolDownPeriod Function

Severity: LOW

Status: FIXED
File(s) affected: SystemSettings.sol
Update: We have set a minimum cooldown period of 1 minute and a maximum of 14 days to ensure that the funds cannot be locked indefinitely by setting an excessively high cooldown period. https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The setCoolDownPeriod function in the SystemSettings contract allows the protocol owner to set a new cooldown period. However, there is no check to ensure that the new cooldown period is above a minimum acceptable value. Setting an extremely low cooldown period could lead to potential issues, such as frequent state changes and increased gas consumption. Adding a check to enforce a minimum cooldown period can help mitigate these risks.

Code	Line 34 - 39 (SystemSettings.sol)
	
	`function setCoolDownPeriod(`
	`  uint256 _newCooldownPeriod`
	`) external onlyProtocolOwner {`
	`  if (_newCooldownPeriod < 1 minutes || _newCooldownPeriod > 14 days)`
	`    revert ErrorLibrary.InvalidCooldownPeriod();`
	`  cooldownPeriod = _newCooldownPeriod;`
	`  emit CooldownPeriodUpdated(_newCooldownPeriod);`
	`}`

Result/Recommendation	Add a check to ensure that the new cooldown period is above a minimum acceptable value before updating the state. This can be achieved by adding a require statement to compare the new cooldown period with a predefined minimum value.

6.2.26 Prevent Division by Zero in getDepositToVaultBalanceRatio function

Severity: LOW
Status: FIXED

File(s) affected: TokenCalculations.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	In the getDepositToVaultBalanceRatio function, division by zero can occur if tokenBalance is zero, which could lead to unexpected reverts. Adding a check to ensure tokenBalance is greater than zero before performing the division will prevent this issue.

Code	Line 43 - 51 (TokenCalculations.sol)
	`function _getDepositToVaultBalanceRatio(`
	`    uint256 depositAmount,`
	`    uint256 tokenBalance`
	`  ) internal pure returns (uint256) {`
	`    if (tokenBalance == 0) revert ErrorLibrary.BalanceOfVaultIsZero();`
	
	`    // Calculate the deposit ratio to 18 decimal precision`
	`    return (depositAmount * ONE_ETH_IN_WEI) / tokenBalance;`
	`  }`

Result/Recommendation	Add a check to ensure tokenBalance is greater than zero before performing the division.
	
	`function getDepositToVaultBalanceRatio(`
	`  uint256 depositAmount,`
	`  uint256 tokenBalance`
	`) internal pure returns (uint256) {`
	`  require(tokenBalance > 0, "Token balance must be greater than zero");`
	`  // Calculate the deposit ratio to 18 decimal precision`
	`  return (depositAmount * ONE_ETH_IN_WEI) / tokenBalance;`
	`}`

6.2.27 Missing Cooldown Period Check for Deposit

Severity: LOW
Status: ACKNOWLEDGED
File(s) affected: VaultManager.sol

Attack / Description	The multiTokenDeposit function lacks a cooldown period check, which is present in the multiTokenWithdrawal function. This inconsistency can lead to scenarios where deposits are made without respecting the intended cooldown period, potentially causing operational issues. Also as large frequent deposits can also destabilize the system in the same way withdrawals do, but this does not harm the system on massive level, might be for few seconds.

Code	Lines: 53 - 90 (VaultManager.sol)
	
	`function multiTokenDeposit(`
	`  uint256[] calldata depositAmounts,`
	`  uint256 _minMintAmount,`
	`  IAllowanceTransfer.PermitBatch calldata _permit,`
	`  bytes calldata _signature`
	`) external virtual nonReentrant {`
	`  // Verify that the user is allowed to deposit and that the system is not paused.`
	`  _beforeDepositCheck(msg.sender, tokens.length, _permit);`
	`  // Charge any applicable fees.`
	`  _chargeFees();`
	
	`  permit2.permit(msg.sender, _permit, _signature);`
	
	`  // Process the multi-token deposit, adjusting for vault token ratios.`
	`  uint256 _depositRatio = _multiTokenDeposit(depositAmounts);`
	`  uint256 _totalSupply = totalSupply();`
	
	`  uint256 tokenAmount;`

	`// If the total supply is zero, this is the first deposit, and tokens are minted based on the initial amount.`
	`if (_totalSupply == 0) {`
	`  tokenAmount = assetManagementConfig().initialPortfolioAmount();`
	`  // Reset the high watermark to zero if it's not the first deposit.`
	`  feeModule().updateHighWaterMark(0);`
	`} else {`
	`  // Calculate the amount of portfolio tokens to mint based on the deposit.`
	`  tokenAmount = getTokenAmountToMint(_depositRatio, _totalSupply);`
	`}`
	
	`// Ensure the minted amount meets the user's minimum expectation to mitigate slippage.`
	`verifyUserMintedAmount(tokenAmount, _minMintAmount);`
	
	`// Mint the calculated portfolio tokens to the user, applying any cooldown periods.`
	`tokenAmount = _mintTokenAndSetCooldown(msg.sender, tokenAmount);`
	
	`// Notify listeners of the deposit event.`
	`emit Deposited(address(this), msg.sender, tokenAmount);`
	`}`

Result/Recommendation	Add a cooldown period check in the multiTokenDeposit function similar to the multiTokenWithdrawal function to ensure consistency and proper enforcement of cooldown periods.
	
	`function multiTokenDeposit(`
	`  uint256[] calldata depositAmounts,`
	`  uint256 _minMintAmount,`
	`  IAllowanceTransfer.PermitBatch calldata _permit,`
	`  bytes calldata _signature`
	`) external virtual nonReentrant {`
	`  // Verify that the user is allowed to deposit and that the system is not paused.`

	```
	  _beforeDepositCheck(msg.sender, tokens.length, _permit);
	  // Validate the cooldown period of the user.
	  checkCoolDownPeriod(msg.sender);
	  // Charge any applicable fees.
	  _chargeFees();

	  permit2.permit(msg.sender, _permit, _signature);

	  // Process the multi-token deposit, adjusting for vault token ratios.
	  uint256 _depositRatio = _multiTokenDeposit(depositAmounts);
	  uint256 _totalSupply = totalSupply();

	  uint256 tokenAmount;

	  // If the total supply is zero, this is the first deposit, and tokens are minted based on the initial amount.
	  if (_totalSupply == 0) {
	    tokenAmount = assetManagementConfig().initialPortfolioAmount();
	    // Reset the high watermark to zero if it's not the first deposit.
	    feeModule().updateHighWaterMark(0);
	  } else {
	    // Calculate the amount of portfolio tokens to mint based on the deposit.
	    tokenAmount = getTokenAmountToMint(_depositRatio, _totalSupply);
	  }

	  // Ensure the minted amount meets the user's minimum expectation to mitigate slippage.
	  verifyUserMintedAmount(tokenAmount, _minMintAmount);

	  // Mint the calculated portfolio tokens to the user, applying any cooldown periods.
	  tokenAmount = _mintTokenAndSetCooldown(msg.sender, tokenAmount);

	  // Notify listeners of the deposit event.
	```

	emit Deposited(address(this), msg.sender, tokenAmount);
	}

INFORMATIONAL ISSUES

During the audit, softstack‘s experts found **23 Informational issue** in the code of the smart contract.

6.2.28 Missing inheritance

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: N/A
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	Some contracts are missing inheritance from their respective interfaces or base contracts.
Code	AccessController should inherit from IAccessController TokenExclusionManager should inherit from ITokenExclusionManager PortfolioCalculations should inherit from AssetManagerCheck EnsoHandler should inherit from IIntentHandler EnsoHandlerBundled should inherit from IIntentHandler
Result/Recommendation	To resolve this issue, ensure that the contracts inherit from their respective interfaces or base contracts. This will enforce the implementation of all required functions and maintain consistency across the codebase.

6.2.29 Too Many Digits

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: FeeConfig.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The contract FeeConfig contains a literal with too many digits, making it difficult to read and review.
Code	MIN_MINT_FEE = 1000000 (contracts/fee/FeeConfig.sol#34)
Result/Recommendation	To improve readability and reduce the risk of errors, use: - Ether suffix - Time suffix - Scientific notation - uint256 constant MIN_MINT_FEE = 1_000_000; // Using underscores for readability

6.2.30 Unused Imports

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: PortfolioFactory.sol, VelvetSafeModule.sol, GnosisDeployer.sol, VaultManager.sol, VaultCalculations.sol, VaultManagerV3_2.sol, VaultManagerV3_4.sol, Rebalancing.sol, RebalancingConfig.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	Importing files that are not used in the contract likely indicates a mistake. These unused imports should be removed to avoid confusion and improve code clarity.

Code	PortfolioFactory (contracts/PortfolioFactory.sol#17)
	import {Clones} from "@openzeppelin/contracts/proxy/Clones.sol";
	
	VelvetSafeModule (contracts/vault/VelvetSafeModule.sol#14)
	import {TransferHelper} from "@uniswap/lib/contracts/libraries/TransferHelper.sol";
	
	GnosisDeployer (contracts/library/GnosisDeployer.sol#5-9)
	import {IVelvetSafeModule} from "../vault/IVelvetSafeModule.sol";
	import {MultiSend} from "@gnosis.pm/safe-contracts/contracts/libraries/MultiSend.sol";

	import {Module} from "@gnosis.pm/zodiac/contracts/core/Module.sol";
	
	VaultManager (contracts/core/management/VaultManager.sol#4-7)
	import {TransferHelper} from "@uniswap/lib/contracts/libraries/TransferHelper.sol";
	import {IERC20Upgradeable} from "@openzeppelin/contracts-upgradeable-4.3.2/token/ERC20/IERC20Upgradeable.sol";
	
	VaultCalculations (contracts/core/calculations/VaultCalculations.sol#4)
	import {IERC20Upgradeable} from "@openzeppelin/contracts-upgradeable-4.3.2/token/ERC20/IERC20Upgradeable.sol";
	
	VaultManagerV3_2 (contracts/mock/upgradeability/changedDependencies/v3_2/VaultManagerV3_2.sol#4-7)
	import {IERC20Upgradeable} from "@openzeppelin/contracts-upgradeable-4.3.2/token/ERC20/IERC20Upgradeable.sol";
	import {TransferHelper} from "@uniswap/lib/contracts/libraries/TransferHelper.sol";
	VaultManagerV3_4 (contracts/mock/upgradeability/changedDependencies/v3_4/VaultManagerV3_4.sol#4-7)
	import {TransferHelper} from "@uniswap/lib/contracts/libraries/TransferHelper.sol";
	import {IERC20Upgradeable} from "@openzeppelin/contracts-upgradeable-4.3.2/token/ERC20/IERC20Upgradeable.sol";
	
	Rebalancing (contracts/rebalance/Rebalancing.sol#7)
	import {IERC20Upgradeable} from "@openzeppelin/contracts-upgradeable-4.3.2/interfaces/IERC20Upgradeable.sol"; IERC20Upgradeable (node_modules/@openzeppelin/contracts-upgradeable-4.3.2/interfaces/IERC20Upgradeable.sol#6)
	import "../token/ERC20/IERC20Upgradeable.sol";
	
	RebalancingConfig (contracts/rebalance/RebalancingConfig.sol#4)
	import {IERC20Upgradeable} from "@openzeppelin/contracts-upgradeable-4.3.2/token/ERC20/IERC20Upgradeable.sol";

Result/Recommendation	Remove the unused imports. If the imports are needed later, they can be added back.

6.2.31 Non-compliance with Solidity Style Guide in AssetManagerCheck and OwnableCheck Contracts

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: AssetManagerCheck.sol, OwnableCheck.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/161

Attack / Description	The AssetManagerCheck and OwnableCheck contracts does not follow the recommended ordering of elements as specified in the Solidity style guide. According to the style guide, the order within a contract should be: - Type declarations - State variables - Events - Modifiers - Functions
Code	N/A
Result/Recommendation	Reorder the elements in the AssetManagerCheck and OwnableCheck contracts to comply with the Solidity style guide.

6.2.32 Missing NatSpec Comments in enableSolverHandler Function

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: SolverManagement.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/160

Attack / Description	The enableSolverHandler function in the SolverManagement contract lacks NatSpec comments, which are essential for providing clear documentation for both developers and end users. According to the Solidity documentation and best practices, public and external functions that aren't view or pure should include NatSpec comments to explain their purpose, parameters, and any side effects.

Code	Line 34 - 38 (SolverManagement.sol)
	
	`function enableSolverHandler(address _handler) external onlyProtocolOwner {`
	`    if (_handler == address(0)) revert ErrorLibrary.InvalidAddress();`
	`    solverHandler[_handler] = true;`
	`    emit SolverHandlerEnabled(_handler);`
	`  }`

Result/Recommendation	Add NatSpec comments to the enableSolverHandler function to improve code readability and maintainability. Include @notice and @dev tags to describe the function's purpose and behavior, and use @param tags to document its parameters.

6.2.33 Non-compliance with Solidity Naming Convention in MathUtils Library

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: MathUtils.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/160

Attack / Description	The MathUtils library contains internal functions that do not follow the Solidity style guide recommendation for naming conventions. According to the style guide, non-external variable and

	function names should begin with an underscore to differentiate them from external and public functions.
Code	min function (contracts/core/calculations/MathUtils.sol#18) max function (contracts/core/calculations/MathUtils.sol#28) subOrZero function (contracts/core/calculations/MathUtils.sol#38)
Result/Recommendation	Rename the internal functions to begin with an underscore to comply with the Solidity style guide. This will improve code readability and maintainability by clearly distinguishing internal functions from external and public ones.

6.2.34 Redundant Return Statement in getTokenBalancesOf Function

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: TokenBalanceLibrary.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/160

Attack / Description	The getTokenBalancesOf function in the TokenBalanceLibrary contract defines a named return variable vaultBalances. According to best practices and the Solidity style guide, adding an explicit return statement when a function defines a named return variable is redundant. This redundancy can lead to confusion and reduce code readability.

Code	Line 22 - 31 (TokenBalanceLibrary sol)
	
	`function getTokenBalancesOf(`
	`  address[] memory portfolioTokens,`
	`  address _vault`
	` ) public view returns (uint256[] memory vaultBalances) {`
	`  uint256 portfolioLength = portfolioTokens.length;`

	`vaultBalances = new uint256[](portfolioLength); // Initializes the array to hold fetched balances.`
	`for (uint256 i; i < portfolioLength; i++) {`
	`  vaultBalances[i] = _getTokenBalanceOf(portfolioTokens[i], _vault); // Fetches balance for each token.`
	`}`
	`}`

Result/Recommendation	Remove the explicit return statement and rely on the named return variable to automatically return the value. This will improve code readability and adhere to best practices.

6.2.35 Use Enums for Array Indices Instead of Numeric Literals in PortfolioCalculations Contract

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
File(s) affected: PortolfioCalculations.sol
Update: This contract is only used to fetch data for the front-end, we do not believe any changes are necessary.

Attack / Description	The PortolfioCalculations contract uses numeric literals to reference array indices, which can lead to errors and reduce code readability. According to best practices, array indices should be referenced via enums rather than numeric literals. This approach improves code maintainability and reduces the risk of errors associated with hard-coded index values.
Code	contracts/front-end-helpers/PortolfioCalculations.sol#36 contracts/front-end-helpers/PortolfioCalculations.sol#37 contracts/front-end-helpers/PortolfioCalculations.sol#39 contracts/front-end-helpers/PortolfioCalculations.sol#40
Result/Recommendation	Define an enum to represent the indices of the userAmounts and vaultBalance arrays. Use this enum to reference the array indices instead of numeric literals. This will make the code more readable and maintainable.

6.2.36 Potential Issues with abi.encodeWithSignature in GnosisDeployer Contract

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: GnosisDeployer.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/161

Attack / Description	The GnosisDeployer contract uses abi.encodeWithSignature to encode function calls. This approach is prone to errors such as typos in the function signature and incorrect parameter types. To avoid these pitfalls, it is recommended to use abi.encodeCall, which provides compile-time checks for function signatures and parameter types, ensuring safer and more reliable code.
Code	contracts/library/GnosisDeployer.sol#66 contracts/library/GnosisDeployer.sol#74
Result/Recommendation	Replace abi.encodeWithSignature with abi.encodeCall to ensure that the function signatures and parameter types are checked at compile time, reducing the risk of errors.

6.2.37 Use bytes.concat Instead of abi.encodePacked in GnosisDeployer Contract

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: GnosisDeployer.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/161

Attack / Description	The GnosisDeployer contract uses abi.encodePacked for concatenating bytes. Solidity version 0.8.4 introduces bytes.concat, which is a safer and more efficient method for concatenating bytes. Using bytes.concat can help avoid potential issues with incorrect concatenation, especially when dealing with mixed data types.
Code	contracts/library/GnosisDeployer.sol#67

Result/Recommendation	Replace abi.encodePacked with bytes.concat to ensure safer and more efficient concatenation of bytes. This change will also make the code more readable and maintainable.

6.2.38 Superfluous Initialization of Loop Variable in getTokenBalancesOf Function

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: TokenBalanceLibrary.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/160

Attack / Description	The getTokenBalancesOf function in the TokenBalanceLibrary contract initializes the loop variable i to zero explicitly. In Solidity, the default value for uninitialized variables is zero, making this explicit initialization redundant. Removing such redundant initializations can improve code readability and slightly optimize gas usage. similarly, In (contracts/front-end-helpers/PortolfioCalculations.sol#51)

Code	Line 22 - 31 (TokenBalanceLibrary.sol)
	
	`function getTokenBalancesOf(`
	`  address[] memory portfolioTokens,`
	`  address _vault`
	` ) public view returns (uint256[] memory vaultBalances) {`
	`  uint256 portfolioLength = portfolioTokens.length;`
	`  vaultBalances = new uint256[](portfolioLength); // Initializes the array to hold fetched balances.`
	`  for (uint256 i; i < portfolioLength; i++) {`
	`   vaultBalances[i] = _getTokenBalanceOf(portfolioTokens[i], _vault); // Fetches balance for each token.`
	`  }`
	` }`

Result/Recommendation	Remove the explicit initialization of the loop variable i to zero. This will adhere to best practices and improve code readability without affecting functionality.

6.2.39 Optimization Opportunity in _disableInitializers Function

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: N/A
Update: https://github.com/Velvet-Capital/v3-contract/pull/224

Attack / Description	The _disableInitializers function in the contract uses an older version of the OpenZeppelin library. A minor optimization has been introduced in the latest version of the OpenZeppelin library, which improves the gas efficiency of the _disableInitializers function by using != instead of <. This optimization reduces the gas cost by 2 units per call. OpenZeppelin/openzeppelin-contracts#3787

Code	`constructor() {`
	`  _disableInitializers();`
	` }`

Result/Recommendation	Update the OpenZeppelin library to the latest version to take advantage of the optimization in the _disableInitializers function. This will improve the gas efficiency of the contract.

6.2.40 Typographical Error in Smart Contract File Name PortolfioCalculations.sol

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: PortolfioCalculations.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The smart contract file intended to be named PortfolioCalculations.sol is mistakenly titled PortolfioCalculations.sol. This typographical error in the file name could lead to several potential issues in a project. The error may cause confusion during project maintenance or scaling, especially if developers rely on naming conventions to locate and reference specific functionalities.
Code	N/A
Result/Recommendation	To rectify and prevent potential disruptions caused by this typo: Immediate Renaming: Rename the file to PortfolioCalculations.sol immediately. This change should be carefully propagated throughout all references in the project's documentation, deployment scripts, and import statements within other smart contracts. Review and Testing: Conduct a thorough review of all project components to ensure that no other areas are impacted by the typo. Automated tests should be rerun to confirm that all functionalities are operating as expected post-renaming.

6.2.41 Unnecessary Variable Usage in _updateWeights Function

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: Rebalancing.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	In the _updateWeights function, the _sellTokens array is assigned to the sellToken variable within the loop. However, the _sellTokens array is accessed directly again in the subsequent line. This redundant access decreases readability and can lead to confusion regarding the purpose of the sellToken variable.
Code	Line 111 - 115 (Rebalancing.sol)

	`for (uint256 i; i < sellTokenLength; i++) {`
	`    address sellToken = _sellTokens[i];`
	`    if (sellToken == address(0)) revert ErrorLibrary.InvalidAddress();`
	`    portfolio.pullFromVault(sellToken, _sellAmounts[i], _handler);`
	`  }`

Result/Recommendation	Update the code to consistently use the sellToken variable once it has been assigned, rather than accessing the _sellTokens array again.
	
	`for (uint256 i = 0; i < sellTokenLength; i++) {`
	`  address sellToken = _sellTokens[i];`
	`  if (sellToken == address(0)) revert ErrorLibrary.InvalidAddress();`
	`  portfolio.pullFromVault(sellToken, _sellAmounts[i], _handler);`
	`}`
	
	This change improves the readability and consistency of the code by ensuring the sellToken variable is used as intended.

6.2.42 Missing Function Parameters in NatSpec Documentation

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: Rebalancing.sol, SystemSettings.sol, VaultManager.sol, TokenExclusionManager.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	There are multiple functions missing documentation for their parameters in the NatSpec comments. This lack of documentation can lead to confusion for users and developers about the purpose and usage of the parameters.
Code	Rebalancing.sol Lines: 179-186 SystemSettings.sol Lines: 59-66

	VaultManager.sol Lines: 46-58 TokenExclusionManager.sol Lines: 114-120

Result/Recommendation	Update the code to consistently use the sellToken variable once it has been assigned, rather than accessing the _sellTokens array again.
	
	for (uint256 i = 0; i < sellTokenLength; i++) {
	  address sellToken = _sellTokens[i];
	  if (sellToken == address(0)) revert ErrorLibrary.InvalidAddress();
	  portfolio.pullFromVault(sellToken, _sellAmounts[i], _handler);
	}
	
	Update the NatSpec documentations to include the missing parameters, providing a clear description of their purpose and usage.

6.2.43 Missing NatSpec Documentation for Interfaces

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: IUniswapV2Router02.sol, IPortfolioFactory.sol, IProtocolConfig.sol, IAssetManagementConfig.sol, IFeeModule
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	Several contracts are missing NatSpec documentation for their functions. NatSpec comments are essential for providing clear, standardized documentation for functions, making them easier to understand and use. The lack of documentation can lead to confusion and misuse of the functions by developers and users.
Code	IUniswapV2Router02.sol, IPortfolioFactory.sol, IProtocolConfig.sol, IAssetManagementConfig.sol, IFeeModule

Result/Recommendation	Add NatSpec documentation to all functions in the affected contracts. Ensure each function has clear and comprehensive descriptions for the parameters, return values, and any other relevant information.

6.2.44 Incorrect Documentation

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: Rebalancing.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	In the claimRewardTokens function, the NatSpec documentation for the rewardTokenBalanceAfter variable incorrectly states "all Tokens", which is misleading as it only fetches the balance of a specific token. This documentation error could confuse developers and users regarding the purpose of the rewardTokenBalanceAfter variable.

Code	Line 300 - 304 (Rebalancing.sol)
	
	`// Fetch the new balance of the reward token in the vault after the claim operation`
	`    uint256 rewardTokenBalanceAfter = _getTokenBalanceOf(`
	`      _tokenToBeClaimed,`
	`      _vault`
	`    );`

Result/Recommendation	Update the NatSpec documentation to accurately describe the rewardTokenBalanceAfter variable as fetching the balance of a specific token, not all tokens.
	
	`  // Fetch the new balance of the reward token in the vault after the claim operation`
	`  uint256 rewardTokenBalanceAfter = _getTokenBalanceOf(`
	`    _tokenToBeClaimed,`

	`_vault`
	`);`

6.2.45 Inconsistent Implementation of updateTokens Function in IRebalancing and Rebalancing Contracts

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: Rebalancing.sol, IRebalancing.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The updateTokens function in the IRebalancing interface is documented and implemented differently compared to the updateTokens function in the Rebalancing contract. This inconsistency can cause confusion for developers and users, as the expected parameters and behavior differ significantly.

Code	Lines: 36-46 (IRebalancing.sol)
	
	`/**`
	` * @notice The function rebalances the portfolio to the updated tokens with the updated weights`
	` * @param tokens The updated token list of the portfolio`
	` * @param denorms The new weights for the portfolio`
	` * @param _slippageSell The allowed slippage for selling tokens`
	` * @param _slippageBuy The allowed slippage for buying tokens`
	` */`
	`function updateTokens(`
	`  address[] memory tokens,`
	`  uint96[] memory denorms,`
	`  uint256[] calldata _slippageSell,`
	`  uint256[] calldata _slippageBuy`

	) external;
	
	Lines: 124-147 (Rebalancing.sol)
	
	`function updateTokens(`
	`  FunctionParameters.RebalanceIntent calldata rebalanceData`
	`) external virtual nonReentrant onlyAssetManager {`
	`  address[] calldata _sellTokens = rebalanceData._sellTokens;`
	`  address[] calldata _newTokens = rebalanceData._newTokens;`
	`  address[] memory _tokens = getCurrentTokens();`
	
	`  // Need a check here to confirm _newTokens has buyTokens in it`
	`  portfolio.updateTokenList(_newTokens);`
	
	`  // Perform token update and weights adjustment based on provided rebalance data.`
	`  _updateWeights(`
	`    _sellTokens,`
	`    _newTokens,`
	`    rebalanceData._sellAmounts,`
	`    rebalanceData._handler,`
	`    rebalanceData._callData`
	`  );`
	
	`  // Update the internal mapping to reflect changes in the token list post-rebalance.`
	`  verifyZeroBalanceForRemovedTokens(_tokens, _newTokens);`
	
	`  emit UpdatedTokens(_newTokens);`
	`}`

Result/Recommendation	Align the updateTokens function in the Rebalancing contract with the IRebalancing interface. Ensure that the parameters and their usage are consistent across both the interface and the implementation. Update the documentation to reflect these changes accurately.

6.2.46 Duplicate NatSpec Documentation for getUserAmountToDeposit Function

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: PortfolioCalculations.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/160

Attack / Description	The getUserAmountToDeposit function has duplicate NatSpec documentation comments. This redundancy can clutter the code and potentially confuse developers and users.
Code	Lines: 16-28 (PortfolioCalculations.sol) /** * @dev This function takes value of portfolio token amounts from user as input and returns the lowest amount possible to deposit to get the exact ratio of token amounts * @notice This function is helper function for user to get the correct amount/ratio of tokens to deposit * @param userAmounts array of amounts of portfolio tokens */ /** * @dev This function takes value of portfolio token amounts from user as input and returns the lowest amount possible to deposit to get the exact ratio of token amounts * @notice This function is helper function for user to get the correct amount/ratio of tokens to deposit * @param userAmounts array of amounts of portfolio tokens */ function getUserAmountToDeposit(

	`  uint256[] memory userAmounts`
	`) external view returns (uint256[] memory, uint256 _desiredShare);`

Result/Recommendation	Remove the duplicate NatSpec documentation and keep only one set of comments. Ensure that the documentation is clear and accurate.

6.2.47 Gas Optimization for setEmergencyPause Function

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: SystemSettings.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The setEmergencyPause function can be optimized for gas efficiency. The _state parameter is always passed as true when calling setProtocolPause(_state) initially, and always passed as false when calling setProtocolPause(_state) subsequently in the unpause logic. This can be simplified by directly passing the boolean literals true or false instead of the _state parameter.

Code	Lines: 59-73 (SystemSettings.sol)
	
	`/**`
	` * @notice This function allows us to pause the protocol before certain operations`
	` * @param _state Boolean parameter to set the pause/unpause state of the protocol`
	` */`
	`function setEmergencyPause(`
	`  bool _state,`
	`  bool _unpauseProtocol`
	`) external virtual onlyProtocolOwner {`
	`  if (_state) setProtocolPause(_state);`

	```
	  isProtocolEmergencyPaused = _state;

	  if (!_state && _unpauseProtocol) {
	    setProtocolPause(_state);
	  }
	}
	```

Result/Recommendation	Update the code to directly pass true or false to the setProtocolPause function where applicable.
	```
	/**
	 * @notice This function allows us to pause the protocol before certain operations
	 * @param _state Boolean parameter to set the pause/unpause state of the protocol
	 * @param _unpauseProtocol Boolean parameter to determine if the protocol should be unpaused
	after emergency pause is lifted
	 */
	function setEmergencyPause(
	  bool _state,
	  bool _unpauseProtocol
	) external virtual onlyProtocolOwner {
	  if (_state) setProtocolPause(true);
	  isProtocolEmergencyPaused = _state;

	  if (!_state && _unpauseProtocol) {
	    setProtocolPause(false);
	  }
	}
	```

6.2.48 Use of Constant ONE_ETH_IN_WEI for Gas Efficiency

Severity: INFORMATIONAL
Status: FIXED

File(s) affected: TokenCalculations.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/160

Attack / Description	In the calculateMintAmount function, the value 10 ** 18 is used, which can be replaced by the constant ONE_ETH_IN_WEI defined above. Using a constant can save gas and make the code more readable and maintainable.

Code	Lines: 14-29 (TokenCalculations.sol)
	
	`uint256 constant ONE_ETH_IN_WEI = 10 ** 18;`
	
	`function calculateMintAmount(`
	`  uint256 _userShare,`
	`  uint256 _totalSupply`
	`) internal pure returns (uint256) {`
	`  return (_userShare * _totalSupply) / ((10 ** 18) - _userShare);`
	`}`

Result/Recommendation	Update the code to use the ONE_ETH_IN_WEI constant instead of recalculating 10 ** 18.
	
	`uint256 constant ONE_ETH_IN_WEI = 10 ** 18;`
	
	`function calculateMintAmount(`
	`  uint256 _userShare,`
	`  uint256 _totalSupply`
	`) internal pure returns (uint256) {`
	`  return (_userShare * _totalSupply) / (ONE_ETH_IN_WEI - _userShare);`
	`}`

6.2.49 Unused Named Returns

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: VaultCalculations.sol, PortfolioToken.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The functions getVaultValueInUSD and _burnWithdraw have unused named returns. Named returns can improve readability and gas consumption, but if not used consistently throughout the function, they may introduce confusion and redundancy.

Code	Lines: 88-107 (VaultCalculations.sol)
	
	`function getVaultValueInUSD(`
	`  IPriceOracle _oracle,`
	`  address[] memory _tokens,`
	`  uint256 _totalSupply,`
	`  address _vault`
	`) external view returns (uint256 vaultValue) {`
	`  if (_totalSupply == 0) return 0;`
	`  if (_mintAndBurnCheck(exitFee, _to)) {`
	`    afterFeeAmount = feeModule()._chargeEntryOrExitFee(_mintAmount, exitFee);`
	`  }`
	
	`  _burn(_to, _mintAmount);`
	
	`  return afterFeeAmount;`
	`}`

Result/Recommendation	Either use the named returns consistently within the function or remove the naming to avoid confusion.

6.2.50 Assignment of tokensLength Variable Earlier in initToken Function

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: VaultConfig.sol
Update: https://github.com/Velvet-Capital/v3-contract/pull/223

Attack / Description	The initToken function checks the length of _tokens multiple times. Assigning the length to the tokensLength variable earlier can improve readability and efficiency by using this variable consistently throughout the function.

Code	Lines: 66-85 (VaultConfig.sol)
	
	`function initToken(address[] calldata _tokens) external onlySuperAdmin {`
	`uint256 _assetLimit = protocolConfig().assetLimit();`
	`if (_tokens.length > _assetLimit)`
	`  revert ErrorLibrary.TokenCountOutOfLimit(_assetLimit);`
	`if (tokens.length != 0) {`
	`  revert ErrorLibrary.AlreadyInitialized();`
	`}`
	`uint256 tokensLength = _tokens.length;`
	`for (uint256 i = 0; i < tokensLength; i++) {`
	`  address token = _tokens[i];`
	`  beforeInitCheck(token);`
	`  if (_previousToken[token]) {`
	`    revert ErrorLibrary.TokenAlreadyExist();`
	`  }`
	`  _previousToken[token] = true;`
	`  tokens.push(token);`
	`}`
	`resetPreviousTokenList(_tokens);`

	```
	    emit PublicSwapEnabled();
	  }
	```

Result/Recommendation	Assign the tokensLength variable earlier and use it consistently in the function.
	```
	function initToken(address[] calldata _tokens) external onlySuperAdmin {
	  uint256 _assetLimit = protocolConfig().assetLimit();
	  uint256 tokensLength = _tokens.length;
	  if (tokensLength > _assetLimit)
	    revert ErrorLibrary.TokenCountOutOfLimit(_assetLimit);
	  if (tokens.length != 0) {
	    revert ErrorLibrary.AlreadyInitialized();
	  }
	  for (uint256 i = 0; i < tokensLength; i++) {
	    address token = _tokens[i];
	    beforeInitCheck(token);
	    if (_previousToken[token]) {
	      revert ErrorLibrary.TokenAlreadyExist();
	    }
	    _previousToken[token] = true;
	    tokens.push(token);
	  }
	  resetPreviousTokenList(_tokens);
	  emit PublicSwapEnabled();
	}
	```

6.3 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✓
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✓
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✓
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✓
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✓
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✓
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✓
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✓

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✓
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✓
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✓
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✓
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✓
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✓
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✓

ID	Title	Relationships	Test Result
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✓
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✓
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✓
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✓
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✓
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✓
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✓
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✓

ID	Title	Relationships	Test Result
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✓
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✓
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓

6.4 Verify Claims

6.4.1 Compliance with Best Practices: The audit should ensure that the contracts adhere to smart contract best practices, including checking for common vulnerabilities such as reentrancy attacks and overflow/underflow issues.
Status: tested and verified ✓

6.4.2 Effective Role-Based Access Control: The audit confirms proper assignment and management of roles like `SUPER_ADMIN`, `ASSET_MANAGER`, and `WHITELIST_MANAGER`, ensuring only authorized entities execute privileged functions. The `AccessController` contract's role setup and management functions should be evaluated for correct implementation and security.
Status: tested and verified ✓

6.4.3 Secure Token Transfer Functions: Token transfer functions are audited to ensure resilience against common vulnerabilities, safeguarding against unauthorized transfers and balance manipulation.
Status: tested and verified ✓

6.4.4 Accurate Fee Calculation and Charging Mechanisms: The audit validates the accuracy and security of fee calculation and charging mechanisms in the `FeeModule`, `FeeCalculations`, and `VaultCalculations` contracts. This includes ensuring the correct implementation of management fees, performance fees, entry/exit fees, and their correct application during deposits and withdrawals.
Status: tested and verified ✓

6.4.5 Correct and Secure Initialization Processes: The audit verifies that the initialization processes in contracts are correctly implemented and securely executed.
Status: tested and verified ✓

7. Executive Summary

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

Overall, the audit identified a total of one issue, classified as follows:
- No critical issues were found.
- 2 high severity issues were found.
- 4 medium severity issues were found.
- 21 low severity issues were discovered
- 23 informational issues were identified

The audit report provides detailed descriptions of each identified issue, including severity levels, CWE classifications, and recommendations for mitigation. It also includes code snippets, where applicable, to demonstrate the issues and suggest possible fixes. Based on the nature of the finding and adherence to the business logic, we recommend that the Velvet Capital team review the suggestions.

Update 21.06.2024: All identified issues have been successfully mitigated by the Velvet Capital team in a timely manner.
Latest audited version: https://github.com/Velvet-Capital/velvet-core/commit/849629b1aacf32d84634d8c4ef1378527bce3bb3

Update 10.07.2024: Our team has confirmed that Velvet Capital has successfully migrated their codebase from a private to a public repository, now accessible at (https://github.com/Velvet-Capital/velvet-core). The source has been thoroughly audited, and the enhancements recommended by the Hats Finance bug bounty program (https://app.hats.finance/bug-bounties/velvet-capital-0xb495c253b33abd5cea007df2ff8ee9f61bc6d35e/scope) have been successfully implemented and re-checked.

Update 13.07.2024: On request we have re-considered the downgrade of medium issues into low severity.

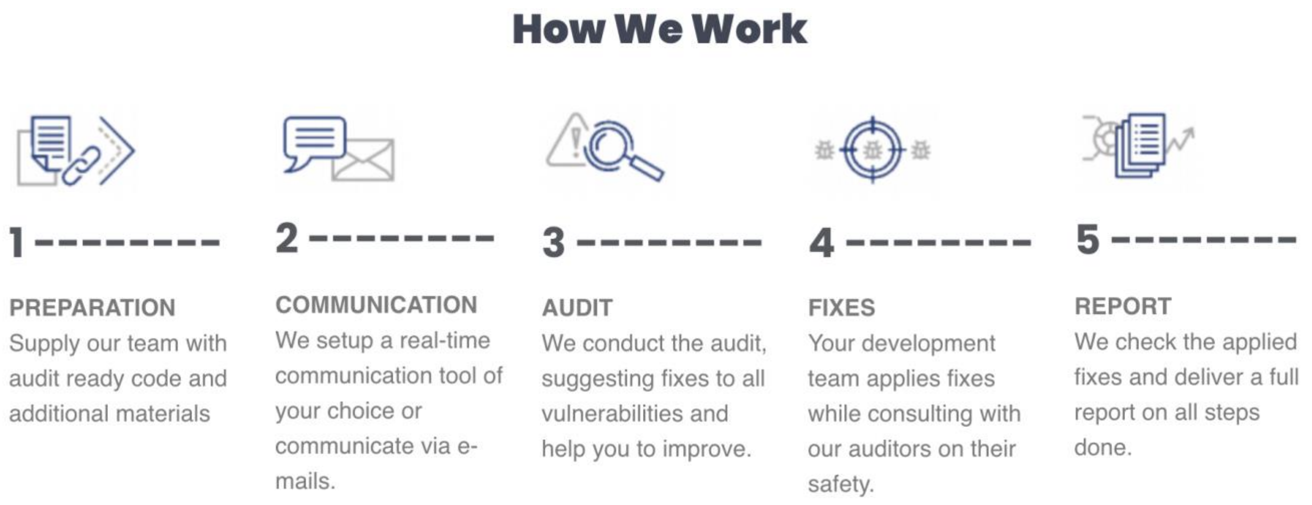
8. About the Auditor

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

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

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

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



How We Work

1 PREPARATION
Supply our team with audit ready code and additional materials

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

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

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

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