



**Fija Finance**

**Protocol**

**SMART CONTRACT AUDIT**

**21.07.2024**

**Made in Germany by Softstack.io**



## Table of contents

1. Disclaimer.....	5
2. About the Project and Company .....	6
2.1 Project Overview.....	7
3. Vulnerability & Risk Level .....	8
4. Auditing Strategy and Techniques Applied.....	9
4.1 Methodology .....	9
5. Metrics .....	10
5.1 Tested Contract Files .....	10
5.2 CallGraph.....	17
5.3 Inheritance Graph.....	18
5.4 Source Lines & Risk.....	19
5.5 Capabilities .....	20
5.6 Dependencies / External imports .....	21
5.7 Source Unites in Scope .....	22
6. Scope of Work.....	24
6.1 Findings Overview .....	25
6.2 Manual and Automated Vulnerability Test.....	27
6.2.1 Inconsistent Withdrawal Parameters Due to Post-Calculation Update of GMtokens.....	27
6.2.2 Potential Risk in Handling Large Redemption, When Requests Exceeding Available Balance.....	33
6.2.3 Potential Risk in Handling Large Withdrawal Requests, When Exceeding Available Balance .....	35
6.2.4 Lack of Flash Loan Callback Verification in _executeFlashloan Function of GMXv2Lib Library .....	37
6.2.5 Potential Hash Collision in status Function Due to abi.encodePacked() Usage.....	38



6.2.6 Precision Loss Due to Division Before Multiplication in GMXv2strategyReader Contract .....	39
6.2.7 Insufficient Liquidity Handling in Withdrawal and Redemption Functions of FijaVault2Txn contract .....	41
6.2.8 Vault Deposits Can Be Front-run.....	47
6.2.9 Unspecified Size Of Uint Variables .....	47
6.2.10 Improper Access Control .....	49
6.2.11 Prevalence of Magic Numbers in Codebase Compromises Maintainability .....	50
6.2.12 Missing Zero Address Validation in GMXv2strategyReader Constructor .....	52
6.2.13 Missing Event Emission in setContract Function of GMXv2strategyReader Contract .....	53
6.2.14 Multiple Reads of Storage Variables in GMXv2strategyReader Contract .....	54
6.2.15 Update Required for Math Library Usage .....	56
6.2.16 Unspecified Variable Visibility .....	57
6.2.17 Unclear Variable Value .....	58
6.2.18 Improper Input Validation.....	59
6.2.19 No Check for Contract Existence at Address .....	60
6.2.20 Missing Checks For Zero Address In Whitelisting Operations .....	62
6.2.21 Potential Manipulation of Execution Fee Calculation in ETH Deposits .....	63
6.2.22 Redundant Check for Zero Assets in Deposit Function Despite Modifier.....	65
6.2.23 Redundancy in Asset Calculation redeem Function.....	66
6.2.24 Missing Struct Packing.....	68
6.2.25 Magic Numbers .....	68
6.2.26 Misleading Variable Name.....	69
6.2.27 Missing Immutable Modifier.....	70
6.2.28 Redundant Payable Conversion.....	71
6.2.29 Unused Return Value .....	72



6.2.30 Missing Verification of Asset Addresses .....	73
6.4 Verify Claims .....	78
7. Executive Summary.....	80
8. About the Auditor .....	83



## 1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. 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 fija Finance GmbH. 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 (05.07.2023)	Layout
0.4 (06.07.2023) Vault	Automated Security Testing Manual Security Testing
0.5 (07.07.2023)	Verify Claims and Test Deployment
0.6 (10.07.2023)	Testing SWC Checks
0.9 (11.07.2023)	Summary and Recommendation
1.0 (11.07.2023)	Final document
1.1 (01.09.2023)	Re-check (ETH-capable vault and strategy)
1.2 (11.09.2023)	Layout
1.3 (18.09.2023) Curve Convex	Automated Security Testing Manual Security Testing
1.5 (20.09.2023)	Verify Claims and Test Deployment
1.6 (22.09.2023)	Testing SWC Checks
1.9 (25.09.2023)	Summary and Recommendation
2.0 (26.09.2023)	Final document
2.1 (02.10.2023)	Merging all audits into one document
2.2 (25.02.2024) Curve Convex & Vault	Re-check (all issues 6.2.1 – 6.2.15)
2.3 (10.04.2024) GMXv2 Strategy	Automated Security Testing Manual Security Testing
2.4 (18.04.2024)	Verify Claims and Test Deployment



2.5 (25.04.2024)	Summary and Recommendation
2.6 (25.04.2024)	Final document
2.7 (21.07.2024)	Re-check (all issues 6.2.1 – 6.2.30 and new fixes)

## 2. About the Project and Company

### Company address:

fija Finance GmbH  
Jahnstr. 48  
80469 München  
Germany



**Website:** <https://www.fija.finance>

**Twitter:** [https://twitter.com/fija\\_finance](https://twitter.com/fija_finance)

**LinkedIn:** <https://de.linkedin.com/company/fija-finance-gmbh>



## 2.1 Project Overview

Fija is a comprehensive DeFi platform providing an array of decentralized finance solutions. Fija utilizes the power of blockchain technology to develop and implement investment strategies that are automated and tokenized, offering Bafin regulated Security Tokens.

The heart of Fija's services is its unique approach to DeFi investment strategies, where predefined "deposit tokens", such as USDC or ETH, are used within DeFi apps to generate yields. These strategies are designed to strike an optimal balance between profitability and risk, allowing investors to diversify their portfolios and maximize potential returns.

One of Fija's standout features is its token, the Fija token. This ERC-20 token acts as a key to a customer's investment. As the strategy contract reaps fees, the value of the Fija token grows, mirroring the increasing value of the investment. Only whitelisted addresses can interact with the Fija smart contract, providing an added layer of security.

Fija's offerings are designed with seamless partner integration in mind. The platform works with resellers, allowing them to select and offer different investment strategies to their customers. These partners maintain control over the customer interface while also benefiting from technical support for integration from Fija, including the development of APIs and smart contract interfaces tailored to the partner's needs. This way, partners can offer a decentralized and trustless earn product to their customers without incurring integration costs.

In addition to its investment strategies and reseller partnerships, Fija offers a scoring system for its strategies. This scoring system assesses the security of the used blockchain and protocols and assigns each strategy a safety score. This helps investors choose a strategy that best suits their risk appetite.

Fija has a vision of democratizing access to DeFi by allowing more people and institutions to tap into decentralized finance's potential, while ensuring compliance and trust in the process.

### 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)
./strategies/CurveConvex/errors.sol	35d125bba091f64bb3c0fd08da810f09
./strategies/CurveConvex/CurveConvexStrategy.sol	d14b06a0134f0712d92de4029ff49450
./strategies/CurveConvex/CurveConvexStrategyProtocol.sol	dbfac1bca9e8c78468d90d544312f631
./strategies/CurveConvex/CurveConvexPeriphery.sol	a38a109f431280ed68bf2c0860b66f23
./strategies/CurveConvex/ICurveConvexPeriphery.sol	c572959fbd23e9ad809986b2aee1febf
./strategies/CurveConvex/CurveConvexBase.sol	0c66009d17f2708e150714d39bb64d86
./protocols/curve/interfaces/ICurveMetaRegistry.sol	26063905ace52c138b18ad45e8223b27
./protocols/curve/interfaces/IExchangeRegistry.sol	a38c2181ebeadab79d24cf2ab0b9ce5d
./protocols/curve/interfaces/IAddressProvider.sol	20a47054c2b6d0d7ce60ad68b476e620
./protocols/curve/interfaces/ICurve.sol	f60e3b58c3d33dc0857a071a670d3f03
./protocols/convex/interfaces/IBooster.sol	eb98dce11b3916fecbe40eabba3a3b36
./protocols/convex/interfaces/ICvxMining.sol	bef1cd797cc24f582e4078c89a6d42a6
./protocols/convex/interfaces/IRewardStaking.sol	d04f05147b14476c69a9893209762efe
./protocols/convex/interfaces/IApr.sol	0d0805c573adeec18f6690f0d905d162

File	Fingerprint (MD5)
------	-------------------



./base/FijaERC4626Base.sol	5383d482612c200492125c15b1f7ca44
./base/errors.sol	48c752e436a1a03dda5747b2cb4c549a
./base/FijaACL.sol	967e372e914dc4c3fbb92269c798a659
./base/FijaVault.sol	9208a31e4bb43a631fab8e44bc08f07
./interfaces/IFijaStrategy.sol	9e6ccd24ed71a07f213622908ab3ccd2
./interfaces/IFijaACL.sol	63823fe906498591f34f0489b8d9a39e
./interfaces/IFijaERC4626Base.sol	7b284d6ede8251e6ede954fbd6322203
./interfaces/IFijaVault.sol	e3c8f9cad7c6b6b04ce9d4d37e9df4e6

#### Updates Version (01.09.2023)

File	Fingerprint (MD5)
./base/FijaERC4626Base.sol	e9cd9fb359b745f54e0a97a5d9e3667f
./base/errors.sol	8e31c86057bd96a0d01903364469a4cf
./base/FijaACL.sol	967e372e914dc4c3fbb92269c798a659
./base/FijaVault.sol	d11d98af0f84c7bf495056f6733bce66
./interfaces/IFijaStrategy.sol	9e6ccd24ed71a07f213622908ab3ccd2
./interfaces/IFijaACL.sol	63823fe906498591f34f0489b8d9a39e
./interfaces/IFijaERC4626Base.sol	6af43b2c33b5a3d8724d7b7b582e10a1
./interfaces/IFijaVault.sol	e3c8f9cad7c6b6b04ce9d4d37e9df4e6

#### Updates Version (25.02.2024)

File	Fingerprint (MD5)
./strategies/CurveConvex/errors.sol	35d125bba091f64bb3c0fd08da810f09
./strategies/CurveConvex/CurveConvexStrategy.sol	e757aac94a3952fb30e5561534c31a0d
./strategies/CurveConvex/CurveConvexStrategyProtocol.sol	970660ff01910d2ba688857f225cfe50
./strategies/CurveConvex/CurveConvexPeriphery.sol	c9521973bc9559b5cd2d7f219bd719ce
./strategies/CurveConvex/ICurveConvexPeriphery.sol	c572959fbd23e9ad809986b2aee1fefb
./strategies/CurveConvex/CurveConvexBase.sol	be2c9cfe11f3f0dacb1b3bd49c005b36
./protocols/curve/interfaces/ICurveMetaRegistry.sol	26063905ace52c138b18ad45e8223b27
./protocols/curve/interfaces/IExchangeRegistry.sol	a38c2181eheadab79d24cf2ab0b9ce5d



./protocols/curve/interfaces/IAddressProvider.sol	20a47054c2b6d0d7ce60ad68b476e620
./protocols/curve/interfaces/ICurve.sol	f60e3b58c3d33dc0857a071a670d3f03
./protocols/convex/interfaces/IBooster.sol	eb98dce11b3916fecbe40eabba3a3b36
./protocols/convex/interfaces/ICvxMining.sol	bef1cd797cc24f582e4078c89a6d42a6
./protocols/convex/interfaces/IRewardStaking.sol	d04f05147b14476c69a9893209762efe
./protocols/convex/interfaces/IApr.sol	0d0805c573adeec18f6690f0d905d162

File	Fingerprint (MD5)
./base/FijaERC4626Base.sol	7c0ae8d613264b35bb8d8ab26db677a8
./base/errors.sol	92c9666bee5bcfd56cb4a1e3fd2888a0
./base/FijaACL.sol	0632c8de5ea26de67f26d6b33d82de42
./base/FijaVault.sol	9795418c72187219a9dcacec2a973416
./interfaces/IFijaStrategy.sol	9e6ccd24ed71a07f213622908ab3ccd2
./interfaces/IFijaACL.sol	63823fe906498591f34f0489b8d9a39e
./interfaces/IFijaERC4626Base.sol	6af43b2c33b5a3d8724d7b7b582e10a1
./interfaces/IFijaVault.sol	9e5c961b1cffb31cd9d245cc7d41ed21

Updates Version (25.04.2024)

File	Fingerprint (MD5)
./strategies/GMXv2/GMXv2Helpers.sol	11935e86e305a35d70a5b2418ee3ddc3
./strategies/GMXv2/errors.sol	e53a217e42b530c2e82472942872fc0a
./strategies/GMXv2/GMXv2strategy.sol	c7fd3a06a0c6cf7b4022db3e07d1ccdf
./strategies/GMXv2/GMXv2strategyReader.sol	c1ac47b1587c763dca38e263cdc07805
./strategies/GMXv2/GMXv2strategyBase.sol	5c4036bcbaf3ed6a95f4fd985fd29df7
./strategies/GMXv2/GMXv2Lib.sol	97606f24cdb72bed20e0d64eaaa12cf7
./strategies/GMXv2/GMXv2Keys.sol	d47b8982891e4c32e13158a548260dea
./strategies/GMXv2/GMXv2Type.sol	2f206d91bb111abda5d5cea3e2acf562
./strategies/GMXv2/IGMXv2strategyReader.sol	1751ad5225bec717e0bf9be490ae539f

./protocols/gmxv2/EventUtils.sol	b5bc054cf4736c43cebf3e5e8f6bf5d
./protocols/gmxv2/IWithdrawHandler.sol	1efa83619e1635d0155faff98b112bae
./protocols/gmxv2/IDatastore.sol	294be4815f67cafd7b3a74bc60a6803e
./protocols/gmxv2/Deposit.sol	9742160e393f15b6879329e2f543ad24
./protocols/gmxv2/IRoleStore.sol	fdd149fce0e3df25088f2448d6d5280c
./protocols/gmxv2/IDepositHandler.sol	a9987d83e3ef3f1190df89674649c893
./protocols/gmxv2/IReader.sol	66aba50731e63ccdd93ead70bd4c899a
./protocols/gmxv2/Withdrawal.sol	7e82f4201326d0160985d341405d0a87
./protocols/gmxv2/Price.sol	8a52f318cad57af698ae20fc3b046a2d
./protocols/gmxv2/Precision.sol	8886c3040951a91086d580c1050e499d
./protocols/gmxv2/IExchangeRouter.sol	511a9a3e48b0f60437889e9a5783ba72
./protocols/gmxv2/Market.sol	f4c5ec3d1d8fb0480831b9304f937e02
./helpers/FijaSwapper.sol	249e5182a9e4c62dd781898f17fb21b5
./helpers/interfaces/IFijaSwapper.sol	289c7778db5dbcf37965d8cefb0c133c
./base/FijaERC4626Base.sol	e4d5fdacc041df8fe5cebf46e731e409
./base/errors.sol	ece0148c7f7455515f9e6e5a29535269
./base/FijaStrategy.sol	40d453c79b5e61f44f619509c9ba7d37
./base/FijaACL.sol	34b19fa1e2d356bcf0fbc1d75cf38a37
./base/FijaStrategyEvents.sol	fb0c945f1fe90c365f6c1e4bbc157489
./base/FijaStrategy2Txn.sol	fa5d958c9e66aeeaf7403e62ecc98d8e
./base/FijaVault2Txn.sol	d54cd3ec70f7d00fbba26de9d449e997
./base/FijaVault.sol	66c9d576818708c6f7d86dbbe1cbccc5
./base/types.sol	9669faa99cca8366c2a685ac9923634e
./base/ERC4626.sol	71ec59b71ef5a54febc355934aee0fe9
./interfaces/IFijaStrategy.sol	af01cdce0f74c8c981d3a483abe00bd7
./interfaces/IFijaStrategy2Txn.sol	18ba7b3fe23c8f3f5f0722f92a87f36a
./interfaces/IFijaACL.sol	63823fe906498591f34f0489b8d9a39e
./interfaces/IFijaERC4626Base.sol	6af43b2c33b5a3d8724d7b7b582e10a1
./interfaces/IFijaVault.sol	7ba7e3dce24e45e11bbca490c3995fda
./interfaces/IERC4626.sol	39ce8a0072809b1170070fc1784b63e0
./interfaces/IFijaVault2Txn.sol	cf4036ae8e15c79820155907cff8d059

Updates Version (21.07.2024)

File	Fingerprint (MD5)
./contracts/strategies/GMXv2/GMXv2Helpers.sol	11935e86e305a35d70a5b2418ee3ddc3
./contracts/strategies/GMXv2/errors.sol	64b4bdd1aa7199ecce8092f8455ca7bb
./contracts/strategies/GMXv2/GMXv2strategy.sol	f0a4bdf8ae741d553ee26312c435a403
./contracts/strategies/GMXv2/GMXv2Lib2.sol	6dca46c204ae3befb09d3f64f0229d4
./contracts/strategies/GMXv2/GMXv2strategyReader.sol	c2259f97dd60ba8cbe8d74a71ab2ef26
./contracts/strategies/GMXv2/IGMXv2strategy.sol	6c5f657f06c66ce7acd1d12486e5b51f
./contracts/strategies/GMXv2/GMXv2strategyBase.sol	5c4036bcbaf3ed6a95f4fd985fd29df7
./contracts/strategies/GMXv2/GMXv2Lib.sol	f11abc4193cdc74c23fc74954a42b9ca
./contracts/strategies/GMXv2/GMXv2Keys.sol	371f206d63060d558472c959855172b0
./contracts/strategies/GMXv2/GMXv2Type.sol	8aba0bd203555963e6a9e6eaf750e3e5
./contracts/strategies/GMXv2/IGMXv2strategyReader.sol	bfe53543ef65f007e257fdcf04e38ba8
./contracts/strategies/CurveConvex/errors.sol	35d125bba091f64bb3c0fd08da810f09
./contracts/strategies/CurveConvex/CurveConvexStrategy.sol	b56166b1a3e76b2557bdae2c67960510
./contracts/strategies/CurveConvex/CurveConvexStrategyProtocol.sol	970660ff01910d2ba688857f225cfe50
./contracts/strategies/CurveConvex/CurveConvexPeriphery.sol	c9521973bc9559b5cd2d7f219bd719ce
./contracts/strategies/CurveConvex/ICurveConvexPeriphery.sol	c572959fbd23e9ad809986b2aee1fefb
./contracts/strategies/CurveConvex/CurveConvexBase.sol	be2c9cfe11f3f0dacb1b3bd49c005b36
./contracts/protocols/gmx/interfaces/IGmxVaultUtils.sol	7c3775a8ce90ecc7f8c55cc12466d1a8
./contracts/protocols/gmx/interfaces/IGmxVault.sol	73b6b33ada9a765583c38fcebada360e8
./contracts/protocols/gmx/interfaces/IGmxRewardRouter.sol	b3c310d0005431224472a00b4c92f569



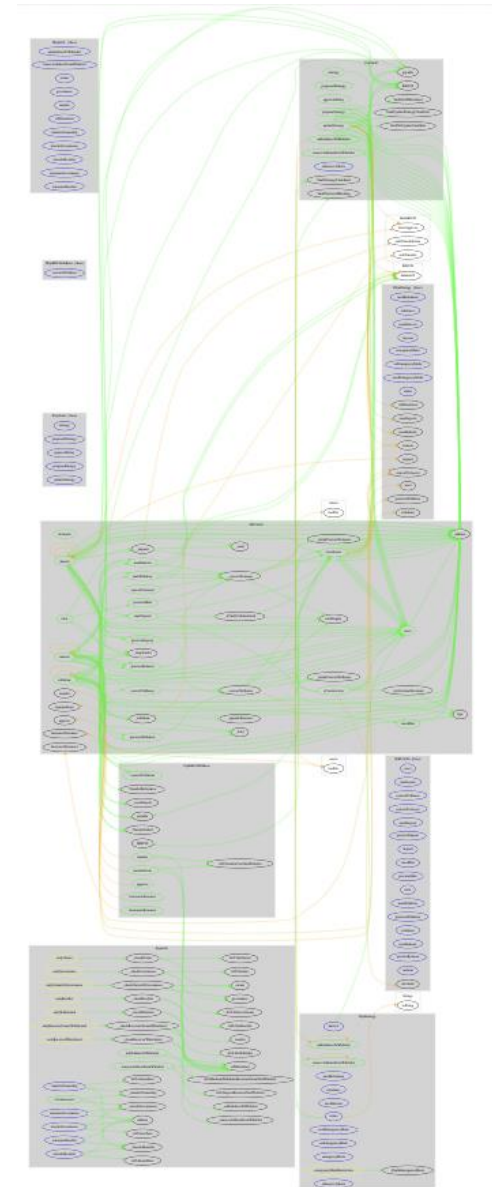
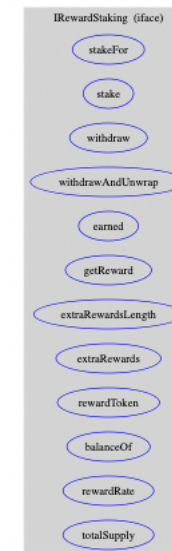
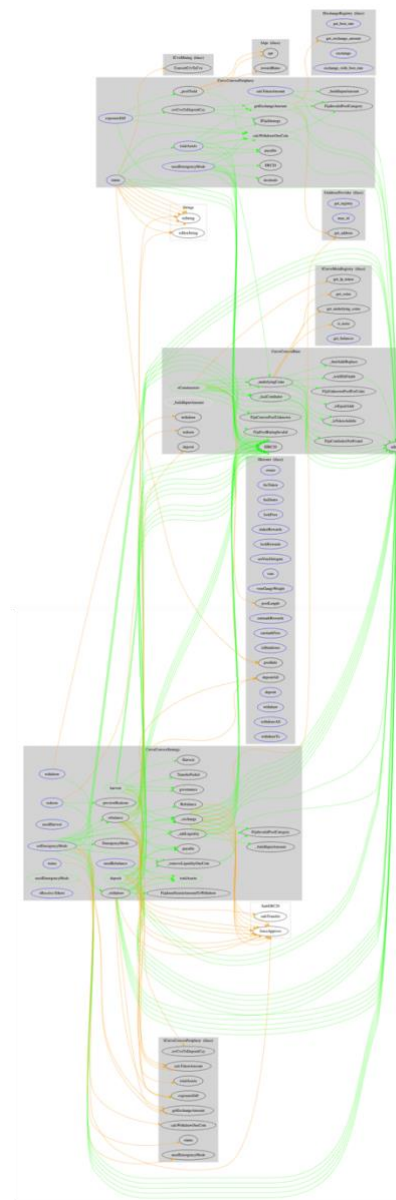
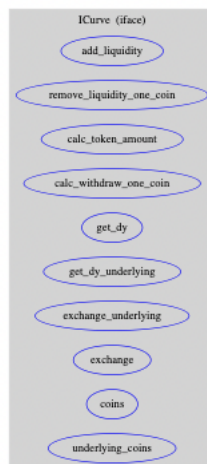
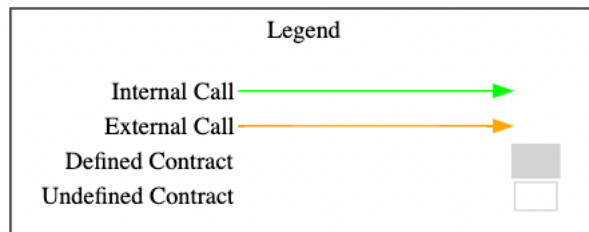
./contracts/protocols/gmx/interfaces/IGmxGlpManager.sol	79b6204a21ed854422e92ecfd49ba0c3
./contracts/protocols/gmx/interfaces/IGmxGlpRewardRouter.sol	a10aeca944312dc45f25d2bfa6c5a39b
./contracts/protocols/gmxv2/EventUtils.sol	b5bc054cf4736c43cebf3e5e8f6bf5d
./contracts/protocols/gmxv2/IWithdrawHandler.sol	1efa83619e1635d0155faff98b112bae
./contracts/protocols/gmxv2/IDatastore.sol	0133d73fa723a9ed2caa61581ca00df2
./contracts/protocols/gmxv2/Deposit.sol	9742160e393f15b6879329e2f543ad24
./contracts/protocols/gmxv2/IRoleStore.sol	fdd149fce0e3df25088f2448d6d5280c
./contracts/protocols/gmxv2/IDepositHandler.sol	a9987d83e3ef3f1190df89674649c893
./contracts/protocols/gmxv2/IReader.sol	66aba50731e63ccdd93ead70bd4c899a
./contracts/protocols/gmxv2/Withdrawal.sol	7e82f4201326d0160985d341405d0a87
./contracts/protocols/gmxv2/Price.sol	8a52f318cad57af698ae20fc3b046a2d
./contracts/protocols/gmxv2/Precision.sol	8886c3040951a91086d580c1050e499d
./contracts/protocols/gmxv2/IExchangeRouter.sol	511a9a3e48b0f60437889e9a5783ba72
./contracts/protocols/gmxv2/Market.sol	f4c5ec3d1d8fb0480831b9304f937e02
./contracts/protocols/curve/interfaces/ICurveMetaRegistry.sol	26063905ace52c138b18ad45e8223b27
./contracts/protocols/curve/interfaces/IExchangeRegistry.sol	a38c2181ebeadab79d24cf2ab0b9ce5d
./contracts/protocols/curve/interfaces/IAddressProvider.sol	20a47054c2b6d0d7ce60ad68b476e620
./contracts/protocols/curve/interfaces/ICurve.sol	f60e3b58c3d33dc0857a071a670d3f03
./contracts/protocols/convex/interfaces/IBooster.sol	eb98dce11b3916fecbe40eabba3a3b36
./contracts/protocols/convex/interfaces/ICvxMining.sol	bef1cd797cc24f582e4078c89a6d42a6
./contracts/protocols/convex/interfaces/IRewardStaking.sol	d04f05147b14476c69a9893209762efe
./contracts/protocols/convex/interfaces/IApr.sol	0d0805c573adeec18f6690f0d905d162
./contracts/helpers/FijaSwapper.sol	249e5182a9e4c62dd781898f17fb21b5
./contracts/helpers/interfaces/IFijaSwapper.sol	289c7778db5dbcfc7965d8cefb0c133c
./contracts/base/FijaERC4626Base.sol	d9381dc732934a1c86cbc73270d22ce9



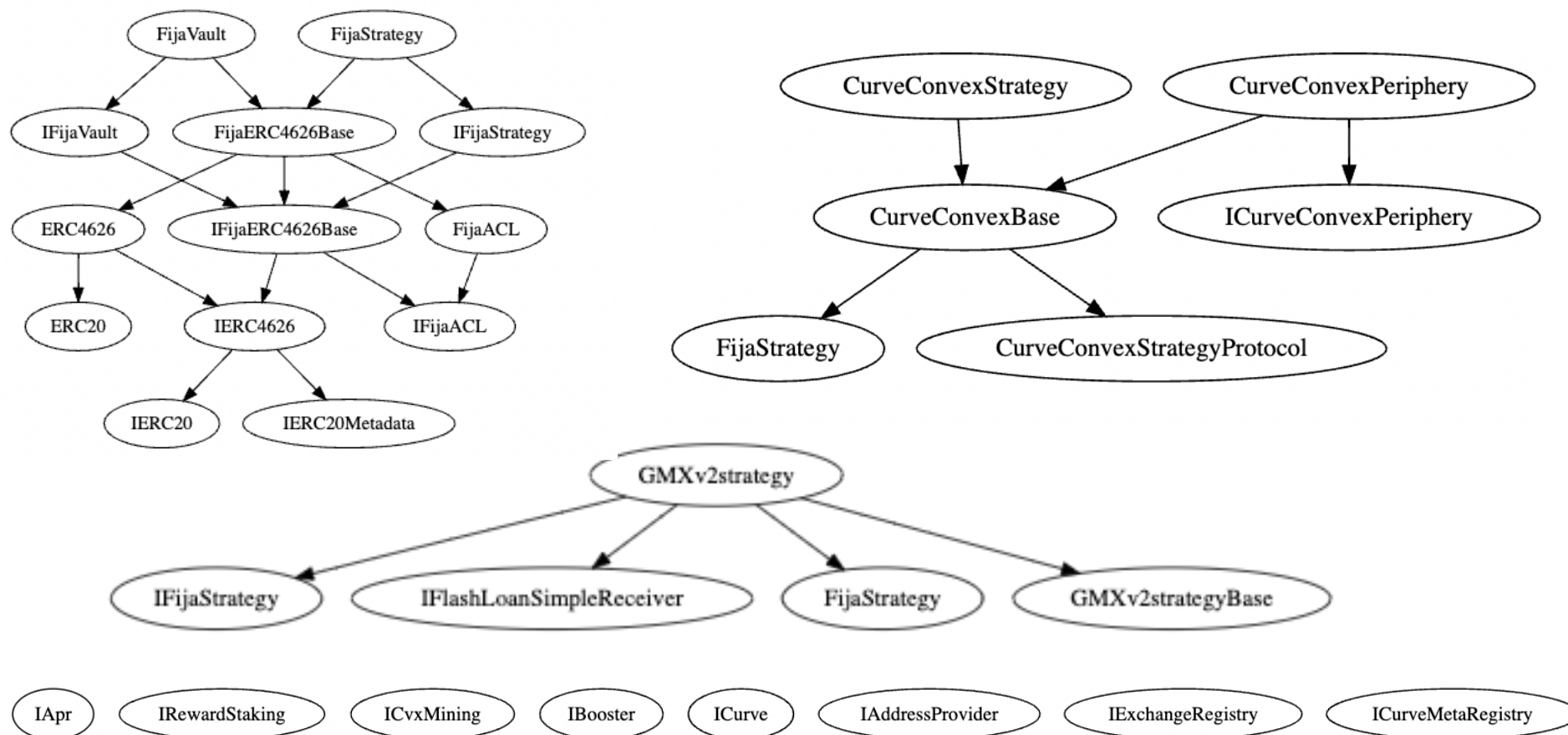
./contracts/base/errors.sol	b31b3038c4c30e8131bd42bf2f3e7160
./contracts/base/FijaStrategy.sol	40d453c79b5e61f44f619509c9ba7d37
./contracts/base/FijaACL.sol	34b19fa1e2d356bcf0fbc1d75cf38a37
./contracts/base/FijaStrategyEvents.sol	fb0c945f1fe90c365f6c1e4bbc157489
./contracts/base/FijaStrategy2Txn.sol	fa5d958c9e66aeeaf7403e62ecc98d8e
./contracts/base/FijaVault2Txn.sol	00ec402175722debc30edbea0453caf3
./contracts/base/FijaVault.sol	66c9d576818708c6f7d86dbbe1cbccc5
./contracts/base/types.sol	9669faa99cca8366c2a685ac9923634e
./contracts/base/ERC4626.sol	71ec59b71ef5a54febc355934aee0fe9
./contracts/interfaces/IFijaStrategy.sol	af01cdce0f74c8c981d3a483abe00bd7
./contracts/interfaces/IFijaStrategy2Txn.sol	63c351f476ef0f6103c4dde7604e31db
./contracts/interfaces/IFijaACL.sol	63823fe906498591f34f0489b8d9a39e
./contracts/interfaces/IFijaERC4626Base.sol	6af43b2c33b5a3d8724d7b7b582e10a1
./contracts/interfaces/IFijaVault.sol	7ba7e3dce24e45e11bbca490c3995fda
./contracts/interfaces/IERC4626.sol	39ce8a0072809b1170070fc1784b63e0
./contracts/interfaces/IFijaVault2Txn.sol	cf4036ae8e15c79820155907cff8d059



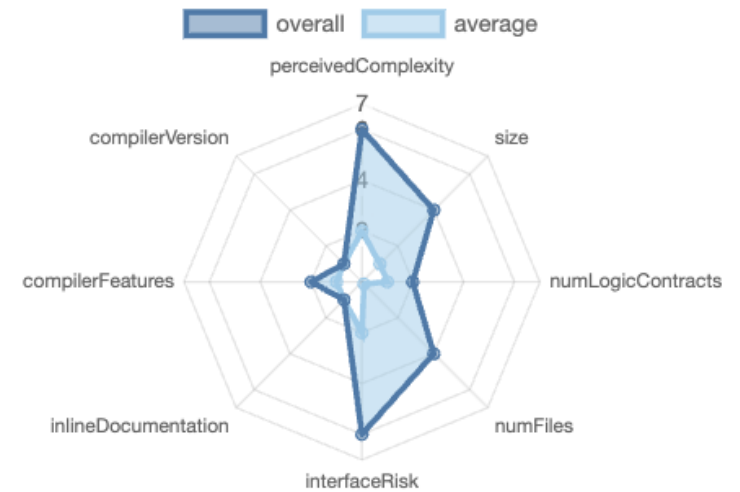
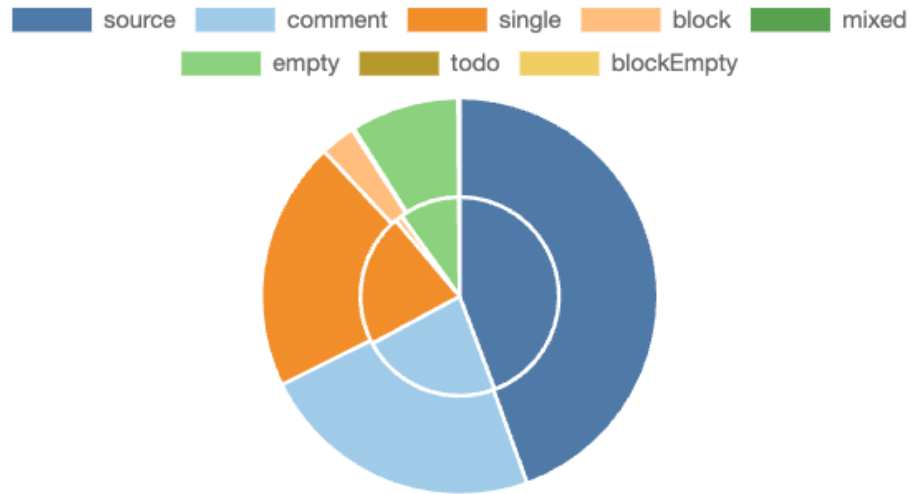
## 5.2 CallGraph



## 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
0.8.10			yes		
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECTRecover	 New/Create/Create2
yes					yes → NewContract:CurveConvexPeriphery

### 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				
111	13				
External	Internal	Private	Pure	View	
103	40	9	6	70	

### StateVariables














Total	 Public
39	0















## 5.6 Dependencies / External imports

Dependency / Import Path	Source
@openzeppelin/contracts/token/ERC20/ERC20.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/token/ERC20/ERC20.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/token/ERC20/ERC20.sol</a>
@openzeppelin/contracts/token/ERC20/IERC20.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/token/ERC20/IERC20.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/token/ERC20/IERC20.sol</a>
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/token/ERC20/extensions/IERC20Metadata.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/token/ERC20/extensions/IERC20Metadata.sol</a>
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/token/ERC20/utils/SafeERC20.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/token/ERC20/utils/SafeERC20.sol</a>
@openzeppelin/contracts/utils/Strings.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/utils/Strings.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/utils/Strings.sol</a>
@openzeppelin/contracts/utils/math/Math.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/utils/math/Math.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/utils/math/Math.sol</a>

## 5.7 Source Unites in Scope

Type	File	Logic Contract s	Interface s	Lin es	nLi nes	nSL OC	Com ment Lines	Comple x. Score	Capabili ties
	contracts/protocols/convex/interfaces/IApr.sol		1	14	5	3	1	5	
	contracts/protocols/convex/interfaces/IRewardStaking.sol		1	30	5	3	1	27	
	contracts/protocols/convex/interfaces/ICvxMining.sol		1	6	5	3	1	3	
	contracts/protocols/convex/interfaces/IBooster.sol		1	69	5	3	1	39	
	contracts/protocols/curve/interfaces/ICurve.sol		1	264	8	3	43	103	
	contracts/protocols/curve/interfaces/IAddressProvider.sol		1	10	5	3	1	7	
	contracts/protocols/curve/interfaces/IExchangeRegistry.sol		1	34	5	3	1	12	
	contracts/protocols/curve/interfaces/ICurveMetaRegistry.sol		1	20	5	3	1	11	
	contracts/strategies/CurveConvex/CurveConvexBase.sol	1		574	553	310	182	164	
	contracts/strategies/CurveConvex/ICurveConvexPeriphery.sol		1	91	12	3	52	17	

Type	File	Logic Contract s	Interface s	Lin es	nLi nes	nSL OC	Com ment Lines	Comple x. Score	Capabili ties
	contracts/strategies/CurveConvex/CurveConvexPeriphery.sol	1		593	552	447	60	263	
	contracts/strategies/CurveConvex/CurveConvexStrategyProtocol.sol	1		69	69	29	27	23	
	contracts/strategies/CurveConvex/CurveConvexStrategy.sol	1		988	941	718	151	381	 
	contracts/strategies/CurveConvex/errors.sol			10	10	8	1		
  	Totals	4	9	2772	2180	1539	523	1055	  

- **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 Flja Team provided us with the files that needs to be tested. The scope of the audit is the curve convex strategy and vault contracts.

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

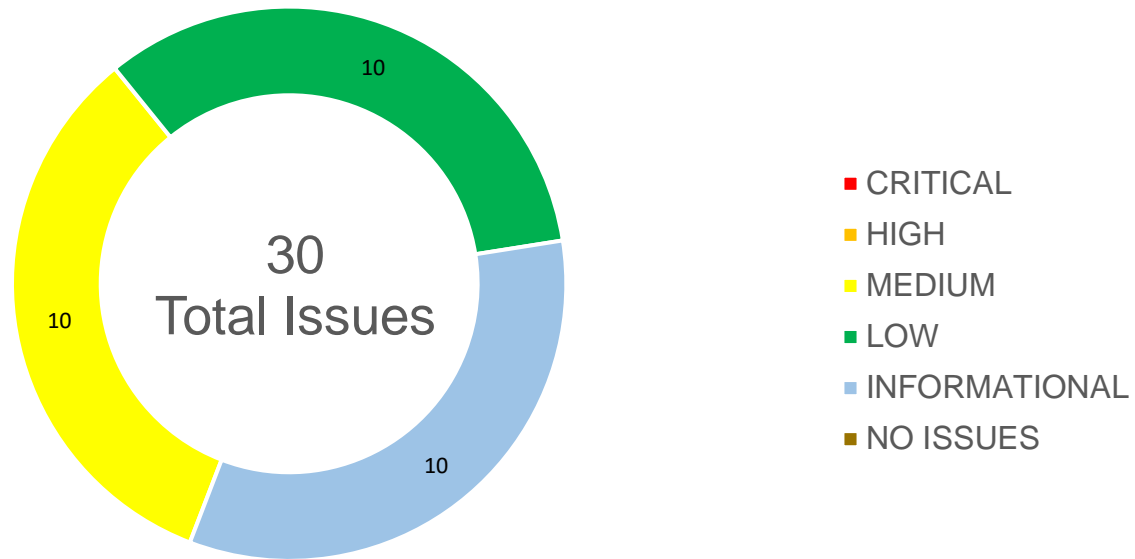
1. The audit should thoroughly investigate the curve convex, vault, GMXv2 contracts to confirm the absence of any vulnerabilities or potential exploits.
2. The audit should rigorously examine the contract's methods that deal with array manipulations to ensure that there are no vulnerabilities that can lead to unintended modifications or exploitable behaviors.
3. The audit must validate the functionality that allows for the replacement of token addresses, ensuring it accurately locates and substitutes the desired addresses without discrepancies, preventing any wrongful replacements.
4. A comprehensive review should be conducted on the contract's mechanisms to build and input amounts in specific array positions. This will ensure that tokens are correctly allocated without any inadvertent impacts or omissions.
5. The audit should rigorously assess the contract's interaction with external Curve, Convex or GMXv2 protocols, ensuring that any calls or references to these platforms are secure, reliable, and resistant to potential vulnerabilities like reentrancy attacks or unexpected behavior from external dependencies.
6. The security audit should thoroughly investigate the Vault smart contract to confirm the absence of any vulnerabilities or potential exploits.
7. The audit must validate the accuracy of token storage and transfer functions, ensuring that these processes are handled correctly and securely within the vault.
8. The correct functioning of deposit and withdrawal mechanisms must be tested to confirm that users' assets can be securely and efficiently moved in and out of the vault.
9. The effectiveness of the implemented yield strategies should be verified to ensure that they are operating as intended and generating expected returns.
10. The robustness of whitelisting protocols and associated security measures needs to be scrutinized to confirm their effectiveness in preventing unauthorized access and transactions.

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	Inconsistent Withdrawal Parameters Due to Post-Calculation Update of GMtokens	MEDIUM	FIXED
6.2.2	Potential Risk in Handling Large Redemption, When Requests Exceeding Available Balance	MEDIUM	FIXED
6.2.3	Potential Risk in Handling Large Withdrawal Requests, When Exceeding Available Balance	MEDIUM	FIXED
6.2.4	Lack of Flash Loan Callback Verification in _executeFlashloan Function of GMXv2Lib Library	MEDIUM	FIXED
6.2.5	Potential Hash Collision in status Function Due to abi.encodePacked() Usage	MEDIUM	ACKNOWLEDGED
6.2.6	Precision Loss Due to Division Before Multiplication in GMXv2strategyReader Contract	MEDIUM	ACKNOWLEDGED
6.2.7	Insufficient Liquidity Handling in Withdrawal and Redemption Functions of FijaVault2Txn contract	MEDIUM	ACKNOWLEDGED
6.2.8	Vault Deposits Can Be Front-run	MEDIUM	ACKNOWLEDGED
6.2.9	Unspecified Size Of Uint Variables	MEDIUM	FIXED
6.2.10	Improper Access Control	MEDIUM	FIXED
6.2.11	Prevalence of Magic Numbers in Codebase Compromises Maintainability	LOW	FIXED
6.2.12	Missing Zero Address Validation in GMXv2strategyReader Constructor	LOW	FIXED
6.2.13	Missing Event Emission in setContract Function of GMXv2strategyReader Contract	LOW	FIXED
6.2.14	Multiple Reads of Storage Variables in GMXv2strategyReader Contract	LOW	ACKNOWLEDGED
6.2.15	Update Required for Math Library Usage	LOW	ACKNOWLEDGED
6.2.16	Unspecified Variable Visibility	LOW	FIXED
6.2.17	Unclear Variable Value	LOW	FIXED
6.2.18	Improper Input Validation	LOW	FIXED

6.2.19	No Check for Contract Existence at Address	LOW	ACKNOWLEDGED
6.2.20	Missing Checks For Zero Address In Whitelisting Operations	LOW	FIXED
6.2.21	Potential Manipulation of Execution Fee Calculation in ETH Deposits	INFORMATIONAL	FIXED
6.2.22	Redundant Check for Zero Assets in Deposit Function Despite Modifier	INFORMATIONAL	ACKNOWLEDGED
6.2.23	Redundancy in Asset Calculation redeem Function	INFORMATIONAL	FIXED
6.2.24	Missing Struct Packing	INFORMATIONAL	FIXED
6.2.25	Magic Numbers	INFORMATIONAL	FIXED
6.2.26	Misleading Variable Name	INFORMATIONAL	FIXED
6.2.27	Missing Immutable Modifier	INFORMATIONAL	ACKNOWLEDGED
6.2.28	Redundant Payable Conversion	INFORMATIONAL	FIXED
6.2.29	Unused Return Value	INFORMATIONAL	FIXED
6.2.30	Missing Verification of Asset Addresses	INFORMATIONAL	ACKNOWLEDGED

## 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 **no High issues** in the code of the smart contract.

### MEDIUM ISSUES

During the audit, softstack's experts found **10 Medium issues** in the code of the smart contract.

#### 6.2.1 Inconsistent Withdrawal Parameters Due to Post-Calculation Update of GMtokens



Severity: MEDIUM

Status: FIXED

Code: CWE-474

File(s) affected: GMXv2Lib.sol

Update: Update to GMTokens to ensure we don't withdraw more than we have is moved before the calculation of minLongTokenAmount and minShortTokenAmount.

Attack / Description	In the genericInvestmentLogic function, the variables minLongTokenAmount and minShortTokenAmount are calculated based on the value of GMTokens. There is a conditional block later in the function that may update GMTokens based on available token balances. If this update occurs, minLongTokenAmount and minShortTokenAmount become inconsistent with GMTokens, as they are not recalculated. This inconsistency can lead to incorrect withdrawal parameters being used in the transaction, potentially affecting the intended operation of the function.
Code	<pre>Line 65 – 160 (GMXv2Lib.sol) // calculate GM token amount to withdraw and unwind to Tlong uint256 GMTokensTemp = ((c.eGsellShort * c.rShort) /     GMXv2Keys.AAVE_BASE_CURRENCY_PRECISION);  GMTokensTemp = GMXv2Helpers.adjustForDecimals(     GMTokensTemp,     strategyData.SEC_CCY_DECIMALS,     strategyData.DEP_CCY_DECIMALS ); GMTokens =     (uint256(-p.Tlong) *         (10 ** strategyData.DEP_CCY_DECIMALS)) /     (c.eGsellLong + GMTokensTemp);</pre>

```

GMtokens = GMXv2Helpers.adjustForDecimals(
    GMtokens,
    strategyData.DEP_CCY_DECIMALS,
    GMXv2Keys.GM_TOKEN_DECIMALS
);

// TODO do we include slippage here and for what exactly for uniswap or gmx swap? or both
GMtokens =
    (GMtokens *
        (GMXv2Keys.BASIS_POINTS_DIVISOR + GMXv2Keys.SLIPPAGE)) /
    GMXv2Keys.BASIS_POINTS_DIVISOR;

uint256 minLongTokenAmount;
uint256 minShortTokenAmount;
{
    // calculate market components we should receive when we burn GM tokens
    // deposit ccy + secondary ccy
    minLongTokenAmount =
        (GMtokens * c.eGsellLong) /
        GMXv2Keys.GM_TOKEN_PRECISION;
    minShortTokenAmount =
        (GMtokens * c.eGsellShort) /
        GMXv2Keys.GM_TOKEN_PRECISION;
}

```

```
Market.Props memory market = IReader(
    contracts[GMXv2Keys.READER]
).getMarket(
    contracts[GMXv2Keys.DATASTORE],
    strategyData.GM_POOL
);

if (market.longToken != strategyData.DEPOSIT_CCY) {
    uint256 temp = minLongTokenAmount;
    minLongTokenAmount = minShortTokenAmount;
    minShortTokenAmount = temp;
}
}

// create withdraw order
withdrawReqParams = Withdrawal.CreateWithdrawalParams({
    receiver: address(this),
    callbackContract: address(this),
    uiFeeReceiver: address(0),
    market: strategyData.GM_POOL,
    longTokenSwapPath: new address[](0),
    shortTokenSwapPath: new address[](0),
    minLongTokenAmount: minLongTokenAmount,
    minShortTokenAmount: minShortTokenAmount,
    shouldUnwrapNativeToken: false,
    executionFee: investLogicData.executionFee,
```

```

        callbackGasLimit: IGMXv2strategyReader(
            strategyData.PERIPHERY
        ).callbackGasLimit(
            investLogicData
                .txParamsWrapper
                .strategyParams
                .txType
        )
    });
}
IExchangeRouter exchangeRouter = IExchangeRouter(
    contracts[GMXv2Keys.EXCHANGE_ROUTER]
);
address withdrawVault = contracts[GMXv2Keys.WITHDRAW_VAULT];
{
    // send execution fee to GMX vault
    exchangeRouter.sendWnt{value: investLogicData.executionFee}(
        withdrawVault,
        investLogicData.executionFee
    );

    SafeERC20.forceApprove(
        IERC20(strategyData.GM_POOL),
        contracts[GMXv2Keys.ROUTER],
        GMtokens
    );
}

```

	<pre> ); uint256 gmTokensLeft = IERC20(strategyData.GM_POOL).balanceOf(     address(this) ); if (gmTokensLeft &lt; GMtokens) {     GMtokens = gmTokensLeft; } } </pre>
<b>Result/Recommendation</b>	<p>To ensure that the withdrawal parameters remain consistent with the value of GMtokens, it is recommended to either:</p> <ol style="list-style-type: none"> <li>1. Move the calculation of minLongTokenAmount and minShortTokenAmount to immediately before the creation of the withdrawReqParams object after any possible updates to GMtokens.</li> <li>2. Encapsulate the recalculations within a function that is called both after the initial calculation and after any potential updates to GMtokens.</li> </ol> <pre> // Initial calculation and potential update of GMtokens {     // [Initial calculation of GMtokens]     ...     // Check and update GMtokens if needed     uint256 gmTokensLeft = IERC20(strategyData.GM_POOL).balanceOf(address(this));      if (gmTokensLeft &lt; GMtokens) {         GMtokens = gmTokensLeft;     } } </pre>



```

    }

    // Recalculate minLongTokenAmount and minShortTokenAmount after finalizing GMtokens

    uint256 minLongTokenAmount = (GMtokens * c.eGsellLong) /
    GMXv2Keys.GM_TOKEN_PRECISION;
    uint256 minShortTokenAmount = (GMtokens * c.eGsellShort) /
    GMXv2Keys.GM_TOKEN_PRECISION;

    // Now create the withdraw request with the updated parameters
    withdrawReqParams = Withdrawal.CreateWithdrawalParams({
        ...
        minLongTokenAmount: minLongTokenAmount,
        minShortTokenAmount: minShortTokenAmount,
        ...
    });
}

```

### 6.2.2 Potential Risk in Handling Large Redemption, When Requests Exceeding Available Balance

Severity: MEDIUM

Status: FIXED

Code: NA

File(s) affected: FijaVault2Txn.sol

Update: A check is added to ensure we don't start redeeming from the strategy if the owner does not have the needed amount of tokens available. A VaultMaxRedeemExceeded error is thrown if the check fails.

#### Attack / Description

The redeem function within the contract attempts to handle redemption requests where the requested assets exceed the currentBalanceAvailable. The function uses the previewWithdraw method from a strategy contract to determine how many strategy tokens to withdraw to meet the shortfall. However, this approach does not account for scenarios where the requested assets are

	significantly greater than the currentBalanceAvailable, potentially leading to operational risks or failure in executing large redemptions.
Code	<p>Line 256 – 275 (FijaVault2Txn.sol)</p> <pre> if (assets &lt;= currentBalanceAvailable) {     assetsToReturn = FijaERC4626Base.redeem(tokens, receiver, owner); } else {     uint256 strategyTokens = _strategy.previewWithdraw(         assets - currentBalanceAvailable     );      // store vault.deposit params for afterRedeem callback called by strategy in 2 tx     _txParams = TxParams(0, tokens, receiver, owner, TxType.REDEEM);      _strategy.redeem{value: msg.value}(         strategyTokens,         address(this),         address(this)     );      _txParams = TxParams(0, 0, address(0), address(0), TxType.REDEEM);      assetsToReturn = 0; } </pre>

<b>Result/Recommendation</b>	<p>To improve the robustness of the redeem function and prevent potential liquidity issues, it is recommended to implement a check for maximum redeemable amounts based on the strategy's liquidity capacity.</p> <pre> if (assets &lt;= currentBalanceAvailable) {     assetsToReturn = FijaERC4626Base.redeem(tokens, receiver, owner); } else {     uint256 strategyTokensNeeded = assets - currentBalanceAvailable;     require(strategyTokensNeeded &lt;= currentBalanceAvailable, "Strategy cannot fulfill the requested token amount");     uint256 strategyTokens = _strategy.previewWithdraw(strategyTokensNeeded);      _strategy.redeem{value: msg.value}(         strategyTokens,         address(this),         address(this)     );      assetsToReturn = 0; } </pre>
------------------------------	---

### 6.2.3 Potential Risk in Handling Large Withdrawal Requests, When Exceeding Available Balance

Severity: MEDIUM

Status: FIXED

Code: NA

File(s) affected: FijaVault2Txn.sol

Update: A check is added to ensure we don't start withdrawing from the strategy if the owner does not have the needed amount of tokens available. A VaultMaxWithdrawExceeded error is thrown if the check fails.

<b>Attack / Description</b>	<p>The withdraw function in the contract attempts to handle withdrawal requests where the requested assets exceed the currentBalanceAvailableToUser. The function uses the withdraw method from a</p>
-----------------------------	---

	<p>strategy contract to cover the shortfall when the available assets are insufficient. This approach does not account for scenarios where the requested assets are significantly greater than the currentBalanceAvailableToUser, potentially leading to operational risks or failure in executing large withdrawals.</p>
<b>Code</b>	<p>Line 330 – 344 (FijaVault2Txn.sol)</p> <pre> if (assets &lt;= currentBalanceAvailableToUser) {     tokens = FijaERC4626Base.withdraw(assets, receiver, owner); } else {     // store vault withdraw params for afterWithdraw callback     _txParams = TxParams(assets, 0, receiver, owner, TxType.WITHDRAW);      _strategy.withdraw{value: msg.value}(         assets - currentBalanceAvailableToUser,         address(this),         address(this)     );      _txParams = TxParams(0, 0, address(0), address(0), TxType.WITHDRAW);     tokens = 0; } </pre>
<b>Result/Recommendation</b>	<p>To improve the robustness of the withdraw function and prevent potential liquidity issues, it is recommended to implement a check for maximum withdrawable amounts based on the strategy's liquidity capacity. Additionally, consider introducing a system to manage large withdrawals more effectively, such as processing them over time or using a queuing mechanism.</p> <pre> if (assets &lt;= currentBalanceAvailableToUser) {     tokens = FijaERC4626Base.withdraw(assets, receiver, owner); } </pre>

```

} else {
    uint256 assetsNeeded = assets - currentBalanceAvailableToUser;
    require(assetsNeeded <= currentBalanceAvailableToUser, "Strategy cannot fulfill the
requested asset amount");

    // store vault withdraw params for afterWithdraw callback
    _txParams = TxParams(assets, 0, receiver, owner, TxType.WITHDRAW);

    _strategy.withdraw{value: msg.value}(
        strategyTokens,
        address(this),
        address(this)
    );

    _txParams = TxParams(0, 0, address(0), address(0), TxType.WITHDRAW);
    tokens = 0;
}

```

#### 6.2.4 Lack of Flash Loan Callback Verification in \_executeFlashloan Function of GMXv2Lib Library

Severity: MEDIUM

Status: FIXED

Code: CWE-234

File(s) affected: GMXv2Lib.sol

Update: A modifier selfAaveOnly(initiator) was added to the executeOperation method to prevent unauthorized access

##### Attack / Description

The function \_executeFlashloan is designed to request flash loans from the Aave Lending Pool, expecting a callback to executeOperation within the same contract. However, the implementation does not include a verification step to ensure that the callback invocation originates from the Aave Lending Pool's contract address. This oversight creates a vulnerability where an attacker could simulate the callback, leading to unauthorized execution of the executeOperation logic.

<b>Code</b>	The vulnerability is present in the <code>_executeFlashloan</code> function, where <code>pool.flashLoanSimple(...)</code> is called without subsequent checks in the <code>executeOperation</code> callback to verify the caller's identity.
<b>Result/Recommendation</b>	<p>Verify Caller in Callback: Implement a verification check within the <code>executeOperation</code> callback to ensure <code>msg.sender</code> matches the Aave Lending Pool's contract address. This can be done by comparing <code>msg.sender</code> with a stored, known, and trusted Aave Lending Pool address.</p> <pre>function executeOperation(     address asset,     uint256 amount,     uint256 premium,     address initiator,     bytes calldata params ) external returns (bool) {     require(msg.sender == address(pool), "Caller is not the Aave Lending Pool"); require(initiator == address(this), "Loan initiator is not this contract");     // Continue with logic }</pre> <p>Securely Store Aave Lending Pool Address: Ensure the Aave Lending Pool address used for comparison is securely stored and accurately represents the legitimate Aave Lending Pool. Consider setting this address during contract deployment or through a secure update mechanism.</p> <p>Use Strongly Typed Interface: Interact with the Aave Lending Pool using a strongly typed interface to further ensure correct contract interactions and prevent accidental calls to incorrect addresses.</p>

#### 6.2.5 Potential Hash Collision in status Function Due to `abi.encodePacked()` Usage

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: CWE-328

File(s) affected: `GMXv2strategyReader.sol`

Update: The status call is used to monitor the internal parameters of the strategy from our backend systems, and returns a delimited string with specific parameter values inside. We don't think the hash collision problem described is impacting us here. As an example this could be a possible output of the status call: `v=11900407058158337957|v_long=86968706789397`



56972|v\_short=3174687877539877762|v\_short\_ccy=0 |v\_gmx\_amount=0|v\_dep\_ccy=28848501678703223|c\_long=19187680075122766360006|c\_short=7345841 412039877762|h=1496885249693701968|i\_short=14920975859|imbalanceBps=62

Attack / Description	The function constructs two strings, str1 and str2, using abi.encodePacked() with various dynamic arguments derived from contract state variables and external calls. These strings are then concatenated again using abi.encodePacked(). This method of encoding is susceptible to hash collisions due to the lack of padding and type information, particularly when used with dynamic types like strings.
Code	Line 143 – 173 (GMXv2strategyReader.sol)  <pre>string memory str1 = string(     abi.encodePacked(         ...     ) )  string memory str2 = string(     abi.encodePacked(         ...     ) )</pre>
Result/Recommendation	Replace the use of abi.encodePacked() for concatenating dynamic types with abi.encode(), which includes padding and type information, ensuring that each set of inputs produces a unique output. If concatenation of strings is necessary, consider handling the concatenation at a higher level (e.g., in the application logic outside the smart contract) or using a secure hashing function like keccak256 to hash the concatenated result if uniqueness is required. There is also discussion of removing abi.encodePacked from future versions of Solidity (ethereum/solidity#11593), so using abi.encode now will ensure compatibility in the future.

## 6.2.6 Precision Loss Due to Division Before Multiplication in GMXv2strategyReader Contract

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: CWE-682



File(s) affected: GMXv2strategyReader.sol

Update: fija has done a thorough scan of the complete codebase, and we are confident no unwanted precision losses are happening. The codebase does indeed contain divisions before multiplications, but the order of these operations is required in order to prevent overflows during the calculations.

<b>Attack / Description</b>	Solidity's integer division truncates any remainder, potentially leading to precision loss when division is performed before multiplication. This issue is prevalent in several functions within the GMXv2strategyReader contract, where calculations related to financial operations could result in inaccurate outcomes due to premature division.
<b>Code</b>	<ol style="list-style-type: none"><li>1. Function <code>_denominatorTlongDeposit(int256)</code> (Lines 745-762): Division before multiplication in the calculation of <code>modifier1</code> and <code>modifier2</code>. Potential precision loss could affect the calculation of deposit-related operations.</li><li>2. Function <code>_numeratorTlongDeposit(int256,int256)</code> (Lines 704-738): Division before multiplication in the calculation of <code>temp</code> and subsequent operations. This could lead to inaccurate numerator calculations for deposit operations.</li><li>3. Function <code>imbalanceBps()</code> (Lines 900-941): Division before multiplication in the calculation of <code>imbalanceBpsTemp</code>. Could result in incorrect calculation of imbalance basis points, affecting financial assessments.</li><li>4. Function <code>_v_short()</code> (Lines 379-394): Division before multiplication in the calculation of <code>unwind</code>. This could inaccurately calculate the value on the short side, impacting financial reporting.</li><li>5. Function <code>_denominatorTlongWithdraw(int256)</code> (Lines 811-854): Division before multiplication in the calculation of <code>modifier1</code> and <code>modifier2</code>. Potential precision loss could affect the calculation of withdrawal-related operations.</li><li>6. Function <code>_numeratorTlongWithdraw(int256,int256)</code> (Lines 770-804): Division before multiplication in the calculation of <code>base</code> and <code>modifier2</code>. This could lead to inaccurate numerator calculations for withdrawal operations.</li></ol>
<b>Result/Recommendation</b>	To mitigate precision loss, it is recommended to reorder arithmetic operations to perform multiplication before division wherever possible. This approach minimizes the impact of Solidity's



	integer division truncation and ensures more accurate calculations. Review and adjust the affected calculations within the contract to adhere to this best practice.
--	--

6.2.7 Insufficient Liquidity Handling in Withdrawal and Redemption Functions of FijaVault2Txn contract

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: NA

File(s) affected: FijaVault2Txn.sol

Update: We have liquidity checks outside the smart contract where we limit the overall capacity of a certain strategy contract to ensure we don't take up too big a portion of any pool used by the contract -- a situation which could lead to liquidity issues upon withdrawal/redeem actions. Although the suggested actions in the audit report are valid suggestions, we decided to not implement them at this point as it would lead to a more complex codebase which is harder to understand and maintain.

Attack / Description	<p>Issue Identification: The contract's withdraw and redeem functions attempt to fulfill requests even when the requested amount surpasses the available balance (currentBalanceAvailableToUser for withdrawals and currentBalanceAvailable for redemptions). This approach disregards scenarios where the liquidity shortfall is substantial, risking transaction failures or operational inefficiencies.</p> <p>Operational Risk: Engaging in transactions without validating the contract's capacity to meet the requested amounts introduces a significant risk of failure. This risk is exacerbated in volatile market conditions or scenarios with substantial withdrawal or redemption demands, potentially destabilizing the contract's operations.</p> <p>Liquidity Management Oversight: The absence of a liquidity capacity check within both functions overlooks the necessity of ensuring the contract can sustain the requested operations without compromising its stability</p>
Code	<p>Line 295 – 348 (FijaVault2Txn.sol)</p> <pre>function withdraw(     uint256 assets,</pre>

```

    address receiver,
    address owner
)

public
payable
virtual
override(FijaVault, IERC4626)
onlyWhitelisted
nonZeroAmount/assets)
onlyReceiverOwnerWhitelisted(receiver, owner)
isUpdateStrategyInProgress
returns (uint256)
{
    uint256 tokens;

    // TODO IMPORTANT: increasing allowance on sToken from vault to strategy
    SafeERC20.forceApprove(
        IERC20(_strategy),
        address(_strategy),
        type(uint256).max
    );

    uint256 currentBalanceAvailableToUser;
    if (asset() == ETH) {
        _executionFee = msg.value;

```

```

// execution fee is reduced from current balance
currentBalanceAvailableToUser = address(this).balance - msg.value;
} else {
    currentBalanceAvailableToUser = IERC20(asset()).balanceOf(
        address(this)
    );
}

if (assets <= currentBalanceAvailableToUser) {
    tokens = FijaERC4626Base.withdraw(assets, receiver, owner);
} else {
    // store vault withdraw params for afterWithdraw callback
    _txParams = TxParams(assets, 0, receiver, owner, TxType.WITHDRAW);

    _strategy.withdraw{value: msg.value}(
        assets - currentBalanceAvailableToUser,
        address(this),
        address(this)
    );

    _txParams = TxParams(0, 0, address(0), address(0), TxType.WITHDRAW);
    tokens = 0;
}
_executionFee = 0;

```

```
return tokens;
```

```
}
```

Line 219 – 279 (FijaVault2Txn.sol)

```
function redeem(
```

```
    uint256 tokens,
```

```
    address receiver,
```

```
    address owner
```

```
)
```

```
    public
```

```
    payable
```

```
    virtual
```

```
    override(FijaVault, IERC4626)
```

```
    onlyWhitelisted
```

```
    nonZeroAmount(tokens)
```

```
    onlyReceiverOwnerWhitelisted(receiver, owner)
```

```
    isUpdateStrategyInProgress
```

```
    returns (uint256)
```

```
{
```

```
    uint256 assets;
```

```
    uint256 assetsToReturn;
```

```
// TODO IMPORTANT: increasing allowance on sToken from vault to strategy
```

```
SafeERC20.forceApprove(
```

```
    IERC20(_strategy),
```

```
    address(_strategy),
```

```

        type(uint256).max
    );

    uint256 currentBalanceAvailable;
    if (asset() == ETH) {
        _executionFee = msg.value;
        assets = previewRedeem(tokens);

        // execution fee is reduced from current balance
        currentBalanceAvailable = address(this).balance - msg.value;
    } else {
        assets = previewRedeem(tokens);
        currentBalanceAvailable = IERC20(asset()).balanceOf(address(this));
    }

    if (assets <= currentBalanceAvailable) {
        assetsToReturn = FijaERC4626Base.redeem(tokens, receiver, owner);
    } else {
        uint256 strategyTokens = _strategy.previewWithdraw(
            assets - currentBalanceAvailable
        );

        // store vault.deposit params for afterRedeem callback called by strategy in 2 tx
        _txParams = TxParams(0, tokens, receiver, owner, TxType.REDEEM);
    }
}

```

	<pre> _strategy.redeem{value: msg.value}{     strategyTokens,     address(this),     address(this) };  _txParams = TxParams(0, 0, address(0), address(0), TxType.REDEEM);  assetsToReturn = 0; } _executionFee = 0;  return assetsToReturn; } </pre>
<b>Result/Recommendation</b>	<p>Implement Liquidity Checks: Introduce a validation step to verify if the requested withdrawal or redemption amount is within the contract's liquidity capacity. This check should occur before attempting to process the request, ensuring the contract only proceeds with transactions it can fulfill.</p> <p>Strategic Liquidity Management: Consider mechanisms for managing liquidity more effectively, such as a queuing system for large requests or a dynamic adjustment of withdrawal/redemption limits based on current liquidity levels. This approach can help mitigate liquidity risks and enhance the contract's resilience.</p> <p>Enhanced Error Handling: Improve error handling to provide clear feedback when requests cannot be processed due to liquidity constraints. This can include reverting transactions with descriptive error messages or implementing alternative solutions for partially fulfilling requests.</p>

### 6.2.8 Vault Deposits Can Be Front-run

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: NA

File(s) affected: ERC4626

Update by the Project: We are aware of a possibility to front-run the (first) transactions to a ERC4626 vault as implemented by OpenZeppelin. We do not consider this to be a problem of high severity as: - All deposits need to be made by whitelisted addresses, i.e. only a whitelisted wallet could exploit the front-running attack. Only customers having a contractual relationship with fija whitelisted, and as a result we will always be able to link an attacker's wallet to a person/entity having a business relationship with fija. - As part of the deployment process for strategies and vaults fija always makes a small test deposit before making the vault or strategy available to customers. After the first deposit the front-running attack becomes economically unviable, and as a result customer funds are not considered to be at risk due to the front-run attack.

<b>Attack / Description</b>	The vault implementation is forked from an older version of OpenZeppelin's ERC4626 source code. This code has a known vulnerability called "inflation attack" where an attacker is able to steal user funds on the first deposits, by front-running the deposit transaction. A detailed description of this issue can be found in the report of OpenZeppelin from October 2022. <a href="https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/audits/2022-10-ERC4626.pdf">https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/audits/2022-10-ERC4626.pdf</a>
<b>Code</b>	NA
<b>Result/Recommendation</b>	To mitigate that risk, it is recommended to implement a mechanism such as virtual liquidity to increase the amount an attacker must afford for stealing user funds. Overall, it is recommended to use the newest version of the OpenZeppelin ERC4626 implementation, which is using virtual liquidity to mitigate the described risks.

### 6.2.9 Unspecified Size Of Uint Variables

Severity: MEDIUM

Status: FIXED



Code: NA

File(s) affected: FijaERC4626Base.sol, FijaStrategy.sol, FijaVault.sol

Update by the Project: Size of uint variables is now specified throughout the codebase.

<b>Attack / Description</b>	The current implementation contains many uint variable definitions without specifying the actual size of the variable. This can lead to suboptimal gas usage or unintended behavior. Solidity offers the option to define the actual byte size of each uint variable explicitly (i.e. uint256, uint8,...).
<b>Code</b>	<div>FijaERC4626Base Line 25: uint immutable MAX_TICKET_SIZE; Line 31: uint immutable MAX_VAULT_VALUE; Line 39: uint maxTicketSize_, Line 40: uint maxVaultValue_ Line 197: uint assets = _convertToAssets(balanceOf(owner), Math.Rounding.Down); Line 210: uint tokens = balanceOf(owner); Line 211: uint assets = _convertToAssets(tokens, Math.Rounding.Down); Line 373: uint maxValueDiff = MAX_VAULT_VALUE - totalAsset;</div> <div>FijaStrategy Line 25: uint maxTicketSize_, Line 26: uint maxVaultValue_</div> <div>FijaVault Line 36: uint maxTicketSize_, Line 37: uint maxVaultValue_</div>
<b>Result/Recommendation</b>	It is recommended to explicitly define the size of all variables in all smart contracts. Explicitly specifying the size of a unit variable in Solidity is crucial for clarity and accuracy in code. It helps preventing errors, optimizes gas usage, and enhances security. Clear variable sizes aid in code audits and adhere to coding standards. Additionally, it future-proofs the code against potential changes in default data types.



## 6.2.10 Improper Access Control

Severity: MEDIUM

Status: FIXED

Code: CWE-284

File(s) affected: FijaACL.sol

Update by the Project: Non-zero checks are added to the codebase to ensure addresses are not accidentally put to 0. Wallets are controlled by a high-security custody system. The owner, governance and reseller wallets used by the fija Vault and the fija Strategy are controlled by a custody system used by fija Finance. The custody system uses HSM technology to generate private keys, has multi-sig capabilities, and a fine-grained permission model to prevent possible abuse of these wallets.

<b>Attack / Description</b>	The current implementation of the smart contract centralizes a lot of control and governance in the owner and governance roles. This centralized control might be a risk factor, as the whole system's security and integrity heavily rely on these roles. This is especially concerning if there's a possibility that the owner's or governance's private keys are compromised.
<b>Code</b>	Line 18 – 20 (FijaACL.sol)  <pre>address private _owner; address private _governance; address private _reseller;</pre>
<b>Result/Recommendation</b>	<p>Add a require statement to check that the provided address is not the zero address in these functions.</p> <p>Consider implementing a more decentralized governance model, like a multi-signature scheme or a DAO, to decentralize control and reduce the risk associated with a single point of failure. For instance, require multiple approvals for critical operations like transferring ownership or governance.</p>

	<p>For multi-signature scheme (gnosis safe), you might want to create a contract that requires a certain number of approvals from authorized addresses before making significant changes.</p> <p>For a DAO model, you could use an external library or existing DAO frameworks, like Aragon or DAOstack, to handle governance. This way, the decision-making power and governance are distributed among a larger group of people, which reduces the risk of system failure due to compromised keys or malicious actions from a single party.</p>
--	--

# LOW ISSUES

During the audit, softstack’s experts found **10 Low issues** in the code of the smart contract

## 6.2.11 Prevalence of Magic Numbers in Codebase Compromises Maintainability

Severity: LOW

Status: FIXED

Code: NA

File(s) affected: GMXv2Lib.sol, GMXv2strategyReader.sol

Update: Occurences of 10\*\*18 were replaced by constant TOKEN\_PRECISION\_18 throughout the codebase. Other examples were not fixed.

<b>Attack / Description</b>	The codebase contains numerous instances where "magic numbers" are directly used within function logic. Magic numbers are hard-coded values that appear without clear explanation or definition, making the code less readable, harder to maintain, and more prone to errors. These numbers often represent configuration parameters, thresholds, or specific coefficients that control the behavior of the code. Using magic numbers directly in the logic without documenting or encapsulating them in named constants can lead to confusion about their purpose and impact, reducing the overall code quality and increasing the risk of mistakes during updates or refactoring.
<b>Code</b>	GMXv2Lib.sol Line: 869



	<pre> emit FijaStrategyEvents.Harvest( block.timestamp, 4 * profitShare, profitShare, IERC4626(address(this)).asset(), "" );  GMXv2strategyReader.sol Line: 277 return GMXv2Helpers.adjustForDecimals( fee, GMXv2Keys.UNISWAP_FEE_DECIMALS, 4 );  Line: 542 uint256 target = map.data &gt;&gt; 16;  Line: 543 uint256 mask = (1 &lt;&lt; 16) - 1;  Line: 735 modifier2 = (modifier2 * int256(_hDivQ())) / 10 ** 18;  Line: 754 int256 modifier1 = (( ((base * int256(_a_short())) / int256(GMXv2Keys.AAVE_BASE_CURRENCY_PRECISION)) * int256(_hDivQ())) / int256(10 ** 18);  Line: 801 modifier2 = (modifier2 * int256(_hDivQ())) / 10 ** 18;  Line: 851 modifier2 = (modifier2 * int256(_hDivQ())) / 10 ** 18; </pre>
<b>Result/Recommendation</b>	<p>To enhance the readability and maintainability of the code, it is recommended to replace all magic numbers with clearly named constants. This approach not only clarifies their purpose but also centralizes configuration, making the code easier to manage and modify.</p>

6.2.12 Missing Zero Address Validation in GMXv2strategyReader Constructor

Severity: LOW

Status: FIXED

Code: NA

File(s) affected: GMXv2strategyReader.sol

Update: Zero address checks were added to GMXv2StrategyReader constructor.

Attack / Description	The constructor of the GMXv2strategyReader contract takes two address parameters, coreStrategy_ and asset_, and assigns them to state variables without checking if they are the zero address. Initializing the contract with a zero address for these critical parameters could render the contract non-functional or vulnerable to attacks.
Code	<p>Asset Address Validation: Location: Line 77 (_asset = asset_;) Impact: Setting the asset address to the zero address could prevent the contract from performing operations related to the asset, such as transfers, balance checks, and other token-related functionalities.</p> <p>Core Strategy Address Validation: Location: Line 78 (CORE_STRATEGY = coreStrategy_;) Impact: Assigning a zero address to the core strategy could disable interactions with the core strategy contract</p>
Result/Recommendation	<p>Implement zero address checks for both coreStrategy_ and asset_ parameters within the constructor to ensure that the contract is initialized with valid addresses. This can be achieved using a require statement to assert that the addresses are not the zero address.</p> <pre>constructor(     ConstructorData memory data_,     address coreStrategy_,     address asset_  ) GMXv2strategyBase(data_, asset_) {     require(coreStrategy_ != address(0), "GMXv2strategyReader: coreStrategy_ is the zero address");     require(asset_ != address(0), "GMXv2strategyReader: asset_ is the zero address");</pre>



	<pre>         _asset = asset_;         CORE_STRATEGY = coreStrategy_;     } </pre>
--	--

### 6.2.13 Missing Event Emission in setContract Function of GMXv2strategyReader Contract

Severity: LOW

Status: FIXED

Code: NA

File(s) affected: GMXv2strategyReader.sol

Update: An event ContractUpdated was added.

<b>Attack / Description</b>	<p>The setContract function allows updating the address associated with a given key in the <code>_contracts</code> mapping. This operation is a critical state change, especially in a decentralized finance (DeFi) context, where contract addresses might represent significant aspects of the protocol, such as asset pools, strategy implementations, or other integrations. The lack of event emission after such updates makes it difficult for off-chain services, interfaces, and users to track changes and respond accordingly.</p>
<b>Code</b>	<p>Line 118 – 127 (GMXv2strategyReader.sol)</p> <pre> function setContract(string memory key, address value) external {     if (msg.sender != CORE_STRATEGY) {         revert ACLNotOwner();     }      // validate key exists     if (_contracts[keccak256(abi.encode(key))] == address(0)) {         revert FijaSetContractWrongKey();     } } </pre>

	<pre>     }     _contracts[keccak256(abi.encode(key))] = value;     } </pre>
<b>Result/Recommendation</b>	<p>Implement an event emission within the setContract function to log the key and new address every time a contract address is updated. Define an event, e.g., ContractUpdated, and emit it after the state change is applied.</p> <p>Example implementation</p> <pre> event ContractUpdated(string key, address indexed newAddress);  function setContract(string memory key, address value) external {     if (msg.sender != CORE_STRATEGY) {         revert ACLNotOwner();     }     // validate key exists     if (_contracts[keccak256(abi.encode(key))] == address(0)) {         revert FijaSetContractWrongKey();     }     _contracts[keccak256(abi.encode(key))] = value;     emit ContractUpdated(key, value); } </pre>

#### 6.2.14 Multiple Reads of Storage Variables in GMXv2strategyReader Contract

Severity: LOW

Status: ACKNOWLEDGED

Code: SWC-107

File(s) affected: GMXv2strategyReader.sol

<b>Attack / Description</b>	The Solidity compiler does not automatically cache storage reads to memory for subsequent accesses within the same function call. As a result, each read operation from a storage variable
-----------------------------	--

	<p>incurs the full gas cost associated with storage access. The identified functions read from the DEP_CCY_DECIMALS and SEC_CCY_DECIMALS storage variables multiple times, which could be optimized by caching these reads into local memory variables at the beginning of the function.</p> <ol style="list-style-type: none"> <li>1. Repeated Reads of DEP_CCY_DECIMALS and SEC_CCY_DECIMALS:</li> <li>2. In the _denominatorTlongWithdraw(int256) function, the DEP_CCY_DECIMALS and SEC_CCY_DECIMALS variables are read multiple times for various calculations.</li> <li>3. Similarly, in the _C_long(int256,int256) function, DEP_CCY_DECIMALS is read multiple times to perform calculations related to investment adjustments.</li> </ol>
<b>Code</b>	<p>Line 361 – 365 (GMXv2strategyReader.sol) Example:</p> <pre>convertedAmountOut = GMXv2Helpers.adjustForDecimals(     convertedAmountOut,     SEC_CCY_DECIMALS,     DEP_CCY_DECIMALS );</pre>
<b>Result/Recommendation</b>	<p>To mitigate this issue, it is recommended to read storage variables once at the beginning of the function and store their values in local memory variables. These local variables should then be used for all subsequent operations within the function.</p> <p>Example fix for _denominatorTlongWithdraw(int256):</p> <pre>function _denominatorTlongWithdraw(int256 e_sellShortModSecCcy) private view returns (int256) {     uint256 depCcyDecimals = DEP_CCY_DECIMALS; // Cache storage read     uint256 secCcyDecimals = SEC_CCY_DECIMALS; // Cache storage read      // Use depCcyDecimals and secCcyDecimals instead of direct storage access ...</pre>

	}
--	---

### 6.2.15 Update Required for Math Library Usage

Severity: LOW

Status: ACKNOWLEDGED

Code: NA

File(s) affected: ERC4626.sol, FijaERC4626Base.sol, FijaVault2Txn.sol, GMXv2strategy.sol

Update: The Math.sol library we are using is coming with the ERC4626 implementation from OpenZeppelin. We reviewed our code and all occurrences of Math. Rounding.Up and Math.Rounding.Down are in snippet of code that we have copied from the OZ implementation to make our contracts compatible with native ETH. Since it's easier to ensure the correctness of our code if the snippets are exactly the same as in the audited OZ base classes we opt to keep the current naming. We will be updating the ERC4626 base in the future, and we will align our codebase with the updated naming conventions at that time.

<b>Attack / Description</b>	As detailed in OpenZeppelin's pull request #4455 revises the Rounding enum within the Math library. The terms "up" and "down" have been replaced with more precise terminology: Floor (toward negative infinity), Ceil (toward positive infinity), Trunc (toward zero), and the newly introduced Expand (away from zero). This change addresses the potential confusion with the previous terms and adds a necessary rounding mode for signed numbers. Our project's current implementation utilizes the old version of the Math library, specifically employing Math.Rounding.Down in the deposit function for calculating the conversion to shares and other related operations.
<b>Code</b>	<p>Line 140 – 147 (FijaVault2Txn.sol)</p> <p>Example:</p> <pre>uint256 supply = totalSupply();  tokens = (assets == 0    supply == 0) ? _initialConvertToShares(assets, Math.Rounding.Down) : assets.mulDiv(     supply,     totalAssetBeforeDeposit,     Math.Rounding.Down</pre>



	);
<b>Result/Recommendation</b>	<p>Replace Math.Rounding.Down with the new corresponding Rounding enum member, likely Rounding.Floor, to maintain the intended behavior. Carefully review each usage to select the appropriate rounding mode based on the context.</p> <pre>// Before update uint256 tokens = assets.mulDiv(supply, totalAssetBeforeDeposit, Math.Rounding.Down);</pre> <pre>// After update uint256 tokens = assets.mulDiv(supply, totalAssetBeforeDeposit, Math.Rounding.Floor);</pre>

#### 6.2.16 Unspecified Variable Visibility

Severity: LOW

Status: FIXED

Code: NA

File(s) affected: CurveConvexBase.sol, CurveConvexPeriphery.sol, CurveConvexStrategy.sol, CurveConvexStrategyProtocol.sol, FijaERC4626Base.sol

Update by the Project: Explicit visibility is now specified for the variables.

<b>Attack / Description</b>	Many global variables are missing visibility modifiers. In Solidity, if you omit the visibility modifier for a state variable, it will default to "internal". This means that the variable can only be accessed within the current contract and derived contracts (i.e., contracts that inherit from the current one). It cannot be accessed from external sources like other contracts or external actors.
<b>Code</b>	<p>CurveConvexBase Line 99 – 225</p> <p>CurveConvexPeriphery Line 20: address immutable STRATEGY;</p> <p>CurveConvexStrategy Line 20: ICurveConvexPeriphery immutable PERIPHERY;</p> <p>CurveConvexStrategyProtocol Line 20 – 67</p>

	FijaERC4626Base Line 18 – 31 <pre>abstract contract FijaERC4626Base is IFijaERC4626Base, FijaACL, ERC4626 {     using Math for uint256;     address constant ETH = 0xEeeeeEeeeEeEeeEeEeEEEEEEEEEEEEEEEEEE;     uint immutable MAX_TICKET_SIZE;     uint immutable MAX_VAULT_VALUE;</pre>
<b>Result/Recommendation</b>	Explicitly setting variable visibility in Solidity is a best practice for clarity, security, and preventing unintended access. It makes code more readable, helps avoid bugs, and follows security principles. It also aids in code reviews and futureproofing.

#### 6.2.17 Unclear Variable Value

Severity: LOW

Status: FIXED

Code: NA

File(s) affected: FijaVault.sol

Update by the Project: Magic number was replaced by type(uint64).max

<b>Attack / Description</b>	The updateStrategy function, updates the underlying strategy, resets the values for strategy implementation address and propose time. However, the proposed time is set to a magic number value not indicating its purpose. Readers may not know if it is a point of time in the future, which it should be.
<b>Code</b>	Line 179: _strategyCandidate.proposedTime = 6000000000;
<b>Result/Recommendation</b>	It is recommended to use the maximum value of the specified unit to implicitly point out the reset of this variable to the highest possible number, which will never be reached. Therefore, type(uint256).max can be used to stress out that the highest possible value is used in a readable way.

## 6.2.18 Improper Input Validation

Severity: LOW

Status: FIXED

Code: CWE-20

File(s) affected: FijaVault.sol

Update by the Project: Code in the base class fijaERC4626Base.sol was updated to include a check on zero amounts.

<b>Attack / Description</b>	In the code of the smart contract, there is a lack of proper validation for the input parameters of the deposit, withdraw, and redeem functions. This absence of validation can result in unexpected behaviors or potential exploits within the contract. Specifically, it is possible to call these functions with a value of 0 as the amount parameter, which can lead to unexpected results and may create vulnerabilities in the contract's functionality.
<b>Code</b>	<p>Line 190 – 193 (FijaVault.sol)</p> <pre>function deposit(     uint256 assets,     address receiver ) public virtual override(FijaERC4626Base, IERC4626) returns (uint256) {</pre> <p>Line 219 – 223 (FijaVault.sol)</p> <pre>function redeem(     uint256 tokens,     address receiver,     address owner ) public virtual override(FijaERC4626Base, IERC4626) returns (uint256) {</pre> <p>Line 248 – 252 (FijaVault.sol)</p>

	<pre> function withdraw(     uint256 assets,     address receiver,     address owner ) public virtual override(FijaERC4626Base, IERC4626) returns (uint256) {     uint256 currentBalance = IERC20(asset()).balanceOf(address(this)); </pre>
<b>Result/Recommendation</b>	<p>To address this issue and enhance the security and integrity of the smart contract, it is recommended to add a condition that validates the non-zero nature of the input amounts for assets and tokens in the deposit, withdraw, and redeem functions. This validation can be implemented using a require statement to check if the amount is greater than zero. By adding the following code snippet to the respective functions, the contract will enforce a minimum value requirement for the amounts:</p> <pre>require(assets &gt; 0, "Deposit amount must be greater than zero");</pre> <p>This validation ensures that the functions are called with valid, non-zero amounts, reducing the risk of unexpected behavior arising from zero-value inputs.</p>

#### 6.2.19 No Check for Contract Existence at Address

Severity: LOW

Status: ACKNOWLEDGED

Code: NA

File(s) affected: FijaVault.sol

Update by the Project: We decided not to implement this change as: Even if we check for an existing contract we would still not be sure that it correctly implements the IfijaStrategy interface. In case of specifying an address without contract (or an incorrect contract)

the `strategyUpdate()` call will fail, signalling to the caller that something is wrong. Funds are not becoming inaccessible, even if specifying a wrong address: the `governance()` wallet can simply specify a new strategy contract and funds will be transferred correctly to the new strategy contract.

<b>Attack / Description</b>	The contract does not check whether a contract actually exists at the strategy address before calling its functions, which might lead to failed calls if there is no contract at the specified address. When an address of a non-existing contract is passed as the <code>strategyCandidate</code> to the <code>proposeStrategy</code> function, the call to <code>strategyCandidate.isWhitelisted</code> will fail.
<b>Code</b>	Line 95 (FijaVault.sol)  <pre>if (!strategyCandidate.isWhitelisted(address(this))) {</pre>
<b>Result/Recommendation</b>	<p>To mitigate this issue, it is recommended to implement a check to verify the existence of a contract at an arbitrary address before making any function calls. This can be achieved by checking the code size at the given address using the following helper function:</p> <pre>function isContract(address _addr) private view returns (bool) {     uint32 size;     assembly {         size := extcodesize(_addr)     }     return (size &gt; 0); }</pre> <p>It is advisable to incorporate this <code>isContract</code> function into the FijaVault contract. Before calling <code>strategyCandidate.isWhitelisted(address(this))</code> within the <code>proposeStrategy</code> function, invoke <code>isContract(strategyCandidate)</code> to ensure that a valid contract exists at the specified address. By performing this validation, you can prevent failed function calls due to the absence of a contract at the given address.</p>

## 6.2.20 Missing Checks For Zero Address In Whitelisting Operations

Severity: LOW

Status: FIXED

Code: NA

File(s) affected: FijaACL.sol

Update by the Project: Code will now revert in case of 0 address

Attack / Description	The <code>_addAddressToWhitelist</code> function does not check if the provided address is the zero address. It's generally a good practice to prevent the zero address from being added to the whitelist, as this address is often used to burn tokens.
Code	Line 89 (FijaACL.sol)  <pre>function addAddressToWhitelist(     address addr ) public virtual override returns (bool) {     if (isWhitelisted(addr)) {         return false;     } }</pre>
Result/Recommendation	Add a require statement to check that the provided address is not the zero address in these functions. The updated code could look like this:  <pre>function _addAddressToWhitelist(address addr) internal {     require(addr != address(0), "FijaACL: Cannot add zero address to whitelist");     ... }</pre>

# INFORMATIONAL ISSUES

During the audit, softstack’s experts found **10 Informational issues** in the code of the smart contract.

## 6.2.21 Potential Manipulation of Execution Fee Calculation in ETH Deposits

Severity: INFORMATIONAL

Status: FIXED

Code: CWE-682

File(s) affected: FijaVault2Txn.sol

Update: We adjusted the checks for the executionFee, and moved all checks to the GMXv2Strategy.sol (there was some duplication earlier where both the vault and the strategy were doing the same check). For all deposit/withdraw/redeem actions:

- A check is put in place to confirm that the executionFee is higher than the expected transaction cost within the current block. The check will return with a NotEnoughETHSent error in case of failure.
- We decided not to check for a maximum of the fee since any limit would be more or less arbitrary and could result in the 2nd GMX transaction not being executed within reasonable amount of time. This could happen e.g. because of changing gas prices between the 2 transactions. It is understood that the user may incur extra costs should he decide to submit a high execution fee.

Attack / Description	The smart contract's deposit function calculates the execution fee as the difference between msg.value and the amount of ETH specified as assets. This approach allows callers to manipulate the execution fee by varying the msg.value sent with the transaction. If msg.value is significantly higher than assets, it results in an unnecessarily high execution fee. Conversely, if msg.value is only slightly higher than assets, the execution fee might not cover the costs of transaction processing, potentially leading to operational deficits or exploitation.
Code	Line 127 - 138 (FijaVault2Txn.sol) <pre>if (asset() == ETH) {     if (msg.value &lt; assets) {         revert NotEnoughETHSent();     }     uint256 executionFee = msg.value - assets;</pre>



	<pre> _executionFee = executionFee;  uint256 totalAssetBeforeDeposit = totalAssets() - assets; require(     assets &lt;= _maxDeposit(receiver, totalAssetBeforeDeposit),     "ERC4626: deposit more than max" ); </pre>
<b>Result/Recommendation</b>	<p>To ensure a fair and predictable execution fee, it's recommended to implement a fixed or percentage-based fee structure. Below are proposed changes to enforce a minimum and potentially a maximum execution fee, thereby preventing manipulation and ensuring the costs are covered adequately.</p> <ol style="list-style-type: none"> <li>1. Define constants for minimum and maximum execution fees.</li> <li>2. Add checks to ensure the execution fee falls within these bounds.</li> <li>3. Modify the function to calculate the fee based on these specifications.</li> </ol> <pre> // Constants defined at the contract level uint256 public constant MIN_EXECUTION_FEE = 0.01 ether; uint256 public constant MAX_EXECUTION_FEE = 0.05 ether; function deposit(uint256 assets, address receiver) public payable virtual override {     require(msg.value &gt;= assets + MIN_EXECUTION_FEE, "Minimum fee not met");     uint256 executionFee = msg.value - assets; require(executionFee &lt;= MAX_EXECUTION_FEE, "Maximum fee exceeded");      // Additional logic remains unchanged ... } </pre>



6.2.22 Redundant Check for Zero Assets in Deposit Function Despite Modifier

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: NA

File(s) affected: FijaVault2Txn.sol

Update: This code is actually a copy/paste from a snippet in the ERC4626 implementation from OZ, but modified to support ETH native assets. We decided to keep the duplicate check there as it makes it easier to confirm that the logic is indeed exactly the same as in the OZ contract, and to ensure correctness of our logic.

Attack / Description	The deposit function includes a redundant check for <code>assets == 0</code> within its logic, even though the function already employs a <code>nonZeroAmount/assets</code> modifier designed to revert any transactions where the assets amount is zero. This redundancy may lead to unnecessary code complexity and potential confusion about the function's behavior and the effectiveness of the modifiers used.
Code	<div>Line 17 -20 (FijaVault2Txn.sol)</div> <pre>uint256 supply = totalSupply();  tokens = (assets == 0    supply == 0)     ? _initialConvertToShares(assets, Math.Rounding.Down)     : assets.mulDiv(         supply,         totalAssetBeforeDeposit,         Math.Rounding.Down     );</pre>
Result/Recommendation	<div>To streamline the function and enhance code clarity, the redundant check for <code>assets == 0</code> should be removed from the function body, relying solely on the <code>nonZeroAmount/assets</code> modifier to handle cases where assets are zero.</div> <div>// Remove the check for <code>assets == 0</code>, rely on modifier</div> <pre>tokens = (supply == 0)</pre>



	<pre> ? _initialConvertToShares(assets, Math.Rounding.Down) : assets.mulDiv(     supply,     totalAssetBeforeDeposit,     Math.Rounding.Down ); </pre>
--	--

### 6.2.23 Redundancy in Asset Calculation redeem Function

Severity: INFORMATIONAL

Status: FIXED

Code: CWE-682

File(s) affected: FijaVault2Txn.sol

Update: Duplication of previewRedeem was removed by pulling this method call outside the if/then/else block.

<b>Attack / Description</b>	<p>The redeem function within the contract redundantly calculates the assets variable in both conditional branches of an if-else structure, leading to unnecessary code duplication. This issue affects the clarity and efficiency of the contract's code, potentially leading to maintenance challenges and increasing the cognitive load required to understand the contract's behavior.</p>
<b>Code</b>	<p>Line 244 - 254 (FijaVault2Txn.sol)</p> <pre> uint256 currentBalanceAvailable;  if (asset() == ETH) {     _executionFee = msg.value;     assets = previewRedeem(tokens);      // execution fee is reduced from current balance     currentBalanceAvailable = address(this).balance - msg.value; } else { </pre>

	<pre> assets = previewRedeem(tokens);  currentBalanceAvailable = IERC20(asset()).balanceOf(address(this));  } </pre>
<b>Result/Recommendation</b>	<p>To improve the efficiency and maintainability of the redeem function, it is recommended to assign the assets variable upon declaration, before the branching logic. This change will eliminate the need for redundant code and clarify that assets is set using the same method regardless of the branch conditions.</p> <pre> // Initialize assets at the start to avoid redundant calculations uint256 assets = previewRedeem(tokens); uint256 assetsToReturn;  SafeERC20.forceApprove(     IERC20(_strategy),     address(_strategy),     type(uint256).max ); uint256 currentBalanceAvailable;  if (asset() == ETH) {     _executionFee = msg.value;     currentBalanceAvailable = address(this).balance - msg.value; } else {     currentBalanceAvailable = IERC20(asset()).balanceOf(address(this)); } // The rest of the function remains unchanged...  } </pre>

#### 6.2.24 Missing Struct Packing

Severity: INFORMATIONAL

Status: FIXED

Code: NA

File(s) affected: IFijaVault.sol

Update by the Project: uint precision was updated to match storage slot

<b>Attack / Description</b>	In Solidity, structs are packed into storage slots. A storage slot is 32 bytes in size, which means that if a struct fits entirely within a storage slot, it will consume only one slot. However, if a struct spans multiple storage slots, it will consume additional gas for storage operations. Currently, the <i>StrategyCandidate</i> struct consists of an address (20 bytes) and an uint256 (32 bytes) for a unit of time which spans over two storage slot where it only needs one.
<b>Code</b>	Line 17 -20: <pre>struct StrategyCandidate {     address implementation;     uint256 proposedTime; }</pre>
<b>Result/Recommendation</b>	It is recommended to use the remaining 12 bytes of the first storage slot for the unit of time. The uint represents a unix timestamp, which is safe if uint64 (8 bytes) is used. Use uint64 for the <i>proposedTime</i> to reduce storage size and gas consumption during the contracts lifetime.

#### 6.2.25 Magic Numbers

Severity: INFORMATIONAL

Status: FIXED

Code: NA

File(s) affected: CurveConvexStrategy.sol, CurveConvexStrategyProtocol.sol

Update by the Project: Variable declarations updated for readability, Variables are now used instead of magic numbers.

<b>Attack / Description</b>	The current implementation contains several magic numbers on the first glance, making the code harder to understand and to maintain in the future. Solidity offers functions and alias to enhance readability of numbers significantly.
<b>Code</b>	<p>CurveConvexStrategy</p> <p>Line 207: <code>if (block.timestamp &gt;= _lastHarvestTime + 2592000) {</code>  Line 434: <code>_lastRebalanceTime + 7257600 &lt; block.timestamp &amp;&amp;</code>  Line 442: <code>_lastRebalanceTime + 2419200 &lt; block.timestamp &amp;&amp;</code></p> <p>CurveConvexStrategyProtocol</p> <p>Line 22: <code>uint256 constant PRECISION_18 = 1_000_000_000_000_000_000;</code>  Line 24 – 25: <code>uint256 constant PRECISION_30 =</code>  <code>1_000_000_000_000_000_000_000_000_000_000_000;</code></p>
<b>Result/Recommendation</b>	<p>Even though some of the snippets already contain comments, indicating what these numbers stand for, we still recommend making them either more explicit or to create memory variables, telling the developer, what those values are meant for.</p> <p>Values with many zeros like <i>PRECISION_18</i> and <i>PERCISION_30</i> should be written in the “power of” form like <i>10**18</i>.</p> <p>Solidity offers time units which should be used to convert seconds to a readable format and prevent calculation errors. These units are seconds, minutes, hours, days and weeks. It is strongly recommended to use <i>30 days</i> instead of writing plain seconds (<i>2592000</i>).</p>

#### 6.2.26 Misleading Variable Name

Severity: INFORMATIONAL

Status: FIXED

Code: NA

File(s) affected: CurveConvexStrategyProtocol.sol

Update by the Project: Variable declaration was adjusted for readability and is now matching variable naming.

<b>Attack / Description</b>	The internal variable <i>PRECISION_30</i> implies that it has 30 zeros, but the actual value has 36 zeros. This can lead to confusion during development and to unintended behavior.
<b>Code</b>	Line 24 – 25: uint256 constant PRECISION_30 = 1_000_000_000_000_000_000_000_000_000_000_000;
<b>Result/Recommendation</b>	It is recommended to rename the <i>PRECISION_30</i> variable to <i>PRECISION_36</i> to avoid any misinterpretation. Additionally, we refer to the issue “magic numbers” to implement a more readable way of variable declaration.

#### 6.2.27 Missing Immutable Modifier

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: NA

File(s) affected: CurveConvexBase.sol

Update by the Project: We were not able to make all variables immutable as that would have exceeded our contract size limit.

<b>Attack / Description</b>	Variables should be declared as immutable in Solidity when their values are known and will not change after deployment. This improves gas efficiency, enhances security, and provides clarity in code. It's particularly useful for constants, configuration values, and other unchanging data. The CurveConvexBase contracts holds many variables meant to be immutable but are not declared correctly.
<b>Code</b>	Line 114 – 139: all vars from DE_PEG_CHECK to LIQUIDITY_THR_BPS Line 154 – 159: address EMERGENCY_CCY;

	address _emergencyPool; 1_000_000_000_000_000_000_000_000_000_000_000;
<b>Result/Recommendation</b>	It is recommended to use the immutable modifier to save gas on contract deployment, enhance readability and ensure that this variables will never be changed in the future. This enhances security, gas efficiency and code readability.

#### 6.2.28 Redundant Payable Conversion

Severity: INFORMATIONAL

Status: FIXED

Code: NA

File(s) affected: CurveConvexPeriphery.sol, CurveConvexStrategy.sol, FijaVault.sol

Update by the Project: Code was updated to remove the payable conversion.

<b>Attack / Description</b>	The current implementation has several references to access the contracts balance. While accessing the contract balance it converts the address to payable, which is not required and consumes additional gas while function execution.
<b>Code</b>	CurveConvexPeriphery Line 462 & 535 CurveConvexStrategy Line 94, 365 & 572 FijaVault Line 147, 196, 227, 263 & 295 payable(address(this)).balance;
<b>Result/Recommendation</b>	It is recommended to remove the conversion to payable because it has no effect on the functionality of the function call and consumes additional gas. The recommended pattern to access the balance of a contract is <i>address(this).balance</i> .

### 6.2.29 Unused Return Value

Severity: INFORMATIONAL

Status: FIXED

Code: NA

File(s) affected: CurveConvexBase.sol

Update by the Project: Code was updated to remove the unused return value.

Attack / Description	<p>The <code>_findAddrReplace</code> function returns the potentially modified address array together with a boolean value, indicating, if the array was modified or not.</p> <p>The only function call to <code>_findAddrReplace</code> does not utilize the returned boolean value "<code>isFound</code>" though. Unused variables are indicators for either missing code logic, making use of the variable, thus leading to missing functionality, or simply forgotten code logic, that can be removed.</p>
Code	<pre>Line 505 – 519 function _findAddrReplace(     address[8] memory inputAddrs,     address find,     address replaceWith ) private pure returns (bool, address[8] memory) {     bool isFound = false;     for (uint8 i = 0; i &lt; inputAddrs.length; i++) {         if (inputAddrs[i] == find) {             isFound = true;             inputAddrs[i] = replaceWith;             break;         }     }     return (isFound, inputAddrs); }</pre>
Result/Recommendation	<p>We suggest either eliminating the return value to conserve gas or providing documentation to explain its potential future necessity. Alternatively, if the use-case still exists, it should be</p>



	appropriately implemented.
--	----------------------------

6.2.30 Missing Verification of Asset Addresses

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: NA

File(s) affected: FijaVault.sol

Update by the Project: Risk covered as part of the deployment process. We are indeed not using a list of pre-verified supported assets, as new Curve pools may be created after finalization of our codebase. As part of our deployment process, we do execute an internal test deposit however, to ensure the contract was correctly created and no mistakes were made with the asset addresses supplied. This test deposit is made by fija before the strategy and vault are made available to external customers.

Attack / Description	No verification of the supported assets addresses on contract deployment.
Code	<div> Line 28 (FijaVault.sol) <pre> constructor(     IFijaStrategy strategy_,     IERC20 asset_, // cannot define (extract) at later stage     string memory tokenName_, // for vault this should be as parameter     string memory tokenSymbol_, // for vault this should be as parameter     address governance_,     address reseller_,     uint256 approvalDelay_,     uint maxTicketSize_,     uint maxVaultValue_ ) </pre> </div>



<b>Result/Recommendation</b>	We recommend creating a list of supported assets and verify the addresses of the assets on contract deployment.
------------------------------	---

## 6.3 SWC Attacks

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


ID	Title	Relationships	Test Result
<a href="#">SWC-123</a>	Requirement Violation	<a href="#">CWE-573: Improper Following of Specification by Caller</a>	✓
<a href="#">SWC-122</a>	Lack of Proper Signature Verification	<a href="#">CWE-345: Insufficient Verification of Data Authenticity</a>	✓
<a href="#">SWC-121</a>	Missing Protection against Signature Replay Attacks	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-120</a>	Weak Sources of Randomness from Chain Attributes	<a href="#">CWE-330: Use of Insufficiently Random Values</a>	✓
<a href="#">SWC-119</a>	Shadowing State Variables	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-118</a>	Incorrect Constructor Name	<a href="#">CWE-665: Improper Initialization</a>	✓
<a href="#">SWC-117</a>	Signature Malleability	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-116</a>	Timestamp Dependence	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-115</a>	Authorization through tx.origin	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-114</a>	Transaction Order Dependence	<a href="#">CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-113</a>	DoS with Failed Call	<a href="#">CWE-703: Improper Check or Handling of Exceptional Conditions</a>	✓
<a href="#">SWC-112</a>	Delegatecall to Untrusted Callee	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-111</a>	Use of Deprecated Solidity Functions	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-110</a>	Assert Violation	<a href="#">CWE-670: Always-Incorrect Control Flow Implementation</a>	✓
<a href="#">SWC-109</a>	Uninitialized Storage Pointer	<a href="#">CWE-824: Access of Uninitialized Pointer</a>	✓
<a href="#">SWC-108</a>	State Variable Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-107</a>	Reentrancy	<a href="#">CWE-841: Improper Enforcement of Behavioral Workflow</a>	✓
<a href="#">SWC-106</a>	Unprotected SELFDESTRUCT Instruction	<a href="#">CWE-284: Improper Access Control</a>	✓
<a href="#">SWC-105</a>	Unprotected Ether Withdrawal	<a href="#">CWE-284: Improper Access Control</a>	✓
<a href="#">SWC-104</a>	Unchecked Call Return Value	<a href="#">CWE-252: Unchecked Return Value</a>	✓


ID	Title	Relationships	Test Result
<a href="#">SWC-103</a>	Floating Pragma	<a href="#">CWE-664: Improper Control of a Resource Through its Lifetime</a>	✓
<a href="#">SWC-102</a>	Outdated Compiler Version	<a href="#">CWE-937: Using Components with Known Vulnerabilities</a>	✓
<a href="#">SWC-101</a>	Integer Overflow and Underflow	<a href="#">CWE-682: Incorrect Calculation</a>	✓
<a href="#">SWC-100</a>	Function Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓

## 6.4 Verify Claims


6.4.1 The audit should thoroughly investigate the curve convex contract to confirm the absence of any vulnerabilities or potential exploits.

**Status:** tested and verified 


6.4.2 The audit should rigorously examine the contract's methods that deal with array manipulations to ensure that there are no vulnerabilities that can lead to unintended modifications or exploitable behaviors.

**Status:** tested and verified 


6.4.3 The audit must validate the functionality that allows for the replacement of token addresses, ensuring it accurately locates and substitutes the desired addresses without discrepancies, preventing any wrongful replacements.

**Status:** tested and verified 


6.4.4 A comprehensive review should be conducted on the contract's mechanisms to build and input amounts in specific array positions. This will ensure that tokens are correctly allocated without any inadvertent impacts or omissions.

**Status:** tested and verified 


6.4.5 The audit should rigorously assess the contract's interaction with external Curve and Convex protocols, ensuring that any calls or references to these platforms are secure, reliable, and resistant to potential vulnerabilities like reentrancy attacks or unexpected behavior from external dependencies.

**Status:** tested and verified 


6.4.6 The security audit should thoroughly investigate the Vault smart contract to confirm the absence of any vulnerabilities or potential exploits

**Status:** tested and verified 


6.4.7 The audit must validate the accuracy of token storage and transfer functions, ensuring that these processes are handled correctly and securely within the vault.

**Status:** tested and verified 


6.4.8 The correct functioning of deposit and withdrawal mechanisms must be tested to confirm that users' assets can be securely and efficiently moved in and out of the vault.

**Status:** tested and verified 

6.4.9 The effectiveness of the implemented yield strategies should be verified to ensure that they are operating as intended and generating expected returns.

**Status:** tested and verified 

6.4.10 The robustness of whitelisting protocols and associated security measures needs to be scrutinized to confirm their effectiveness in preventing unauthorized access and transactions.

**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 Fija Finance 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 30 issues, classified as follows:

- No critical issues were found.
- No high severity issues were found.
- Ten medium severity issues were found.
- Ten low severity issues were discovered
- Ten 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. Overall, the code quality and documentation have been showing the auditor team a high grade of professionalism and have been greatly benefiting the auditing process. We advise the Fija Finance team to implement the recommendations to further enhance the code's security and readability.

Update (01.09.2023): The Eth-capable vault and strategy implementations have undergone thorough scrutiny and assessment, and no vulnerabilities or concerns have been identified. Rigorous controls have been meticulously integrated to ensure the security and integrity of the system. Comprehensive measures have been taken to safeguard against potential threats and risks. As a result, users can confidently engage with the components, knowing that they have been meticulously designed and tested to meet the highest standards of security and functionality.





Update (25.02.2024): Following the thorough audit conducted by softstack's experts, the Fija development team has diligently addressed all necessary issues. We have successfully re-verified the fixes implemented for the identified vulnerabilities and improvements in the smart contract's code. The team's commitment to security and proactive approach in resolving the audit findings have significantly enhanced the integrity and robustness of the smart contract, ensuring a higher level of trust and reliability for its users.

Update (26.04.2024): We are pleased to provide an update on the recent GMXv2 audit conducted on the Fija Protocol. The audit has identified a total of 18 new issues, which are classified as follows:

Medium Issues: 7 new issues have been identified. These issues mostly pertain to inconsistencies and potential operational risks in the withdrawal and redemption functions, as well as potential risks from large transaction requests exceeding available balances.

Low Issues: 8 new issues have been detected. These primarily involve minor security oversights and efficiency improvements in contract operations that could lead to precision loss or redundant code.

Informational Issues: 3 new issues were noted, highlighting areas for non-critical improvements, such as optimizing existing functionalities and enhancing clarity in the codebase.

These findings will guide the next steps in enhancing the security and operational efficiency of the Fija Protocol. Further details and recommendations have been outlined in the full audit report.

Update (21.07.2024): After a thorough re-check, we have verified the additional code changes made by the Fija team and the mitigated audit findings. They have successfully addressed the identified issues and implemented key security enhancements and performance improvements. This proactive approach has significantly strengthened the system's security, efficiency, and user experience, ensuring a higher level of trust and reliability for its users. Below are additional code changes provided by the Fija Team:

#185	add getExecutionGasLimit to Strategy contract, not only to reader contract	Code update	Code improvement to make getExecutionGasLimit method directly accessible from Strategy Contract.	/strategies/GMXv2/GMXv2strategy.sol
#157	Remove need for increaseAllowance(strategy)	Code update	Remove the need for an increaseAllowance call to transfer strategy tokens from vault to strategy	/base/FijaVault2Txn.sol /base/Errors.sol



#188	Ensure strategy status remains consistent in case of GMXKeeper delays	Code update	Fixes a problem where the strategy value was reduced between the submission of a GMX buy/sell order and the actual execution of that order by the GMXKeeper. This value reduction would have allowed malicious actors to enter a strategy at a reduced valuation between the 2 transactions.	/base/FijaVault2Txn.sol /protocols/GMXv2/IDataStore.sol /strategies/GMXv2/GMXv2Keys.sol /strategies/GMXv2/GMXv2Lib.sol /strategies/GMXv2/GMXv2Lib2.sol /strategies/GMXv2/GMXv2Type.sol /strategies/GMXv2/GMXv2Strategy.sol /strategies/GMXv2/GMXv2StrategyReader.sol /strategies/GMXv2/IGMXv2StrategyReader.sol
#195	Include GMXPendingAssets in status() output	Code update	Include the amount of GMX Pending Assets (introduced as part of ticket 188) in the status call for better strategy monitoring	/strategies/gmxv2/GMXv2Strategy.sol /strategies/gmxv2/GMXv2StrategyReader.sol /strategies/gmxv2/IGMXv2Strategy.sol
#196	Automated whitelisting of strategy to the vault	Code update	Remove manual deployment step needed to whitelist strategy to the vault.	/base/FijaVault2Txn.sol /base/Errors.sol
#197	Changes to codebase due to testing with different starting blocks	Code update	The Execution fee for entering emergency mode was increased due to failures on some starting blocks	/strategies/GMXv2/GMXv2StrategyReader.sol
#198	Put afterWithdraw() and afterRedeem() in a try/catch	Code update	The execution of the afterWithdraw() and afterRedeem() methods was put in a try/catch to prevent the entire transaction from failing, which would have resulted in an invalid strategy status.	/strategies/gmxv2/GMXv2Lib.sol
#200	Fix harvest 4.4 tests	Code update	Updates to harvest test cases and codebase to make harvest cycle more robust.	/strategies/GMXv2/GMXv2Lib.sol
#203	Get GMX withdrawals converted to 1 currency instead of 2	Code update	Update to the GM token withdrawal logic where GMX is now responsible for the swap of the output to the single input currency. Earlier we used Uniswap to execute the swap ourselves, but this lead to situations where the swap was not properly executed if the transaction was reverted for some reason. That in return then led to an incorrect strategy value calculation, allowing malicious actors to enter a strategy at a discounted valuation.	/strategies/GMXv2/GMXv2Keys.sol /strategies/GMXv2/GMXv2Lib.sol /strategies/GMXv2/GMXv2StrategyReader.sol /strategies/GMXv2/IGMXv2StrategyReader.sol
#202	Adjust emergency mode to leave some funds behind	Code update	Adjustment to the logic to enter emergency mode where a small portion of the funds is left behind on Aave. Since Aave was recalculating token amounts during our transaction we had some unwanted exceptions due to running out of Aave tokens.	/strategies/GMXv2/GMXv2Strategy.sol
#205	For empty strategy return false for needRebalance	Code update	Empty strategies were continuously rebalancing due to a faulty needRebalanc() flag	/strategies/GMXv2/GMXv2Lib2.sol
#206	Reduce impact of deposits on token price	Code update	Code adjustment to reduce the impact of very big deposits (compared to strategy size) to the token price.	/strategies/GMXv2/GMXv2StrategyReader.sol
#208	Add additional fields to status() call	Code update	Add OpenInterest related data to status() output for easier better strategy monitoring.	/strategies/GMXv2/GMXv2Keys.sol /strategies/GMXv2/GMXv2StrategyReader.sol
#213	Unexpected reduction of strategy token price on ETH deposit	Code update	The deposit() logic for native currency (ETH) deposits was incorrectly calculating the value of the strategy, and took the value of the deposit into account while determining the pending profitshare amount. The resulting strategy value was structurally low, and as a result too many strategy tokens were minted upon ETH deposits.	/strategies/GMXv2/GMXv2Strategy.sol
#217	Adjust execution fees for deposit that results in a GMX withdrawal	Code update	Execution fees for the 2nd GMX transaction were not always correctly calculated. Specifically in the case where a strategy desposit results in a GMX token withdrawal we had too low execution fees.	/strategies/GMXv2/GMXv2StrategyReader.sol
#218	Take out ARB tokens airdropped to strategy	Code update	GMX is executing airdrops of ARB tokens as part of their incentive program. This change converts ARB tokens to USDC in order for the strategy to absorb rewards.	/strategies/GMXv2/GMXv2Keys.sol /strategies/GMXv2/GMXv2Lib.sol /strategies/GMXv2/GMXv2strategy.sol /strategies/GMXv2/IGMXv2strategyReader.sol /strategies/GMXv2/IGMXv2strategyReader.sol

## 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, security, and consulting services. Softstack's competency extends across the security landscape of prominent blockchains like Solana, Tezos, 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.

