Fija Finance

Aave Optimiser

SMART CONTRACT AUDIT

16.01.2025

Made in Germany by Softstack.io

Table of contents

1. Disclaimer

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

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of 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 (17.11.2024)	Layout
0.4 (28.11.2024)	Automated Security Testing Manual Security Testing
0.5 (06.12.2024)	Verify Claims and Test Deployment
0.6 (06.12.2024)	Testing SWC Checks
0.9 (06.12.2024)	Summary and Recommendation
1.0 (18.12.2024)	Final document
1.1 (16.01.2025)	Re-check

2. About the Project and Company

Company address:

fija Finance GmbH
Müllerstr. 43
80469 Munich
Germany

Website: https://www.fija.finance

Twitter: 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

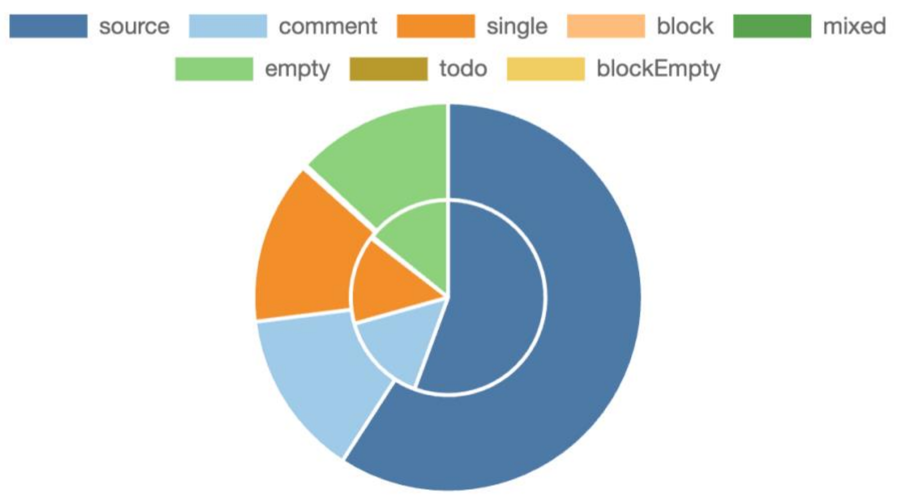
File	Fingerprint (MD5)
./strategies/AaveOptimizer/AaveOptimizerStrategy.sol	c6cf5dc7afc9bc46988294ad8c359514
./strategies/AaveOptimizer/AaveOptimizerLib.sol	69bd1ba3dbcf2812f4fa7744e509ecd9
./strategies/AaveOptimizer/AaveHelpers.sol	801483e7eebbacfc67db5a0ad0aa70c5
./strategies/AaveOptimizer/AaveKeys.sol	2b412e6b6328f4290e2d32e64168cc9c
./protocols/aave/IDefaultInterestRateStrategyV2.sol	fcbb195057b5a9c649acc2f5aba4d72e
./base/FijaERC4626Base.sol	b2aceb0cba6432f7587b78a978e3eccc
./base/FijaStrategy.sol	b12d70800e0f5a2a80f533b58004e0ed
./base/FijaStrategyBase.sol	681ddf71c4c953f27a1e647c7bbf99fc

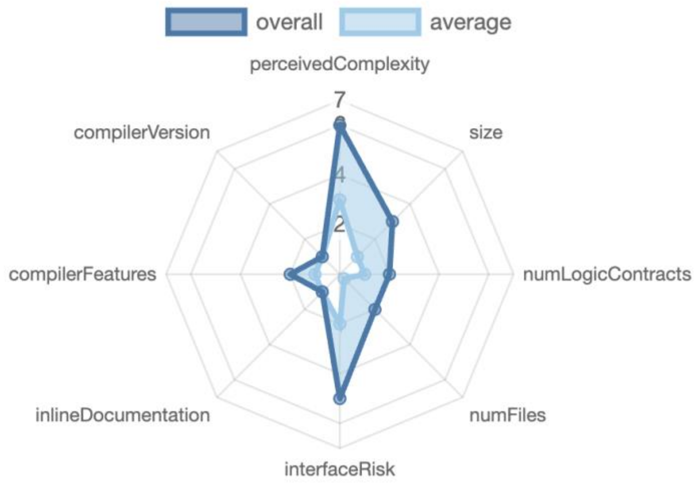
Update 09.01.2025

File	Fingerprint (MD5)
./strategies/AaveOptimizer/AaveOptimizerStrategy.sol	21f64cb93daa3e7da0559b2797f197c3
./strategies/AaveOptimizer/AaveOptimizerLib.sol	6dc3442165ffbc941776137655988b5d
./strategies/AaveOptimizer/AaveHelpers.sol	a90afafe8aba6dfc7135c22754453afe
./strategies/AaveOptimizer/AaveKeys.sol	1484ef5a66e1da3d0a37c5500d958d29
./protocols/aave/IDefaultInterestRateStrategyV2.sol	6d352d34c897e5e0c9dcf2812036c570
./base/FijaERC4626Base.sol	b2aceb0cba6432f7587b78a978e3eccc

./base/FijaStrategy.sol	018eaf6e90f50e6a7dfb3e12331f1c0d
./base/FijaStrategyBase.sol	681ddf71c4c953f27a1e647c7bbf99fc

5.2 Source Lines & Risk





5.3 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	ECRecover	New/Create/Create2
yes					

TryCatch	Σ Unchecked
	yes

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

Public	Payable
52	15

External	Internal	Private	Pure	View
24	40	12	2	49

StateVariables

Total	Public
34	0

5.4 Dependencies / External imports

Dependency / Import Path	**Source**
@aave/core-v3/contracts/interfaces/IAaveOracle.sol	https://github.com/aave/aave-v3-core/blob/master/contracts/interfaces/IAaveOracle.sol/
@aave/core-v3/contracts/interfaces/IPool.sol	https://github.com/aave/aave-v3-core/blob/master/contracts/interfaces/IPool.sol
@aave/core-v3/contracts/interfaces/IPoolDataProvider.sol	https://github.com/aave/aave-v3-core/blob/master/contracts/interfaces/IPoolDataProvider.sol
@aave/periphery-v3/contracts/misc/interfaces/IWETH.sol	https://github.com/aave/aave-v3-core/blob/master/contracts/misc/interfaces/IWETH.sol
@openzeppelin/contracts/utils/Strings.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/utils/Strings.sol
@openzeppelin/contracts/utils/math/Math.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.9.3/contracts/utils/math/Math.sol
@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol	https://github.com/Uniswap/v3-core/blob/main/contracts/interfaces/IUniswapV3Pool.sol
@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol	https://github.com/Uniswap/v3-periphery/blob/main/contracts/interfaces/ISwapRouter.sol

5.5 Source Unites in Scope

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/base/FijaStrategyBase.sol	1		180	150	74	56	69
contracts/base/FijaStrategy.sol	1		426	368	210	92	173
contracts/base/FijaERC4626Base.sol	1		469	385	200	134	168
contracts/protocols/aave/IDefaultInterestRateStrategyV2.sol		1	16	13	3	9	3
contracts/strategies/AaveOptimizer/AaveKeys.sol	1		41	41	16	17	15
contracts/strategies/AaveOptimizer/AaveHelpers.sol	1		39	39	28	4	4
contracts/strategies/AaveOptimizer/AaveOptimizerLib.sol	1		185	173	135	4	83
contracts/strategies/AaveOptimizer/AaveOptimizerStrategy.sol	1		892	870	670	46	347
Totals	**7**	**1**	**2248**	**2039**	**1336**	**362**	**862**

- **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

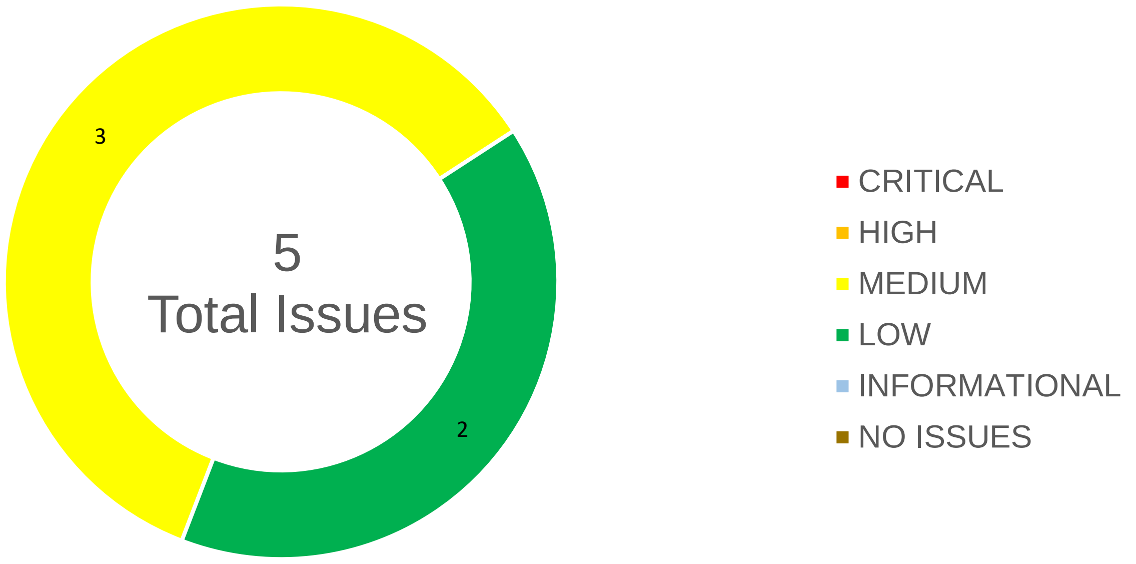
The Fija Team provided us with the files that needs to be tested. The scope of the audit are the Aave optimiser contracts.

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

1. Safe & Accurate Aave Integration The audit should thoroughly investigate how the strategy interacts with Aave’s lending pool and related contracts (e.g., IPool, IPoolDataProvider) to ensure calls and references are secure, correctly implemented, and not susceptible to reentrancy or other external attack vectors.
2. Correct Handling of aToken Balances The audit must confirm that the strategy’s logic for depositing into and withdrawing from Aave correctly tracks and updates users’ balances in aTokens. Any discrepancies in aToken balances could lead to misallocation of funds or yield miscalculations.
3. Robust Swap Mechanisms via Uniswap The security review should rigorously inspect the strategy’s use of Uniswap V3 swaps (ISwapRouter and IUniswapV3Pool) to move between the vault’s primary asset and each Aave token. Special attention must be given to slippage settings, minimum output parameters, and fee calculations, ensuring they cannot be manipulated by malicious actors.
4. Accurate Distribution of Allocations A comprehensive review is required of the strategy’s allocation logic—which uses scoring and utilization data to decide how much capital to place into each Aave token. This allocation must correctly account for minimum and maximum thresholds, ensuring no unintended over-allocation to riskier tokens or under-allocation that could reduce potential returns.
5. Secure Whitelisting & Emergency Mode Controls The strategy must be scrutinized for its handling of whitelisted addresses (the only addresses allowed to call certain sensitive functions) and the emergency mode switch. The auditor should confirm that only authorized individuals can trigger or revert from emergency mode and that these safeguards cannot be bypassed, ensuring asset safety if the protocol faces unforeseen failures.

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	Emergency Mode Restriction Not Applied to Critical Functions	MEDIUM	ACKNOWLEDGED
6.2.2	Incomplete Fund Allocation in _recalculateAllocations Function	MEDIUM	FIXED
6.2.3	Excessive Swap Deadline Exposes Transactions to Sandwich Attacks	MEDIUM	FIXED
6.2.4	Unused Payable Functions Leading to Potential Ether Lock	LOW	ACKNOWLEDGED
6.2.5	Insufficient Input Validation in AaveOptimizerStrategy Constructor	LOW	FIXED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

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

HIGH ISSUES

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

MEDIUM ISSUES

During the audit, softstack‘s experts found **3 Medium issues** in the code of the smart contract.

6.2.1 Emergency Mode Restriction Not Applied to Critical Functions

Severity: MEDIUM
Status: ACKNOWLEDGED
Code: NA
File(s) affected: FijaStrategy.sol
Update by the Project: The emergency mode modifier is indeed not applied to the withdraw and redeem functions. The ermergency mode is activated when the strategy contracts deems that a certain position (in this case an Aave loan) is too risky to maintain, e.g. due to fija taking up a too big portion of a small pool. In this case we will withdraw all funds from the respective positions and convert them back to deposit currency. During emergency mode:
- Deposits are restricted -- we do not want our customers to unwillingly invest funds in a strategy that would not be generating yield
- The strategy will allow for withdrawals and redeems -- as we want the customers to be able to access their funds and invest them in a different strategy. Restricting the deposits/withdrawals would lead to the unwanted behaviours that we would first have to exit emergency mode (thereby investing the funds again in unsafe pools) before the customer gets access to the funds.

Attack / Description	The current implementation of the emergency mode restriction is incomplete. While the emergencyModeRestriction modifier is correctly applied to the deposit function, it is not applied to other critical functions such as withdraw and redeem. This oversight could potentially allow users to

	perform these operations even when the contract is in emergency mode, which may lead to unexpected behavior or security risks. In the FijaStrategy contract, the withdraw and redeem functions are implemented without the emergencyModeRestriction modifier. These functions include logic to handle cases where the requested withdrawal amount is greater than the current balance, but they do not explicitly check for the emergency mode status. This inconsistency in applying emergency mode restrictions across different functions could compromise the effectiveness of the emergency mode feature.

Code	Line 320 – 345 (FijaStrategy.sol)
	`function withdraw(`
	`        uint256 assets,`
	`        address receiver,`
	`        address owner`
	`    )`
	`//..//`
	`        if (assets > currentBalance) {`
	`            _genericInvestmentLogic(int256(currentBalance) - int256(assets));`
	`        }`
	
	`        return super.withdraw(assets, receiver, owner);`
	`    }`
	
	Line 286 – 315 (FijaStrategy.sol)
	`function redeem(`
	`        uint256 tokens,`
	`        address receiver,`
	`        address owner`
	`    )`

	```
	//..//
	    if (withdrawAmount > currentBalance) {
	      _genericInvestmentLogic(
	        int256(currentBalance) - int256(withdrawAmount)
	      );
	    }

	    return super.redeem(tokens, receiver, owner);
	  }
	```
Result/Recommendation	Recommendation: 1. Apply the emergencyModeRestrictionmodifier to the withdraw and redeem functions in the FijaStrategy contract. 2. Implement a consistent check for emergency mode status across all critical functions.

6.2.2 Incomplete Fund Allocation in _recalculateAllocations Function

Severity: MEDIUM
Status: FIXED
Code: NA
File(s) affected: AaveOptimizerStrategy.sol
Update by the Project: We've adjusted the calculation logic so no funds remain behind. Note that the current calculation logic may lead to 'overallocation' of funds, i.e. the sum of all pool allocations may be bigger than the funds available. As we have checks in place to never actually transfer more funds than available to the pools this is not a problem, and the last pool will get a slightly lower allocation than calculated.

Attack / Description	The _recalculateAllocations function in the AaveOptimizerStrategy contract has a potential issue with incomplete allocation of funds. Due to rounding errors in integer division operations involving BASIS_POINTS_DIVISOR, not all of the remainingFunds may be allocated to tokens. This can result in some funds remaining idle, leading to inefficient use of capital and suboptimal yield generation. The issue arises from two main factors: 1. The allocationBps calculation may not sum up to exactly BASIS_POINTS_DIVISOR due to integer division. 2. The calculation of allocations[i] involves another division by BASIS_POINTS_DIVISOR, which can lead to further rounding down. **Impact:** 1. Inefficient use of capital, as some funds may remain unallocated and idle. 2. Potential discrepancies between the total allocated funds and the actual balance of the contract. 3. Suboptimal yield generation, as not all funds are being utilized for investment.

Code	Line 85 – 102 (AaveOptimizerStrategy.sol)
	`for (uint256 i; i < TOKEN_NUMBER; ) {`
	`        AaveInput memory aaveInput = data_.aaveInput[i];`
	
	`        _aaveTokens.push(aaveInput.token);`
	`        _tokenMinAllocationBps[aaveInput.token] = aaveInput`
	`            .minAllocationBps;`
	
	`        _tokenMaxTvlBps[aaveInput.token] = aaveInput.maxTvlBps;`

	```
	        _tokenUniswapPool[aaveInput.token] = aaveInput.uniswapPool;

	        _tokenEmeTvlBps[aaveInput.token] = aaveInput.emergencyTvlBps;

	        unchecked {
	          ++i;
	        }
	      }
	    }
	```

Result/Recommendation	Implement a "sweep" mechanism at the end of the allocation process to ensure all funds are allocated:
	
	1. After the main allocation loop, calculate the total allocated amount.
	2. Determine the difference between total Funds and the total allocated amount.
	3. Distribute this difference among the tokens, either equally or based on their existing allocations.
	
	Example implementation:
	```
	uint256 totalAllocated = 0;
	for (uint256 i = 0; i < TOKEN_NUMBER; i++) {
	  totalAllocated += allocations[i];
	}
	uint256 unallocatedFunds = totalFunds - totalAllocated;
	if (unallocatedFunds > 0) {
	  // Distribute unallocated funds among tokens
	```

	```
	  for (uint256 i = 0; i < TOKEN_NUMBER; i++) {
	    allocations[i] += unallocatedFunds / TOKEN_NUMBER;
	} }
	```

6.2.3 Excessive Swap Deadline Exposes Transactions to Sandwich Attacks

Severity: MEDIUM
Status: FIXED
Code: NA
File(s) affected: AaveOptimizerStrategy.sol
Update by the Project: We don't think this was a problem in the first place: the Uniswap swap is initiated from the smart contract, which means that it will always be executed in the same transaction/block, regardless of the swap deadline we specify. Nevertheless, we've adjusted the deadline and changed it to 'block. timestamp' to explicitly specify that this will be executed in the same transaction.

Attack / Description	The _deposit and _withdraw functions in the AaveOptimizerStrategy contract use a fixed deadline of block.timestamp + 120 (2 minutes) for Uniswap swaps. This extended timeframe provides a significant window during which the transaction remains pending in the mempool, potentially exposing it to sandwich attacks and other forms of MEV (Maximal Extractable Value) exploitation. Sandwich attacks occur when malicious actors front-run a user's transaction by placing their own transaction immediately before and after the user's, manipulating the price and extracting value from the user's trade. The longer a transaction remains pending, the more opportunity there is for such attacks. Users may experience unexpected slippage and financial losses due to sandwich attacks or other MEV exploitation techniques. This undermines the fairness and efficiency of trades, potentially eroding user trust in the protocol.

Code	Line 781 – 791 (AaveOptimizerStrategy.sol)
	```
	ISwapRouter.ExactInputSingleParams memory swapInput = ISwapRouter
	        .ExactInputSingleParams(
	```

	```
	        depositCcy,
	        token,
	        IUniswapV3Pool(_tokenUniswapPool[token]).fee(),
	        address(this),
	        block.timestamp + 120,
	        assetsToDeposit,
	        minOut,
	        0
	      );
	```

Result/Recommendation	Recommendation:
	
	1. Implement a shorter, configurable deadline parameter that users can set according to their risk tolerance and network conditions. A typical range might be 30-60 seconds.
	
	2. Consider using a commit-reveal scheme or integrating with MEV protection services like Flashbots to mitigate the risk of sandwich attacks.
	
	3. Implement a maximum deadline limit to prevent excessively long pending times while still allowing user flexibility.
	```
	uint256 constant MAX_DEADLINE = 60; // Maximum 60 seconds
	function setSwapDeadline(uint256 deadline) external {
	  require(deadline > 0 && deadline <= MAX_DEADLINE, "Invalid
	deadline");
	  userDeadline = deadline;
	}
	// In swap function
	```

```
uint256 swapDeadline = block.timestamp + (userDeadline > 0 ?
userDeadline : MAX_DEADLINE);
ISwapRouter.ExactInputSingleParams memory swapInput = ISwapRouter
  .ExactInputSingleParams(
    depositCcy,
    token,
    IUniswapV3Pool(_tokenUniswapPool[token]).fee(),
    address(this),
    swapDeadline,
    assetsToDeposit,
    minOut,
    0
);
```

LOW ISSUES

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

6.2.4 Unused Payable Functions Leading to Potential Ether Lock

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: FijaStrategyBase.sol
Update by the Project: We introduced the payable method since some of our strategies do require these methods to be payable. As we cannot introduce the payable modifier in a subclass we decided to introduce it in the base class thereby having a consistent

interface across all our strategies.

Attack / Description	The contract FijaStrategyBase contains three functions (rebalance(), harvest(), and setEmergencyMode()) that are marked as payable but do not utilize the Ether sent to them. This design could lead to accidental loss of funds if Ether is mistakenly sent to these functions. Specifically: 1. The rebalance() function is payable but only emits an event. 2. The harvest() function is payable but only emits an event. 3. The setEmergencyMode() function is payable but only sets a boolean flag and emits an event. None of these functions use the msg.value or provide any mechanism to retrieve accidentally sent Ether. Any Ether sent to these functions will be locked in the contract without a way to recover it, unless a separate withdrawal mechanism is implemented.

Code	Line 81 – 90 (FijaStrategyBase.sol)
	`function rebalance()`
	`    external`
	`    payable`
	`    virtual`
	`    override`
	`    onlyGovernance`
	`    emergencyModeRestriction`
	`  {`
	`    emit FijaStrategyEvents.Rebalance(block.timestamp, "");`
	`  }`

	Line 104 – 113 (FijaStrategyBase.sol)
	`function harvest()`
	`    external`
	`    payable`
	`    virtual`
	`    override`
	`    onlyGovernance`
	`    emergencyModeRestriction`
	`  {`
	`    emit FijaStrategyEvents.Harvest(block.timestamp, 0, 0, asset(), "");`
	`  }`
	
	Line 127 – 132 (FijaStrategyBase.sol)
	`function setEmergencyMode(`
	`    bool turnOn`
	`  ) external payable virtual override onlyGovernance {`
	`    _isEmergencyMode = turnOn;`
	`    emit FijaStrategyEvents.EmergencyMode(block.timestamp, turnOn);`
	`  }`

Result/Recommendation	Recommendation: 1. Remove the payable keyword from the rebalance(), harvest(), and setEmergencyMode() functions if they are not intended to receive Ether. 2. If these functions are meant to be payable in derived contracts, implement a mechanism in the base contract to handle or refund any received Ether.

	3. Add explicit checks at the beginning of these functions to ensure msg.value is zero, and revert the transaction if any Ether is sent: `require(msg.value == 0, "This function does not accept Ether");`

6.2.5 Insufficient Input Validation in AaveOptimizerStrategy Constructor

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: AaveOptimizerStrategy.sol
Update by the Project: Additional validations were added to the constructor

Attack / Description	The constructor of the AaveOptimizerStrategy contract lacks proper input validation for critical parameters, particularly the SLIPPAGE_BPS value. This oversight could lead to the contract being deployed with inappropriate slippage settings, potentially resulting in unexpected behaviour or vulnerabilities in trading operations. Specifically, the constructor sets the SLIPPAGE_BPS value directly from the input data without any checks: SLIPPAGE_BPS = data_.slippageBps; Without validation, it's possible to set SLIPPAGE_BPS to an unreasonably high value, which could lead to significant losses during token swaps, or to zero, which might cause transactions to fail due to price movements. Additionally, other parameters such as minAllocationBps, maxTvlBps, and emergencyTvlBps are also set without validation, which could lead to suboptimal or potentially harmful contract configurations.

Code	Line 83 (AaveOptimizerStrategy.sol) `SLIPPAGE_BPS = data_.slippageBps;`

Result/Recommendation	1. Implement input validation for SLIPPAGE_BPS in the constructor:
	
	`require (data_.slippageBps > 0 && data_.slippageBps <=`
	`MAX_SLIPPAGE_BPS, "Invalid slippage value");`
	`SLIPPAGE_BPS = data_.slippageBps;`
	
	Where MAX_SLIPPAGE_BPS is a reasonable upper limit (e.g., 1000 for 10%).
	
	2. Add similar checks for other critical parameters:
	`require (aaveInput.minAllocationBps <= BASIS_POINTS_DIVISOR, "Invalid minAllocationBps");`
	`require (aaveInput.maxTvlBps <= BASIS_POINTS_DIVISOR, "Invalid maxTvlBps");`
	`require (aaveInput.emergencyTvlBps <= BASIS_POINTS_DIVISOR, "Invalid emergencyTvlBps");`
	
	3. Consider implementing additional logical checks, such as ensuring emergencyTvlBps is less than maxTvlBps.

6.3 SWC Attacks

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

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

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

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

6.4 Verify Claims

6.4.1 Safe & Accurate Aave Integration The audit should thoroughly investigate how the strategy interacts with Aave's lending pool and related contracts (e.g., IPool, IPoolDataProvider) to ensure calls and references are secure, correctly implemented, and not susceptible to reentrancy or other external attack vectors.
Status: tested and verified ✅

6.4.2 Correct Handling of aToken Balances The audit must confirm that the strategy's logic for depositing into and withdrawing from Aave correctly tracks and updates users' balances in aTokens. Any discrepancies in aToken balances could lead to misallocation of funds or yield miscalculations.
Status: tested and verified ✅

6.4.3 Robust Swap Mechanisms via Uniswap The security review should rigorously inspect the strategy's use of Uniswap V3 swaps (ISwapRouter and IUniswapV3Pool) to move between the vault's primary asset and each Aave token. Special attention must be given to slippage settings, minimum output parameters, and fee calculations, ensuring they cannot be manipulated by malicious actors.
Status: tested and verified ✅

6.4.4 Accurate Distribution of Allocations A comprehensive review is required of the strategy's allocation logic—which uses scoring and utilization data to decide how much capital to place into each Aave token. This allocation must correctly account for minimum and maximum thresholds, ensuring no unintended over-allocation to riskier tokens or under-allocation that could reduce potential returns.
Status: tested and verified ✅

6.4.5 Secure Whitelisting & Emergency Mode Controls The strategy must be scrutinized for its handling of whitelisted addresses (the only addresses allowed to call certain sensitive functions) and the emergency mode switch. The auditor should confirm that only authorized individuals can trigger or revert from emergency mode and that these safeguards cannot be bypassed, ensuring asset safety if the protocol faces unforeseen failures.
Status: tested and verified ✅

7. Executive Summary

Two independent experts from Softstack conducted an unbiased and isolated audit of the smart contract codebase provided by the Fija Finance team. The primary objective of this audit was to verify the security, functionality, and adherence to best practices within the smart contracts. The audit process included a comprehensive manual code review, along with automated security testing to detect any potential vulnerabilities.

The audit identified a total of five issues, classified as follows:

- **No critical issues** were found.
- **No high severity issues** were found.
- **Three medium severity issue** was identified.
- **Two low severity issues** were discovered.
- **No informational issues** were noted.

This audit report provides detailed descriptions for each identified issue, including their severity levels and recommendations for mitigation. Code snippets are included where applicable to illustrate the issues and offer possible fixes. We recommend that the Fija Finance team carefully review these suggestions to improve the security and robustness of their smart contracts.

Update (16.01.2025): We are pleased to report that all identified issues have been thoroughly addressed. All necessary fixes have been implemented to enhance the security and functionality of the system.

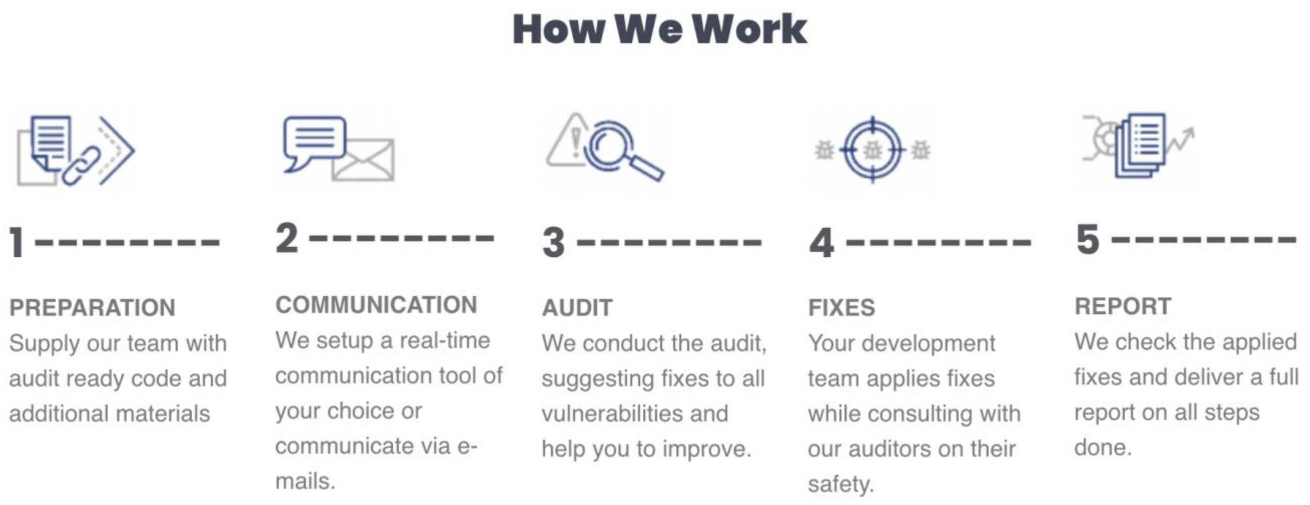
8. About the Auditor

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

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

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

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



How We Work

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

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

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

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

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