SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Javsphere

AUDIT

SECURITY ASSESSMENT

28. September, 2024

FOR

javsphere

SolidProof_io

@solidproof_io

Introduction

SolidProof.io is a brand of the officially registered company MAKE Network GmbH, based in Germany. We're mainly focused on Blockchain Security such as Smart Contract Audits and KYC verification for project teams.
Solidproof.io assess potential security issues in the smart contracts implementations, review for potential inconsistencies between the code base and the whitepaper/documentation, and provide suggestions for improvement.

Disclaimer

SolidProof.io reports are not, nor should be considered, an "endorsement"or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyze.

Project Overview

Summary

Project Name	Javsphere
Website	https://leveragex.trade/
About the project	Javsphere offers the next generation of Decentralized Finance (DeFi) products. Our ecosystem is being built upon the foundation of DeFiChain, a blockchain-based on Bitcoin code that brings native DeFi capabilities to Bitcoin. On top of native DeFi, DeFi Meta Chain (DMC), was added as a functionality upgrade. DMC enables projects to build on the EVM-compatible layer quickly and deploy dApps and Smart Contracts (SC). Through the DMC, DeFiChain connects Bitcoin with Ethereum.
Chain	TBA
Language	Solidity
Codebase	https://github.com/Javsphere/contracts/tree/leverageX/contracts/trade
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/javsphere_start
Twitter	https://x.com/javsphere
Facebook	N/A
Instagram	N/A
GitHub	N/A
Reddit	N/A
Medium	N/A
Discord	https://discord.gg/ssfmAn6D5h
YouTube	https://youtube.com/@javsphere
TikTok	N/A
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Change Log
v1.0	14. August 2024	· Layout Project · Automated/Manual-Security Testing · Summary
v1.1	28. September 2024	· Reaudit

Note – The following audit report presents a comprehensive security analysis of the smart contract utilized in the project that includes outside manipulation of the contract's functions in a malicious way. This analysis did not include functional testing (or unit testing) of the contract/s logic. We cannot guarantee 100% logical correctness of the contract as we did not functionally test it. This includes internal calculations in the formulae used in the contract.

File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
contracts\JavPairsStorage.sol	1cd6ad592dcddfac2d816bd17ec1295601367b01
contracts\JavTradingProcessing.sol	e882029cf74e7b528578509ca48cb5ab434ea0a4
contracts\JavTradingInteractions.sol	72c85c4311c33f42533c486e27fe1859a7835239
contracts\JavTradingStorage.sol	94c1cb7db03bf262de7369e17bba4bbff12e135b
contracts\JavReferrals.sol	5bfbd53b678d4a498701c265b74c7217340d5018
contracts\JavPriceImpact.sol	74977e966c6ed66fb693d3043c7283806799f019
contracts\JavPriceAggregator.sol	36a5a0b5cf123233452fee878a957e5bdd09f7ce
contracts\JavMultiCollatDiamond.sol	d4371da254ba2e14b28181c568969074f11d6e81
contracts\JavLPToken.sol	2945b6294b93d9c9f0975bb1493058b15ae6d447
contracts\JavFeeTiers.sol	da8df7d3ea043dbb6d3da7a64cc58f2471ca0d1d
contracts\JavBorrowingProvider.sol	7680dcf7266359de89699c81deac380c132b23d5
contracts\JavBorrowingFees.sol	9d066edc83077b1c8ef22334125bcad0e84c3a7f
contracts\abstract\JavDiamondStorage.sol	d3cc1b4a3eba1de872637ce2766a75e10840691a
contracts\abstract\JavDiamondLoupe.sol	b44e0a886f8802b0b5f6a7f51f80969c65bd3523
contracts\abstract\JavDiamondCut.sol	ff4ef664762e48c7b55a208d7d1e8123cbbd55b2
contracts\abstract\JavAddressStore.sol	52af4eb947bd02171f881721eaeb3209f2a5c9f8
contracts\Base\BaseUpgradable.sol	34acb73db0dcb3fdb24a73a08be1a1fdf09e4032
contracts\libraries\DiamondUtils.sol	cc4e1bd80332893233a2a49ddda90155874e79b3
contracts\libraries\StorageUtils.sol	343d2f87743d6b0b94e440baf4bdb826f0ec6a57
contracts\libraries\BorrowingFeesUtils.sol	a06bdf19d5581197086088dd459889c1a1d01804
contracts\libraries\ChainUtils.sol	732b91b7be1695ada9ce51e729971851b7b0a7c3
contracts\libraries\PriceAggregatorUtils.sol	155e0755c17cc2aa23cdbba9114e399fdf95c7a5
contracts\libraries\FeeTiersUtils.sol	d54c8ee0436db46fc739193e231fc9d84f401022
contracts\libraries\PairsStorageUtils.sol	8e3c7eabf6014c77ac362e36f998ee428dfbe1b7
contracts\libraries\PriceImpactUtils.sol	d07c66b50dc4023e465bc3b9406f4921c6516f35

File Name	SHA-1 Hash
contracts\libraries\ReferralsUtils.sol	ab4c9b5115114a295ee6537fdf76bbf23c046c23
contracts\libraries\TradingInteractionsUtils.sol	322a0bf04731ca15a3ffa8c9e0076c50ec202f7c
contracts\libraries\TradingProcessingUtils.sol	6fdae1c747bb37f8d60420c1dadec2c5685920bf
contracts\libraries\TradingStorageUtils.sol	1766ac5b3753720aebdb54096f08667ea39d764d
contracts\helpers\LPProvider.sol	ea4c0277fd1d0cd26103319c7642203b54637296

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

Imported packages.

Used code from other Frameworks/Smart Contracts.

Dependency / Import Path	Count
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	5
@openzeppelin/contracts/utils/math/Math.sol	1

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way.

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract'sstate. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be needed within visibility modifier, such as public, private or internal, which determines the access level of the variable.

Components

📝 Contracts	📚 Libraries	🔍 Interfaces	🎨 Abstract
14	12	0	4

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
204	9

External	Internal	Private	Pure	View
193	461	11	32	225

StateVariables

Total	🌐 Public
55	13

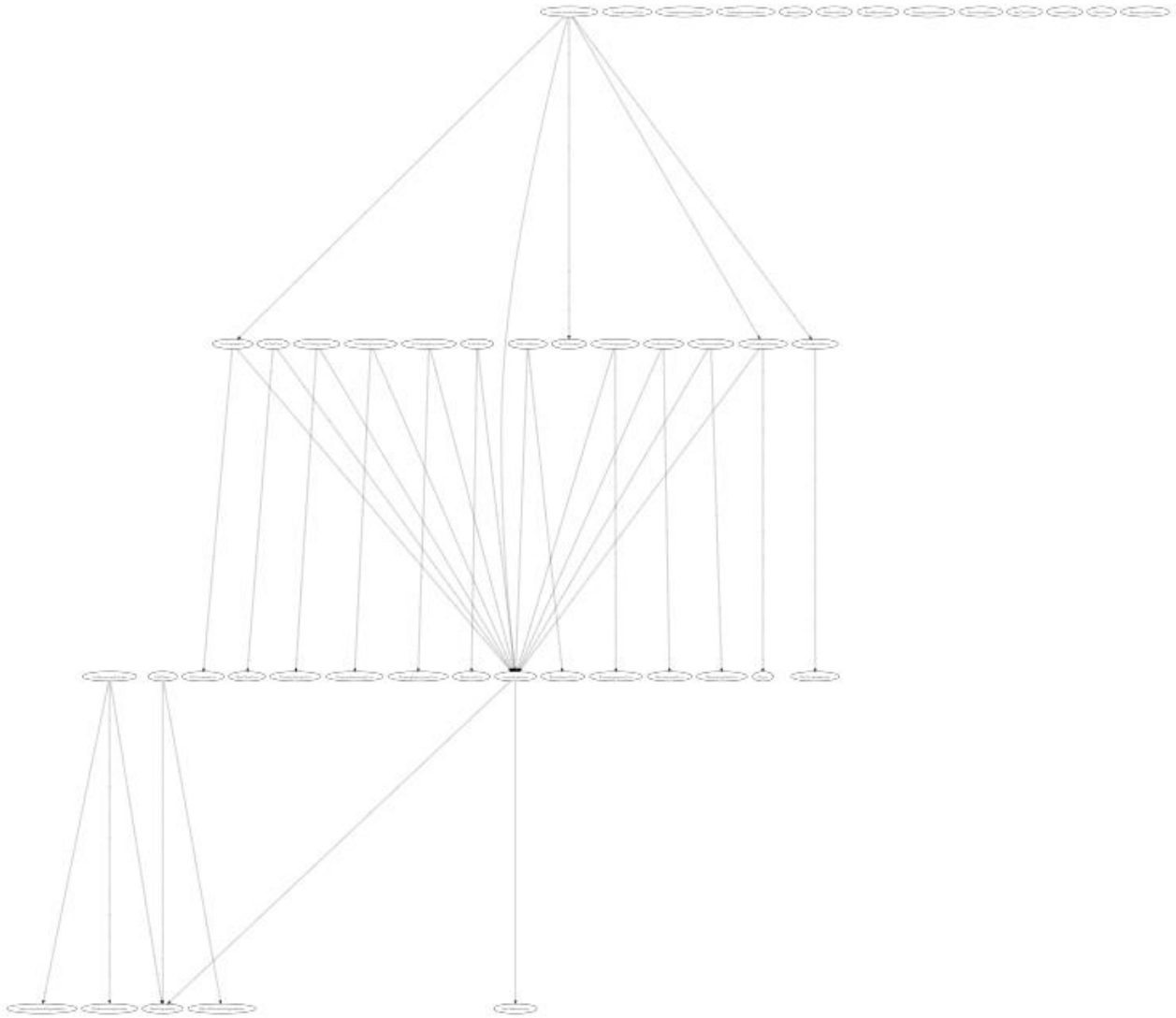
Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
`^0.8.23` `^0.8.16`	`----------`	`Yes`	`yes` (13 asm blocks)	`----------`

Transfers ETH	Low-Level Calls	Delegate Call	Uses Hash Functions	ECRecover	New/Create/Create2
`yes`		`yes`			

Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methodsfrom another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit the vulnerability and the impact of that event on the organization or system.The 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 aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executingthe 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

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers andsmart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   a. Reviewing the specifications, sources, and instructions provided to
   SolidProof to ensure we understand the size, scope, and functionality of the
   smart contract.
   b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
   c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.

2. Testing and automated analysis that includes the following:
   a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
   b. Symbolic execution, which is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.

4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

Overall Security

Upgradeability

Contract is not upgradable	✗ Deployer can update the contract with new functionalities.
Description	The contract is an upgradeable contract. The Deployer is not able to change or add any functionalities to the contract after deploying.
Comment	The contract contains the functionality in which the deployer of the contract can update and deploy the new version of the contract.

Ownership

Contract ownership is not renounced.	❌ **The ownership is not renounced.**
Description	The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including: • Centralizations • The owner has significant control over contract's operations.
Comment	N/A

Note – *The contract cannot be considered as renounced till it is not deployed or having some functionality that can change the state of the contract.*

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refer to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who has the ability to add new tokens to the network's total supply.

Contract owner can mint new tokens.	❌ **The owner can mint new tokens.**
Description	The owner is able to mint new tokens once the contract is deployed.
Comment	The borrowing provider in the contract is able to mint an unlimited amount of new tokens after initial deployment, which is not recommended as there must be a threshold limit so that the tokens cannot be minted to an indefinite number. Also, the Jav Token contains the functionality in which the owner of the contract can mint an unlimited amount of tokens from the contract to any particular wallets.

File/Line(s): L40-42

Codebase: LLPToken.sol

```
ftrace | funcSig
function mint(address account, uint256 amount) external onlyBorrowingProvider {
    _mint(account, amount);
}
```

Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens	✅ **The owner cannot burn tokens.**
Description	The owner is not able burn tokens without any allowances.
Comment	N/A

Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses.	✅ **The owner cannot blacklist wallets.**

Description	The owner cannot blacklist wallets.
Comment	N/A

Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort indeveloping and maintaining the contract.

Contract owner can levy high taxes	✗ **The owner can set fees more than 25%.**
Description	The owner can set fees of more than 25%.
Comment	The contract contains the functionality in which the fees can be set to more than 25% in the contract which is not recommended as high fees can cause the loss of funds for the users. It is recommended that no fees in the contract should be more than 25% in the contract. Also, the vault closing fees in the contract can be set to 100% which is not recommended.

File/Line(s): L52-58, L63-69, L74-80
Codebase: ReferralUtils.sol

```
ftrace | funcSig
function updateAllyFeeP(uint256 _value) internal {
    if (_value > MAX_ALLY_FEE_P) revert IGeneralErrors.AboveMax();

    _getStorage().allyFeeP = _value;

    emit IReferralsUtils.UpdatedAllyFeeP(_value);
}

/**
 * @dev Check IReferralsUtils interface for documentation
 */
ftrace | funcSig
function updateStartReferrerFeeP(uint256 _value) internal {
    if (_value > MAX_START_REFERRER_FEE_P) revert IGeneralErrors.AboveMax();

    _getStorage().startReferrerFeeP = _value;

    emit IReferralsUtils.UpdatedStartReferrerFeeP(_value);
}

/**
 * @dev Check IReferralsUtils interface for documentation
 */
ftrace | funcSig
function updateReferralsOpenFeeP(uint256 _value) internal {
    if (_value > MAX_OPEN_FEE_P) revert IGeneralErrors.AboveMax();

    _getStorage().openFeeP = _value;

    emit IReferralsUtils.UpdatedOpenFeeP(_value);
}
```

File/Line(s): L57-63

Codebase: TradingProcessingUtils.sol

```
ftrace | funcSig
function updateVaultClosingFeeP(uint8 _valueP↑) internal {
    if (_valueP↑ > 100) revert IGeneralErrors.AboveMax();

    _getStorage().vaultClosingFeeP = _valueP↑;

    emit ITradingProcessingUtils.VaultClosingFeePUpdated(_valueP↑);
}
```

Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When token or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Contract owner can lock function.	❌ **The owner can lock function.**
Description	The owner can lock the functions.
Comment	The smart contract includes a pausable feature that allows the contract administrator to temporarily halt operations if any issues arise, ensuring that the contract's functionality remains secure. Additionally, a lock period can be set to prevent the contract from being indefinitely restricted. Furthermore, the manager has the ability to pause or resume the sale of JLP tokens as needed to prevent any disruptions, ensuring that the tokens remain secure in the event of a functionality issue.

Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example,if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project, there are authorities that have access to the following functions:

File	Privileges
JavAddressStore	➢ The role manager can set roles in the contract.
JavDiamondCut	➢ The role manager can add, replace, or remove functions from the facets.
JavBorrowingFees	➢ The manager can set the borrowing pair parameters in the contract. ➢ The manager can update the borrowing group parameter in the contract.
JavBorrowingProvider	➢ The admin can add new tokens to the contract. ➢ The admin can initially buy the JLP tokens. ➢ The owner can update the PNL handler in the contract. ➢ The PNL handler in the contract can send assets to the receiver.
JavFeeTiers	➢ The governor can set the group indices and volume multiplier in the contract. ➢ The governor can add fee tiers in the contract.
JavLPToken	➢ The owner can update any arbitrary address as the borrowing provider address in the contract including zero or dead address. ➢ The borrowing provider can mint an unlimited amount of tokens to any arbitrary address.
JavPairsStorage	➢ The governor can add new pairs to the contract. ➢ The governor can update the existing pairs in the contract. ➢ The governor can add or update groups in the contract. ➢ The governor can add or update the pair's fees in the contract. ➢ The manager can update the maximum leverage value in the contract.

JavPriceAggregator	➢ The governor can update the collateral USD Price feed value in the contract. ➢ The admin can add or remove signers from the contract. ➢ The admin can update the price update fees in the contract.
LPProvider	➢ The admin can update any arbitrary address as the bot address in the contract. ➢ The admin can update the rewards distributor address in the contract. ➢ The admin can update any arbitrary address as the WDFI token address. ➢ The admin can swap tokens ETH to WDFI and WDFI to DFI tokens. ➢ The admin can withdraw tokens from the contract. ➢ The admins can withdraw DFI tokens from the contract.
JavPriceImpact	➢ The governor can update the windows count to not more than 5 in the contract. ➢ The governor can update the price impact windows duration in between 10 minutes to 30 days in the contract. ➢ The manager can update the pair depths.
JavReferrals	➢ The governor can update the Ally fee, Start referrer fee, update referral open fees. ➢ The governor can update the referrer target volume in USD value in the contract. ➢ The governor can whitelist/un-whitelist allies in the contract. ➢ The governor can whitelist/un-whitelist referrers in the contract.
JavTradingInteractions	➢ Any arbitrary address can open and close trades for themself. ➢ Any arbitrary address can update the open orders for themself.
JavTradingProcessing	➢ The governor can update the vault closing fees. ➢ The governor can claim pending fees.
JavTradingStorage	➢ The governor can activate trading. ➢ The governor can add collaterals to the contract. ➢ The governor can active/inactive the state of

BaseUpgradable

collaterals.

- The admin can pause and un-pause the contract.
- The admin can set the admin address.

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g. Gnosis Safe
- Use of a timelock at least with a latency of e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement
- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.

Audit Result

Critical Issues

No critical issues

High Issues

No high issues

Medium Issue

#1 | The owner can lock selling.

File	Severity	Location	Status
JavBorrowingProvider.sol	Medium	L159-171	Open

Description – The contract contains the functionality in which the manager has the ability to pause or resume the sale of JLP tokens as needed to prevent any disruptions, ensuring that the tokens remain secure in the event of a functionality issue. The contract can have a locking period in the contract so that the functionality of the contract should not be locked for an indefinite period.

Low Issue

#1 | Floating pragma solidity version.

File	Severity	Location	Status
All	Low	--	Open

Description — Adding the constant version of Solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs

#2 | Missing zero or dead address check.

File	Severity	Location	Status
BaseUpgradable.sol	Low	L41-45	Open
JavPriceAggregator.sol	Low	L57-67	Open
LPProvider.sol	Low	L72-76, L78-82	Open

Description – It is recommended to check that the address cannot be set to zero or dead address.

#3 | Remove math library.

File	Severity	Location	Status
JavBorrowingProvider.sol	Low	--	Open

Description – The compiler version above 0.8.0 has the ability to control arithmetic overflow/underflow. It is recommended to remove the unwanted code in order to avoid high gas fees.

Informational Issue

#1 | Natspec documentation missing.

File	Severity	Location	Status
All	Informational	--	Open

Description – The documentation generated from the Natspec format typically includes information about the functions and variables in the contract, including descriptions of their functionality and usage, as well as details about their input and output parameters. It may also include other information about the contract, such as its overall purpose and intended use. It is recommended to comment on your code to describe the functionality of the code.

Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY