cSIGMA

cSigma Finance

SMART CONTRACT AUDIT REPORT

IMMUNEBYTES

31 MAY 2024

Introduction

About cSigma Finance

Launched in 2023, cSigma protocol connects borrowers and lenders with excess capital globally in a low-friction and high-transparency process. Using AI, the protocol systematizes the critical steps of the lending process and offers in-built credit rating, pricing, and risk management optimized for Private Credit. It handles capital movements and on-chain accounting and settlement in a decentralized manner and enables 3rd party underwriters and risk assessors to participate in the protocol. It empowers credit pool operators, which are typically growing businesses in various verticals, to originate loans with their existing SMB borrower relationships, plug their underwriting and domain expertise into the protocol, and make loan pools available to lenders globally.

Visit https://csigma.finance/ to learn more about it.

About ImmuneBytes

ImmuneBytes is a security start-up that provides professional services in the blockchain space. The team has hands-on experience conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and understand DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has secured 255+ blockchain projects by providing security services on different frameworks. The ImmuneBytes team helps start-ups with detailed system analysis, ensuring security and managing the overall project.

Visit http://immunebytes.com/ to learn more about the services.

Documentation Details

The team has provided the following doc for audit:

1. https://immunebytes.notion.site/ImmuneBytes-Requirement-Document-Csigma-Finance-7b985a251f3b422881263d56b57b0277

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include

    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project, starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is a structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

Project Name	cSigma Finance
Platform	EVM
Languages	Solidity
GitHub Link	https://github.com/csigma-labs/csigma-protocol/tree/main
Commit - Final Audit	e52e380883fd2184af4c36bf046e94fcab5437b9
Platforms & Tools	Remix IDE, Truffle, VScode, Contract Library, Slither, SmartCheck, Fuzz

Security Review Lifecycle for cSigma Finance

1. **Initial Reconnaissance and Planning**

    ○ **Scope Definition**: The scope was defined based on the provided technical documentation, whitepaper, and codebase, focusing on critical smart contracts and their interactions.
    ○ **Objective Setting**: Establish the primary goals of the audit, including identifying security vulnerabilities, ensuring compliance with best practices, and validating the functionality against the documented specifications.

2. **Threat Modeling**

    ○ **Threat Identification**: Identified potential threat vectors, including both external and internal threats, through brainstorming and expert analysis.
    ○ **Risk Assessment**: Assessed the impact and likelihood of identified threats, prioritizing them based on their severity.
    ○ **Mitigation Strategies**: Developed and recommended strategies to mitigate identified threats, enhancing the overall security posture of the protocol.

3. **Tracking User Funds and Security**

    ○ **Fund Flow Analysis**: Analyzed the flow of funds within the protocol to ensure secure handling of user assets.
    ○ **Secure Storage Verification**: Verified that funds are securely stored and that proper mechanisms are in place to prevent unauthorized access.
    ○ **Withdrawal Processes**: Ensured that withdrawal processes are secure and only authorized users can initiate and complete withdrawals.

4. **Access Control Enforcement**

    ○ **Role Verification**: Reviewed access control mechanisms to ensure that roles are correctly defined and enforced.
    ○ **Permission Checks**: Ensured that sensitive functions are protected and can only be accessed by authorized roles.
    ○ **Audit Trails**: Verified that actions performed by users are logged and can be audited to detect and respond to unauthorized activities.

5. **Automated Code Review**

    ○ **Tool Selection**: Utilized industry-standard tools such as MythX, Slither, and Oyente to perform initial automated analysis.
    ○ **Static Analysis**: Conducted static code analysis to detect common vulnerabilities, such as reentrancy, overflow/underflow, and unchecked external calls.
    ○ **Report Generation**: Generated preliminary reports highlighting potential issues for further manual review.

6. **Manual Code Review**

    ○ **Line-by-Line Analysis**: Performed a thorough manual review of the smart contract code to identify subtle bugs, logical errors, and complex vulnerabilities that automated tools might miss.
    ○ **Functionality Verification**: Ensured that the implementation matches the described functionality in the technical documentation and whitepaper.
    ○ **Best Practices Check**: Verified adherence to Solidity best practices, including proper use of design patterns, modularity, and efficient gas usage.

7. **User Experience**
    ○ **Documentation Review**: Compared the technical documentation and whitepaper with the implemented code to identify discrepancies and ensure accuracy.

System Architecture Overview

The cSigma protocol V1 is built using a multi-facet proxy smart contract system based on the ERC-2535 diamond proxy standard. This architecture allows for modular and upgradable smart contracts, providing a robust and flexible foundation for the protocol. The key components of the system and their interactions are described below:

Core Components and Facets

1. **Diamond Proxy (ERC-2535)**

    ○ The diamond proxy pattern is used to compose multiple contracts into a single logical contract. It acts as a dispatcher, routing function calls to the appropriate facet based on a given selector. This enables adding new functionality through separate contracts without disrupting the existing contract structure.

2. **Facets**

    ○ **DiamondCutFacet**: Manages the addition, modification, or removal of facets within the contract. It enables seamless upgrades while preserving the current state of the contract.

    ○ **DiamondLoupeFacet**: Provides insight into the various facets and functions present in the contract. It allows developers to query and understand the available functionalities, enhancing contract visibility and aiding in effective development and debugging.

    ○ **OwnershipFacet**: Manages ownership and access control within the contract. It includes functions to assign and transfer ownership, facilitating permissions for administrative actions.

    ○ **AccessControlFacet**: Manages roles and features for access control, ensuring that only authorized users can perform sensitive actions.

    ○ **LenderFacet**: Maintains a record of lenders' on-chain attributes, including lenderId, userId, hash, country, wallet address, KYB status, onboarding time, invested poolIds, and paymentIds.

- **PoolManagerFacet**: Manages the attributes related to pool managers, including poolManagerId, userId, hash, country, wallet address, KYB status, onboarding time, created poolIds, and paymentIds.
- **CreditPoolFacet**: Handles credit pool on-chain attributes and manages the lifecycle of credit pools.
- **VaultFacet**: Manages the internal accounts of lenders and pools, facilitating the flow of funds from lender to pool, pool to pool manager, pool manager to pool, and pool to lender.
- **PaymentFacet**: Keeps records of all payment-related transactions.
- **MetadataFacet**: Provides essential protocol information.
- **DistributeExtension**: Manages the distribution of rewards and incentives within the protocol.
- **StableCoinExtension**: Handles stablecoin-related functionalities within the protocol.

Functional Overview

1. **Lending Flow**
    - Lenders deposit USD-T into the protocol's vault, which manages the lender's internal account with the deposited amount.
    - Lenders can request to invest tokens in active pools. Once approved by the protocol, the amount is invested in the specified pool.
2. **Borrowing Flow**
    - Pool managers raise requests to receive USD-T invested in their pools. Upon approval, USD-T is transferred to the pool manager's externally owned account (EOA).
    - Pool managers are responsible for paying back the principal and coupon once the grace period is over.
3. **Role Management**
    - **Lender**: Provides funds to the protocol, receives interest, and can invest in various pools.
    - **Pool Manager**: Oversees the movement of tokens into credit pools, manages investments, and ensures repayment.
    - **Vault**: Serves as a secure repository for safeguarding funds contributed by lenders, enhancing confidence in the system's reliability.
4. **Data Handling and Security**
    - **On-chain Data**: Used for public data that needs to be fully transparent and immutable. Only specific and essential data models are stored on-chain to reduce costs and ensure security.
    - **IPFS Storage**: Utilized for storing non-sensitive public data, offering flexibility and a degree of immutability. The content hash of the files is stored on-chain to verify data integrity.
    - **Private Database**: Stores sensitive data in an encrypted format, ensuring confidentiality and security.

The cSigma protocol's system architecture is designed to provide a secure, flexible, and scalable platform for decentralized lending and borrowing. The protocol ensures modularity and upgradability by leveraging the ERC-2535 diamond proxy standard, enabling continuous improvement and adaptation to evolving requirements. The comprehensive role management, secure data handling, and robust governance framework collectively contribute to the protocol's reliability and effectiveness in facilitating decentralized financial services.

Project Goals

1. **Access Control and Authorization**
    - Is there any way to revoke the RBAC for authorized users to steal user funds or bring loss to the protocol in any way?
    - Are the roles and permissions correctly enforced to ensure that only authorized users can execute critical functions like transferring funds, creating pools, or updating sensitive parameters?
    - Are there mechanisms in place to prevent unauthorized elevation of privileges within the protocol?
2. **Funds Security and Management**
    - Is the flow of funds from lender to pool, pool to pool manager, pool manager to pool, and pool to lender secure and free from potential vulnerabilities that could lead to theft or mismanagement?
    - Are the internal accounts managed by the VaultFacet properly secured to prevent unauthorized access or modification of balances?
3. **Smart Contract Logic and Integrity**
    - Are there any logical errors or vulnerabilities in the implementation of the diamond proxy pattern that could be exploited to disrupt the protocol's functionality?
    - Are the facets correctly integrated and do they interact securely with each other, ensuring that the modular design does not introduce new attack vectors?
4. **Data Integrity and Storage**
    - Is the data stored on-chain and in IPFS correctly hashed and validated to ensure that it has not been tampered with?
    - Does the protocol correctly handle sensitive user data, ensuring that it is encrypted and stored securely to prevent unauthorized access?
5. **Transaction and Payment Security**
    - Are all payment-related transactions within the PaymentFacet accurately recorded and free from potential double-spending or tampering vulnerabilities?
    - Is the process for managing and recording lender and pool manager payments robust and secure, preventing any unauthorized modifications?

6. **Protocol Upgradability and Governance**

    - Is the DiamondCutFacet implementation secure and does it prevent unauthorized upgrades or modifications to the protocol?
    - Are the governance mechanisms robust enough to ensure that protocol changes are made transparently and with proper authorization?

7. **Threat Modeling and Risk Management**

    - Have potential threat vectors been identified and mitigated, ensuring that both internal and external threats are addressed?
    - Does the protocol have mechanisms to monitor and respond to evolving threats, ensuring continuous security?

8. **Compliance and Regulatory Considerations**

    - Does the protocol enforce KYB/AML compliance effectively, ensuring that all participants are properly vetted and monitored?
    - Are there any potential regulatory risks associated with the protocol's operation, particularly regarding the handling of user funds and personal data?

9. **Performance and Scalability**

    - Are the smart contracts optimized for gas efficiency, ensuring that transactions are cost-effective for users?
    - Can the protocol handle an increase in users and transactions without compromising security or performance?

10. **Incident Response and Recovery**

    - Does the protocol have mechanisms in place for incident response, including the ability to quickly address and recover from security breaches?
    - Are there fallback procedures to ensure the continuity of operations in case of a smart contract failure or other critical issues?

11. **User Experience and Usability**

    - Are the user interfaces and interactions designed to prevent common user errors, such as sending funds to incorrect addresses or making invalid transactions?
    - Does the protocol provide clear and transparent information to users regarding their actions and the state of their funds?

12. **Integration and Interoperability**

    - Is the protocol compatible with major DeFi platforms and standards, ensuring seamless integration with other systems and services?
    - Are there any interoperability issues that could prevent the protocol from functioning as intended within the broader DeFi ecosystem?

13. **Economic and Financial Security**

    - Are the economic models, such as APR setting and yield distribution, designed to prevent abuse and ensure fair returns for all participants?
    - Are there any vulnerabilities in the financial logic that could be exploited for economic gain at the expense of the protocol or its users?

14. **Audit Trails and Transparency**

    - Does the protocol provide comprehensive audit trails for all critical actions, enabling thorough review and analysis of any incidents?
    - Are all relevant actions and transactions logged and accessible for audit purposes, ensuring transparency and accountability?

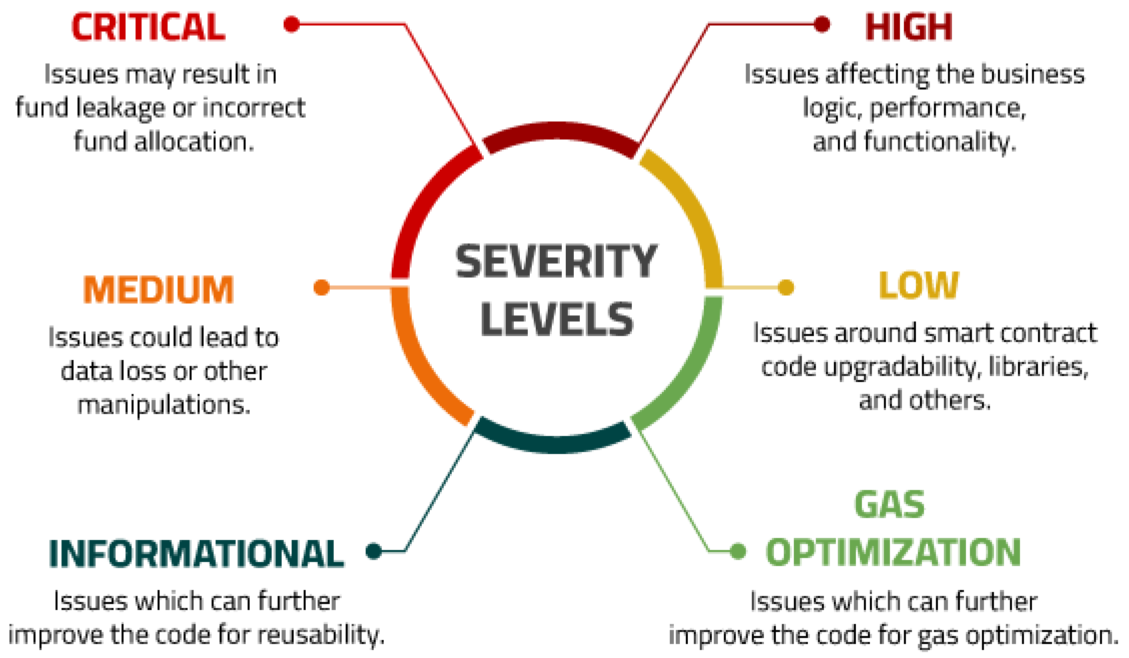
15. **Security of External Integrations**

    - Are external integrations, such as oracles and third-party services, securely managed and free from vulnerabilities that could compromise the protocol?

Does the protocol regularly audit and update its integrations to ensure they remain secure and reliable?

Security Level References

Every issue in this report was assigned a severity level from the following:


CRITICAL
Issues may result in fund leakage or incorrect fund allocation.
HIGH
Issues affecting the business logic, performance, and functionality.
SEVERITY LEVELS
MEDIUM
Issues could lead to data loss or other manipulations.
LOW
Issues around smart contract code upgradability, libraries, and others.
INFORMATIONAL
Issues which can further improve the code for reusability.
GAS OPTIMIZATION
Issues which can further improve the code for gas optimization.

Severity Status References

OPEN

The issue hasn't been fixed as per our recommendation(s) and still stays pertinent.

FIXED

The issue has been agreed upon to be true and fixed as per our recommendation(s).

ACKNOWLEDGED

The issue is an intended behaviour and doesn't require a change as per our recommendation(s).

REDACTED

The reported issue is invalid, confirmed to be a false positive after communication with dev team.

Severity Status

Issues	Critical	High	Medium	Low	Informational	Gas
Open	-	-	-	-	-	-
Fixed	-	-	-	-	-	-
Redacted	-	-	-	-	-	-
Acknowledged	-	-	-	2	1	-

Finding

#	Findings	Risk	Status
1	Inflexible Domain Separator Setting Mechanism	Low	Acknowledged
2	Missing Input Validation	Low	Acknowledged
3	Insufficient Events Emission in Critical Functions	Informational	Acknowledged

Critical Severity Issues

No issues were found.

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

Issue	**Inflexible Domain Separator Setting Mechanism**
Location	\`setDomainSeperator()\` function in \`DistributeState\` contract inside contracts/facets/DistributeExtension.sol file
Description	The \`setDomainSeperator()\` function currently sets a static EIP-712 domain separator using hardcoded values for the contract name and version. This approach lacks flexibility, preventing dynamic updates to the domain separator, which might be necessary for future versions or modifications of the contract.
Impact	The inflexible domain separator setting mechanism can lead to the following issues: ● Maintenance Challenges: Future updates or versioning of the contract may require changes to the domain separator. The current hardcoded values necessitate redeployment of the contract to reflect such changes. ● Reduced Adaptability: The inability to update the domain separator dynamically limits the contract's adaptability to evolving requirements or standards. ● Operational Constraints: Any required change in the domain separator will involve significant operational overhead, including redeployment and potential downtime.

Code Affected	`function setDomainSeperator() internal {`
	`    AccessControlLib.enforceIsConfigManager();`
	`    DistributeState storage distributeState = diamondStorage();`
	`    distributeState.domainSeperator = keccak256(`
	`        abi.encode(`
	`            EIP712_DOMAIN_TYPEHASH,`
	`            keccak256(bytes("cSigmaDiamond")),`
	`            keccak256(bytes("1")),`
	`            getChainId(),`
	`            address(this)`
	`        )`
	`    );`
	`}`

Recommendation	Refactor the \`setDomainSeperator()\` function to accept the contract version as an input parameter. This change will enable dynamic updates to the domain separator without requiring contract redeployment.

	Suggested Code Change:
	`function setDomainSeperator(string memory version) internal {`
	`    AccessControlLib.enforceIsConfigManager();`
	`    DistributeState storage distributeState = diamondStorage();`
	`    distributeState.domainSeperator = keccak256(`
	`        abi.encode(`
	`            EIP712_DOMAIN_TYPEHASH,`
	`            keccak256(bytes("cSigmaDiamond")),`
	`            keccak256(bytes(version)),`
	`            getChainId(),`
	`            address(this)`
	`        )`
	`    );`
	`    emit DomainSeperatorUpdated(distributeState.domainSeperator);`
	`}`

Developer Response	• The name and version are specific and bound to the contract deployed and verified. These values are constant and not meant to be changed. These tags are only changed when there is a change in the contract code. If there is a change in the contract code, then it needs to be deployed. Hence, impacts like operational constraints and maintenace challenges do not apply in this case. • The only possible variable components are chainId and contract address. chainId advise changed only when hard fork happens.
Status	**Acknowledged**

Issue	**Missing Input Validation**

Location	contracts/facets/VaultFacet.sol
	setMinDepositLimit(uint256 _limit)
	contracts/facets/PoolManagerFacet.sol
	updatePoolManagerWallet(string calldata _poolManagerId, address _wallet)

Description	The functions setMinDepositLimit in VaultFacet and updatePoolManagerWallet in PoolManagerFacet lack proper input validation. Input validation is crucial to ensure that the provided values meet expected constraints and to prevent potential security vulnerabilities or logical errors.
Impact	The lack of input validation can lead to the following issues: ● Security Risks: Invalid inputs could be exploited by malicious actors to perform unintended actions or to disrupt the system. ● Logical Errors: Without validation, invalid or unexpected inputs can cause the contract to behave incorrectly or produce unintended outcomes. ● Operational Issues: Improper inputs can lead to states that are difficult to manage or recover from, increasing the operational complexity

Proof of Concept	**VaultFacet:** The setMinDepositLimit function does not validate the _limit parameter. There should be a check to ensure the limit is within a reasonable range.
	
	function setMinDepositLimit(uint256 _limit) external {
	  return VaultLib.setMinDepositLimit(_limit);
	}
	function setMinDepositLimit(uint256 _limit) internal {
	  AccessControlLib.enforceIsConfigManager();
	  VaultState storage vaultState = diamondStorage();
	  vaultState.minDepositLimit = _limit;
	}
	
	PoolManagerFacet: The updatePoolManagerWallet function does not validate the _wallet address. There should be a check to ensure the address is not zero.
	
	function updatePoolManagerWallet(
	    string calldata _poolManagerId,
	    address _wallet
	  ) external {

	```
	    address _prevWallet = PoolManagerLib.getPoolManagerWallet(
	        _poolManagerId
	    );
	    PoolManagerLib.updatePoolManagerWallet(_poolManagerId, _wallet);
	    emit UpdatePoolManagerWalletEvent(_poolManagerId, _prevWallet, _wallet);
	}

	function updatePoolManagerWallet(
	        string calldata _poolManagerId,
	        address _wallet
	    ) internal {
	    AccessControlLib.enforceIsEditManager();
	    enforceIsPoolManagerIdExist(_poolManagerId);
	    PoolManagerState storage poolManagerState = diamondStorage();
	    poolManagerState.poolManagers[_poolManagerId].wallet = _wallet;
	```
Recommendation	Add input validation to both functions to ensure the inputs are valid and within acceptable ranges.
Developer Response	● It agreed that these functions missed input validation. However, these functions are restricted access functions and meant to be called by wallet with different role. ● The chance of error in providing wrong inputs is a bit low in this case. On the other hand, we benefit from saving some gas costs by not adding these validations. ● I think we can leave with it as of now.
Status	**Acknowledged**

Informational

Issue	Insufficient Events Emission in Critical Functions
Location	**AccessControlFacet** initializeAccessControl, updateRole, updateFeatures **CreditPoolFacet** removeCreditPoolPaymentId, removeCreditPoolPaymentIdByIndex **LenderPoolFacet** removeLenderPaymentId, removeLenderPaymentIdByIndex **PoolManagerFacet** removePoolManagerPaymentId, removePoolManagerPaymentIdByIndex **VaultFacet** setMinDepositLimit
Description	Several critical functions within the smart contract lack event emissions. Events are crucial for off-chain systems to track and monitor state changes and operations within the contract. The absence of events in these functions hampers off-chain traceability, making it difficult to audit and monitor the contract's activity.
Impact	The missing events result in: ● Reduced Transparency: Off-chain systems cannot track changes made by these functions, reducing the transparency of the contract's operations. ● Auditability Issues: Without event logs, it becomes challenging to audit and verify the actions performed by these functions. ● Monitoring Difficulties: Automated systems that rely on event logs for monitoring contract activities will fail to detect changes made by these functions.
Recommendation	Emit relevant events in the identified functions to enhance off-chain traceability and monitoring.
Developer Response	initializeAccessControl is called only one time. Hence, no need to track any event for it. `RoleUpdated` event is emitted whenever updateRole and updateFeature function triggers. The remove functions are emergency kind of functions and not meant to be called frequently.
Status	Acknowledged

Concluding Remarks

While conducting the audits of cSigma Finance, it was observed that the contracts contain Low-severity issues, along with a few recommendations.

Our auditors suggest that the developers resolve the issue of low severity. The recommendations will improve the smart contract's operations.

Note:
The cSigma team has acknowledged the low issues, and no critical/high vulnerabilities were found during the audit.

Disclaimer

ImmuneBytes' audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program complementing this audit is strongly recommended.

Our team does not endorse the cSigma Finance platform or its product, nor is this audit investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for code refactoring by the team on critical issues.

IMMUNEBYTES

AUDITS