



HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Zharta

Date: Jan 23 2023

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Zharta
Approved By	Evgeniy Bezuglyi SC Audits Department Head at Hacken OU
Type	Lending platform
Platform	EVM
Language	Vyper
Methodology	Link
Website	https://www.zharta.io
Changelog	09.12.2022 - Initial Review 06.01.2023 - Second Review 20.01.2023 - Third Review 23.01.2023 - Fourth Review



Table of contents

Introduction	4
Scope	4
Severity Definitions	6
Executive Summary	7
Checked Items	9
System Overview	12
Findings	17
Disclaimers	31

Introduction

Hacken OÜ (Consultant) was contracted by Zharta (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository	https://github.com/Zharta/protocol-v1
Commit	8e1875d7bcc14f8d56c20ac100517def469dd63b
Functional Requirements	Link
Technical Requirements	Link
Contracts	<p>File: ./contracts/CollateralVaultCore.vy SHA3: 6cf4bda51a502dd0234f640763e2d5de0b7ea6c8427ca011bfe015512bae4188</p> <p>File: ./contracts/CollateralVaultPeripheral.vy SHA3: 599798a4b4e29693878b94d41e831c5a9bb20b351e701ffee1acbca8341f2fb5</p> <p>File: ./contracts/LendingPoolCore.vy SHA3: 2cf2c81d76fe59bdda32f071a66ad351965e0982a0064d8a0f6e60c4d2730cd1</p> <p>File: ./contracts/LendingPoolPeripheral.vy SHA3: 9c3d055f932d831a5ab2e5ee4a6bf5d9f684b03eaf4cb0f5d4580857007bf42c</p> <p>File: ./contracts/LiquidationsCore.vy SHA3: e9b6afe0d789348c7a1c634ed718f485c972dc6f229e47453d6cd2b6b66ee457</p> <p>File: ./contracts/LiquidationsPeripheral.vy SHA3: 350b0c7aaabd15943ec8ab6b32707ffc7f54166b493c74531b81a93deea33f4f</p> <p>File: ./contracts/LiquidityControls.vy SHA3: 4b08b008241e6a4ee61b0a6e52e756bb7a7c1c19e19b25b45854aed8551ff16d</p> <p>File: ./contracts/Loans.vy SHA3: f0da410caaa762a02969b5d370413031d9277bea26ea33238c14b9ceb0345d4d</p> <p>File: ./contracts/LoansCore.vy SHA3: fde2e2401664fea2bad480f3d476524c0288358a270de04e8bc8f331c3040b</p> <p>File: ./interfaces/ICollateralVaultCore.vy SHA3: 4edc7e1ce9d65148ac318159b34046902b358b011b33167a8e55baca1459eb2</p> <p>File: ./interfaces/ICollateralVaultPeripheral.vy SHA3: de3fa0a4684ecab0351ec1f7a80bff8c0f5069f9c3c8a4ccd93ea47dbe77f971</p> <p>File: ./interfaces/ILendingPoolCore.vy SHA3: ddbef7f603eea76ab569d495bdcabb79467d95c2893269ca0613296d6849d748</p> <p>File: ./interfaces/ILendingPoolPeripheral.vy SHA3: 11e8f2b716f2d60210847913ec5c48830e575e3e1e86ee9a06d277b189c27fde</p> <p>File: ./interfaces/ILiquidationsCore.vy SHA3: b69b4978adb887d85612e5d77060ffead103dca2eb79587c315bf7af91cc9601</p>

	<p>File: ./interfaces/ILiquidationsPeripheral.vy SHA3: 53c5e2ef492a6e9b8cbce3c17710628958a638c7debdcb17cb9544fe621636d2</p> <p>File: ./interfaces/ILiquidityControls.vy SHA3: 7ab19761c877bc1476e5d549389149495c2228005b70ee348b8af4f6c37af183</p> <p>File: ./interfaces/ILoans.vy SHA3: a9f7da67e59f2e486532894312c9c1c978d0229c69f9fb2d52d01ff585756437</p> <p>File: ./interfaces/ILoansCore.vy SHA3: c52a523488ebb0c5b29c208f085906c0a4ab5b3bf843a916a79ca17a2db01a9e</p>
--	--

Second review scope

Repository	https://github.com/Zharta/protocol-v1
Commit	9e860868a0f47217d57c8c643c999b0cf732672e
Functional Requirements	Link
Technical Requirements	Link
Contracts	<p>File: ./contracts/CollateralVaultCore.vy SHA3: baf1e75dd4a569fcd62e3f06cbb028ba1fcc1559b322c464be0783cdd63ebf65</p> <p>File: ./contracts/CollateralVaultPeripheral.vy SHA3: b2dcd41427819042b3c32693b7a0f40924604435810ceb41140e24a8a411a3bd</p> <p>File: ./contracts/CryptoPunksVaultCore.vy SHA3: 72d17dc6403f932c790f60601871553c79f56003e2160b9db027efaf1cea5dd8</p> <p>File: ./contracts/LegacyLendingPoolCore.vy SHA3: df9676c2a3acebd2754012a091258f2a8cb829c46bce07815b25b164b6e53403</p> <p>File: ./contracts/LendingPoolCore.vy SHA3: a154187f8d1da31a1a80d8d59ab9cdfef7b8056bc31826d52051e4c40ae67328</p> <p>File: ./contracts/LendingPoolLock.vy SHA3: df447e102f913bd8680d9152eacf53c1d0b9280836d9d07f3e07b9ac4c965ea1</p> <p>File: ./contracts/LendingPoolPeripheral.vy SHA3: 9d84e650d178767f8a62d78c31a536659246024339b50b8b037f55da3dce6137</p> <p>File: ./contracts/LiquidationsCore.vy SHA3: ecffaa0ec537dac5310708ba36ecc0da1806ed274ba0e8c6d111dbe4df113b07</p> <p>File: ./contracts/LiquidationsPeripheral.vy SHA3: 49b0ac97d986d10eae18c18dd49f8a255de84738e5631101279d4010a62e7024</p> <p>File: ./contracts/LiquidityControls.vy SHA3: 811c10a35b53b86897e9bb84853d9bfb7389f6cc4aef780c693c63cae7a2e32f</p> <p>File: ./contracts/Loans.vy SHA3: 2f78c28b5db338d7729962bb762d193b56c265440714e90505bfa3941997af1e</p> <p>File: ./contracts/LoansCore.vy SHA3: 16e086df18123bbeb89cbd60f0f540e5d2e82ff886e240ed138e7757dab82e69</p> <p>File: ./interfaces/ICollateralVaultCore.vy SHA3: fb0e1aa149e94b98792545aad78b4e21e7d06b1e0e643c3102affc265a3fe7f</p> <p>File: ./interfaces/ICollateralVaultPeripheral.vy SHA3: 53ea8d0a4725607c72a71d75f6355a72b1731f36a3c36cdf19034b0447fdddf8</p>

	<p>File: ./interfaces/ICryptoPunksVaultCore.vy SHA3: 78454a883fd7b6e9859ace84923efa20e7ce7b236caab3c20aa94a486263ec00</p> <p>File: ./interfaces/ILendingPoolCore.vy SHA3: 392ce0b36d19c59b56b9f1bfb5c21c8c4020f5fffb418d6e40a6260f6db1df5f</p> <p>File: ./interfaces/ILendingPoolLock.vy SHA3: d7d63d7388fb7820c973a5f40857914ee2b21f2a8c2a159e87e334f6d7d7085d</p> <p>File: ./interfaces/ILendingPoolPeripheral.vy SHA3: d2670359c2082822994e7d16deac7d88eb6b0ff7ce2939f043e2aa7ae9b13505</p> <p>File: ./interfaces/ILiquidationsCore.vy SHA3: 5c7ca5a21c3aec0a996dce77a9b22d7aed3ace1e8bf44f32df83c80fac2218b7</p> <p>File: ./interfaces/ILiquidationsPeripheral.vy SHA3: 116c937d5937355f8e86ba446745a69fb40d8e2dcab6802cde6e77ecf80571fd</p> <p>File: ./interfaces/ILiquidityControls.vy SHA3: 3d49a22427d2e3302ce380512491d3df2393409c3d5b1b0b16943cb0bdb05efc</p> <p>File: ./interfaces/ILoans.vy SHA3: d9b309f1b31f15fc568be9b6b58689f636cc6a2ccbd4bdd874cf704364b7e3b6</p> <p>File: ./interfaces/ILoansCore.vy SHA3: 446407cecd32e0e0374cd95bca21cb99d5657f044d14008a2259fe91941a8d46</p>
--	--

Third review scope

Repository	https://github.com/Zharta/protocol-v1
Commit	eff35233ad80dcecf45954d86e31eea95656c2ad
Functional Requirements	Link
Technical Requirements	Link
Contracts	<p>File: ./contracts/CollateralVaultCore.vy SHA3: baf1e75dd4a569fcd62e3f06cbb028ba1fcc1559b322c464be0783cdd63ebf65</p> <p>File: ./contracts/CollateralVaultPeripheral.vy SHA3: b2dcd41427819042b3c32693b7a0f40924604435810ceb41140e24a8a411a3bd</p> <p>File: ./contracts/CryptoPunksVaultCore.vy SHA3: 72d17dc6403f932c790f60601871553c79f56003e2160b9db027efaf1cea5dd8</p> <p>File: ./contracts/LegacyLendingPoolCore.vy SHA3: df9676c2a3acebd2754012a091258f2a8cb829c46bce07815b25b164b6e53403</p> <p>File: ./contracts/LendingPoolCore.vy SHA3: a72042ce26e05439cc98a65b32fe8f0180e2a93d06fd567ae5eb5f0d90f6ae48</p> <p>File: ./contracts/LendingPoolLock.vy SHA3: df447e102f913bd8680d9152eacf53c1d0b9280836d9d07f3e07b9ac4c965ea1</p> <p>File: ./contracts/LendingPoolPeripheral.vy SHA3: bab1ff3e718fe5ac21cdf6b7f6010e6ac978f4c75758799e62383f72eb2e64c2</p> <p>File: ./contracts/LiquidationsCore.vy SHA3: ecffaa0ec537dac5310708ba36ecc0da1806ed274ba0e8c6d111dbe4df113b07</p> <p>File: ./contracts/LiquidationsPeripheral.vy SHA3: 57316280ec62b843575b1f08ee9b3d79fe40757aa723e2a3c6f248218b7f468b</p>

	<p>File: ./contracts/LiquidityControls.vy SHA3: 811c10a35b53b86897e9bb84853d9bfb7389f6cc4aef780c693c63cae7a2e32f</p> <p>File: ./contracts/Loans.vy SHA3: 33850b1e5128b05a7c20097b89224d1eb98954b64291aced074532afa4e39e81</p> <p>File: ./contracts/LoansCore.vy SHA3: 16e086df18123bbeb89cbd60f0f540e5d2e82ff886e240ed138e7757dab82e69</p> <p>File: ./interfaces/ICollateralVaultCore.vy SHA3: fb0e1aa149e94b987925454aad78b4e21e7d06b1e0e643c3102affc265a3fe7f</p> <p>File: ./interfaces/ICollateralVaultPeripheral.vy SHA3: 53ea8d0a4725607c72a71d75f6355a72b1731f36a3c36cdf19034b0447fddff8</p> <p>File: ./interfaces/ICryptoPunksVaultCore.vy SHA3: 78454a883fd7b6e9859ace84923efa20e7ce7b236caab3c20aa94a486263ec00</p> <p>File: ./interfaces/ILendingPoolCore.vy SHA3: c4dc0e2ee58d94754a77b9a3d1d5fd292657f026af7bd799884f5553e4438380</p> <p>File: ./interfaces/ILendingPoolLock.vy SHA3: d7d63d7388fb7820c973a5f40857914ee2b21f2a8c2a159e87e334f6d7d7085d</p> <p>File: ./interfaces/ILendingPoolPeripheral.vy SHA3: 4f2896bcb1a22dcfb5a0a9368962a08fcd2171460fe4ff6fdbec72d766fb0cd</p> <p>File: ./interfaces/ILiquidationsCore.vy SHA3: 5c7ca5a21c3aec0a996dce77a9b22d7aed3ace1e8bf44f32df83c80fac2218b7</p> <p>File: ./interfaces/ILiquidationsPeripheral.vy SHA3: 6a5e1268fd2b39c287074d6483d41c1cb8d98e4534681bc075d84dff9df398f8</p> <p>File: ./interfaces/ILiquidityControls.vy SHA3: 3d49a22427d2e3302ce380512491d3df2393409c3d5b1b0b16943cb0bdb05efc</p> <p>File: ./interfaces/ILoans.vy SHA3: ad53244bb87eb1cd93d7c2d5deb103c8003e14aea35ad997a24d16ed850e7bed</p> <p>File: ./interfaces/ILoansCore.vy SHA3: 446407cecd32e0e0374cd95bca21cb99d5657f044d14008a2259fe91941a8d46</p>
--	---

Fourth review scope

Repository	https://github.com/Zharta/protocol-v1
Commit	5dcd9f978960f4d336c026041af44e8d95324549
Functional Requirements	Link
Technical Requirements	Link
Contracts	<p>File: ./contracts/CollateralVaultCore.vy SHA3: baf1e75dd4a569fcd62e3f06cbb028ba1fcc1559b322c464be0783cdd63ebf65</p> <p>File: ./contracts/CollateralVaultPeripheral.vy SHA3: b2dcd41427819042b3c32693b7a0f40924604435810ceb41140e24a8a411a3bd</p> <p>File: ./contracts/CryptoPunksVaultCore.vy SHA3: 72d17dc6403f932c790f60601871553c79f56003e2160b9db027efaf1cea5dd8</p> <p>File: ./contracts/LegacyLendingPoolCore.vy SHA3: df9676c2a3acebd2754012a091258f2a8cb829c46bce07815b25b164b6e53403</p>

	<p>File: ./contracts/LendingPoolCore.vy SHA3: a72042ce26e05439cc98a65b32fe8f0180e2a93d06fd567ae5eb5f0d90f6ae48</p> <p>File: ./contracts/LendingPoolLock.vy SHA3: df447e102f913bd8680d9152eac53c1d0b9280836d9d07f3e07b9ac4c965ea1</p> <p>File: ./contracts/LendingPoolPeripheral.vy SHA3: 56428d797cbda0867e68e386a99a623296397009ca1bc73b9ad42c85a1deee49</p> <p>File: ./contracts/LiquidationsCore.vy SHA3: ecffaa0ec537dac5310708ba36ecc0da1806ed274ba0e8c6d111dbe4df113b07</p> <p>File: ./contracts/LiquidationsPeripheral.vy SHA3: 7cfef46eb693fb10896e23095d2fe2a57cc881b9f26c9da62b7d4e1fe5c5cdcb</p> <p>File: ./contracts/LiquidityControls.vy SHA3: 811c10a35b53b86897e9bb84853d9bfb7389f6cc4aef780c693c63cae7a2e32f</p> <p>File: ./contracts/Loans.vy SHA3: 402c25df077277b5561523dfbbf785ebc341ca666bc20b6a505fb62bf1e54bf1</p> <p>File: ./contracts/LoansCore.vy SHA3: 16e086df18123bbeb89cbd60f0f540e5d2e82ff886e240ed138e7757dab82e69</p> <p>File: ./interfaces/ICollateralVaultCore.vy SHA3: fb0e1aa149e94b987925454aad78b4e21e7d06b1e0e643c3102affc265a3fe7f</p> <p>File: ./interfaces/ICollateralVaultPeripheral.vy SHA3: 53ea8d0a4725607c72a71d75f6355a72b1731f36a3c36cdf19034b0447fddff8</p> <p>File: ./interfaces/ICryptoPunksVaultCore.vy SHA3: 78454a883fd7b6e9859ace84923efa20e7ce7b236caab3c20aa94a486263ec00</p> <p>File: ./interfaces/ILendingPoolCore.vy SHA3: c4dc0e2ee58d94754a77b9a3d1d5fd292657f026af7bd799884f5553e4438380</p> <p>File: ./interfaces/ILendingPoolLock.vy SHA3: d7d63d7388fb7820c973a5f40857914ee2b21f2a8c2a159e87e334f6d7d7085d</p> <p>File: ./interfaces/ILendingPoolPeripheral.vy SHA3: adef9f2aa7d524fddac2c703c0f843bda62c305d0a882100afc5d609beda7a53</p> <p>File: ./interfaces/ILiquidationsCore.vy SHA3: 5c7ca5a21c3aec0a996dce77a9b22d7aed3ace1e8bf44f32df83c80fac2218b7</p> <p>File: ./interfaces/ILiquidationsPeripheral.vy SHA3: f7d7a7f66f6f2cc40d963942fbc3bfbdb2486e43b6313a9c61991fe96681c14d1</p> <p>File: ./interfaces/ILiquidityControls.vy SHA3: 3d49a22427d2e3302ce380512491d3df2393409c3d5b1b0b16943cb0bdb05efc</p> <p>File: ./interfaces/ILoans.vy SHA3: 703af4cc19c64a7a5f0e7b0dc9b489861a1940137fc59f2940ae98abb6c2e5cf</p> <p>File: ./interfaces/ILoansCore.vy SHA3: 446407cecd32e0e0374cd95bca21cb99d5657f044d14008a2259fe91941a8d46</p>
--	---

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations, but cannot lead to assets loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution, but affect the code quality

Executive Summary

The score measurement details can be found in the corresponding section of the [scoring methodology](#).

Documentation quality

The total Documentation Quality score is **7** out of **10**.

- The documentation gives a good overview of the system, and it is illustrated.
- Technical description is not provided. There is little information about actual formulas for fees, interests, earnings, collateral pricing. Ideally, the documentation should be detailed enough to be used as requirements for developers of this very system.
- Highly permissive owner's role is not described in the documentation.
- It is claimed that Zharta indexer calls LoansPeripheral validate/invalidate methods. The contract name is actually different, and there are no such methods.

Code quality

The total Code Quality score is **9** out of **10**.

- Code is well-written and formatted, architecture is well-designed.
- Presence of duplicate code.
- Code is documented with comments

Test coverage

Test coverage of the project is **70%**.

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is missed.
- Some tests are failing.
- Weak coverage for the LiquidationsPeripheral, LiquidityControls, and LoansCore contracts.

Security score

As a result of the audit, the code contains **2** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the "Findings" section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.4**.



The final score



Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
9 December 2022	4	5	16	2
6 January 2023	2	1	15	0
20 January 2023	2	1	1	0
23 January 2023	2	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Vyper compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Vyper Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	Passed
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Not Relevant
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	Custom	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Custom	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	Custom	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Not Relevant

Gas Limit and Loops	Custom	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	Custom	Style guides and best practices should be followed.	Passed
Requirements Compliance	Custom	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	Custom	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	Custom	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	Custom	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	Custom	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

The protocol part of the Zharta offering comprises the smart contracts that govern the logic for the lending pools (LPs), the loans, and the liquidations.

Lenders deposit funds which will be used for borrowing. Lenders can withdraw their funds after a lock period. The withdrawable amount is always \geq the total deposited amount, but sometimes funds may not be immediately available for withdrawal, because they are in use in loans. There is an upper bound on the ratio of the total borrowed amount to the total deposited amount. The interest from the paid loans is distributed across lenders as rewards (a fraction of the interest goes to the system). The lenders are claimed to have an additional privilege - there is a period after a loan default, when they are allowed to buy items from the collateral.

Borrowers may open a loan, which requires the system approval in the form of an off-chain signature on the loan request. The loan request includes a list of collaterals - NFT ids with a price that the system chose off-chain. When a loan is opened, the system transfers the desired funds to the borrower, and transfers the collateral NFTs in from the borrower. The loan has a maturity timestamp, up until which the loan can be paid for to receive the collateral back - only 100% payments are possible, no partial payments. The interest is accrued linearly across the interval up until the maturity.

If the maturity has been reached, the system may trigger the default via the respective contract call. After that call, there are several special adjacent periods (in that order): grace period, lender period, auction period. During the grace period, the borrower is still allowed to buy back all the collateral, similarly to the normal payment logic. The borrower is also allowed to buy a particular item back in this period. In the lender period, only those who deposited funds for borrowing are supposed to be able to buy an item from the collateral. During the auction period, anybody can trigger the function, which will try to sell a collateral item via NFTX (a DEX for NFTs); the minimal price for auto-selling is calculated by the system when the default is triggered. After the auction period, the contract owner can transfer an item from the collateral to any address.

Each domain of the system is represented by two smart contracts, the Core and the Periphery.

The Core contracts define the data structures and the necessary storage for the domain. They are less flexible contracts and can be viewed as the database of the domain. Within each domain, the Core contract methods can only be called by its correspondent Periphery contract.

The Periphery contracts define the protocol logic and are the gateway to interact with the domain. Each Periphery contract can interact with several other Periphery contracts. The system consists of the following contracts:

- *CollateralVaultCore* - a simple contract for storing and transferring ERC721 collaterals.
- *CollateralVaultPeripheral* - a simple contract that handles business logic for *CollateralVaultCore* contract.
- *LendingPoolPeripheral* - contract that allows to deposit and withdraw funds for users and invest these funds for loans contract.
- *LiquiditationCore* - the contract that stores liquidation data and provides an API for *LiquiditationPeripheral* to add/remove liquidations.
- *LiquiditationPeripheral* - business logic contract, that interacts with multiple contracts to add liquidations, pay loans, buy NFT grace/lender period and liquidate NFTX.
- *LiquidityControls* - helper contract that helps to validate if landing out of lock period, calculate pool, loan and collection share limits.
- *Loans* - business logic contract to create and pay loan.
- *LoansCore* - helper contracts to store loans and update their values.

Privileged roles

- The *CollateralVaultCore* contract has the next privileged roles:
 - owner can:
 - transfer ownership
 - set new collateral vault peripheral address
 - collateral vault peripheral can:
 - approve ERC721 token
 - transfer from sender to contract ERC721 token
 - transfer our from the contract ERC721 token
- The *CollateralVaultPeripheral* contract has the next privileged roles:
 - owner can:
 - transfer ownership
 - add/remove loan peripheral address
 - set liquidation peripheral address
 - loan peripheral can:
 - store collateral
 - transfer collateral from loan
 - liquidation peripheral can:
 - transfer collateral from liquidation
 - approve backstop buyer
- The *LendingPoolPeripheral* contract has the next privileged roles:
 - owner can:
 - transfer ownership

- change max capital efficiency
 - change protocol wallet
 - change protocol fee share
 - set loan peripheral address
 - set liquidation peripheral address
 - set liquidation control address
 - change whitelist status
 - add/remove addresses from whitelist
 - change pool active status
 - deprecate pool
- loans contract can:
 - invest funds from the contract
 - send back funds after loans contract investment
- liquidation peripheral can:
 - sends funds from liquidation to the contract
- The *LiquidationsCore* contract has the next privileged roles:
 - owner can:
 - transfer ownership
 - set liquidation peripheral address
 - add/remove loan core address
 - liquidation peripheral can:
 - add/remove liquidation
- The *LiquidationsPeripheral* contract has the next privileged roles:
 - owner can:
 - transfer ownership
 - set grace period duration
 - set lenders period duration
 - set auction period duration
 - set liquidation core address
 - add/remove loans core addresses
 - add/remove lending peripheral addresses
 - set collateral peripheral address
 - set NFTX vault factory address
 - set NFTX marketplace zap address
 - set sushi router address
 - transfer collateral after auction period
- The *LiquidityControls* contract has the next privileged roles:
 - owner can:
 - set max pool share conditions
 - set max loans share conditions
 - set max collections share conditions
 - change lock period duration
- The *Loans* contract has the next privileged roles:
 - owner can:

- transfer ownership
- set max allowed loans
- set max allowed loans duration
- set max loan amount
- change interest actual period
- add/remove collateral from whitelist
- set lending pool peripheral address
- set collateral vault peripheral address
- set liquidation peripheral address
- set liquidity controls address
- change wallet whitelist status
- add/remove wallet from whitelist
- change contract accepting loans status
- deprecate contract
- The *LoansCore* contract has the next privileged roles:
 - owner can:
 - transfer ownership
 - set loans peripheral address
 - loans peripheral can:
 - add/remove collateral from loan
 - update collaterals
 - add loan
 - update loan started
 - update loan invalidated
 - update loan paid amount
 - update paid loan
 - update default loan
 - update canceled loan
 - update highest single collateral loan
 - update highest bundle collateral loan
 - update highest repayment
 - update highest defaulted loan

Risks

- Highly permissive role access allows the owner to update implementations in contracts, see mitigated high issues for reference.
- *NFTX* integration is out of the audit scope, but the existing system interacts with them.
- The system relies on some contract methods being called at right timings (e.g. by off-chain software) - that may not happen for various reasons, and lead to unpleasant consequences. For example, if a loan maturity has passed, the loan is stuck until the default is

initiated, which defers the ability of the borrower to buy the collateral back during the grace period.

- *LiquidityControls* does not have a key-rotation scheme like the other contracts. If the single private key for the “owner” address is lost, there is no way to regain control.

Findings

■■■■ Critical

1. Access Control Violation

`LiquidationsPeripheral.addLiquidation` can be called by anybody.
Exploit example:

- `addr1` opens a loan with id `N`, giving `tok1`, `tok2` as the collateral (assume some predefined NFT contract, to which the tokens belong).
- The loan gets past maturity, transitions into the defaulted state via `Loans.settleDefault` (the liquidations are created, etc.)
- The liquidations get closed in some way e.g. `addr1` buys the collateral back during the grace period.
- After some time, imagine that `tok1` and `tok2` made it into the ownership of some other party `addr2`. Imagine that `addr2` opens a loan giving the same tokens as collateral again.
- Now, `tok1` and `tok2` are not under liquidation, and are under the system ownership. Anybody can open liquidations on the tokens again, by calling `LiquidationsPeripheral.addLiquidation` with `borrower=addr1`, `loanId=N`. This is while they are supposed to be locked until the maturity of the loan `addr2` opened.
- The normal liquidation logic now works for `tok1`, `tok2`, e.g. `addr1` can buy them immediately at the old price, and so on.

Path: `./contracts/LiquidationsPeripheral.vy : addLiquidation()`

Recommendation: Do not allow calling the function by anybody. Do not allow creating liquidations from loans that are in a terminal state.

Status: Fixed (9e860868a0f47217d57c8c643c999b0cf732672e)

2. Requirements Violation

`LiquidationsPeripheral.buyNFTLenderPeriod` can be called by anybody, not necessarily a lender, like it is declared in the requirements: *The liquidation periods are periods where only certain actors can take part in the liquidation of the collateral:*

- *Grace period: only for the borrower*
- *Lenders period: only for the lenders*

Path: `./contracts/LiquidationsPeripheral.vy : buyNFTLenderPeriod()`

Recommendation: Synchronize the code with the requirements.

Status: Fixed (9e860868a0f47217d57c8c643c999b0cf732672e)

■■■ High

1. Denial of Service Vulnerability

LiquidationsPeripheral contracts interact with Solidity-based contracts to execute the *vaultsForAsset* function, which has not limited array length, but existing contracts expect *DynArray[address, 20]*.

Execution will be reverted in this case.

Path: `./contracts/LiquidationsPeripheral.vy:
_getNFTXVaultAddrFromCollateralAddr()`

Recommendation: Prevent DoS vulnerability in this case.

Status: **Fixed** (eff35233ad80dcecf45954d86e31eea95656c2ad)

2. Highly Permissive Role Access

The owner could unexpectedly use *CollateralVaultCore.setCollateralVaultPeripheralAddress* to set an implementation that could break the system or impact users' assets.

Example: the new implementation could call *CollateralVaultCore.transferCollateral* to send any locked collateral to any address.

Path: `./contracts/CollateralVaultCore.vy :
setCollateralVaultPeripheralAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once.

Status: **Mitigated** (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

3. Highly Permissive Role Access

The owner could unexpectedly use *CollateralVaultPeripheral.addLoansPeripheralAddress* to set an implementation that could break the system or impact users' assets.

Example: the new implementation could call *CollateralVaultPeripheral.transferCollateralFromLoan* to send any locked collateral to any address.

Path: `./contracts/CollateralVaultPeripheral.vy :
addLoansPeripheralAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once per ERC20 token.

Status: **Mitigated** (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

4. Highly Permissive Role Access

The owner could unexpectedly use `CollateralVaultPeripheral.setLiquidationsPeripheralAddress` to set an implementation that could break the system or impact users' assets.

Example: the new implementation could call `CollateralVaultPeripheral.transferCollateralFromLiquidation` to send any locked collateral to any address.

Path: `./contracts/CollateralVaultPeripheral.vy` :
`setLiquidationsPeripheralAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once.

Status: **Mitigated** (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

5. Highly Permissive Role Access

The owner could unexpectedly use `LendingPoolCore.setLendingPoolPeripheralAddress` to set an implementation that could break the system or impact users' assets.

Example: the new implementation could call `LendingPoolCore.withdraw` to send locked funds to any address.

Path: `./contracts/LendingPoolCore.vy` :
`setLendingPoolPeripheralAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once.

Status: **Mitigated** (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

6. Highly Permissive Role Access

The owner could unexpectedly use `LendingPoolPeripheral.setLoansPeripheralAddress` to set an implementation that could break the system or impact users' assets.

Example: the new implementation could call `LendingPoolPeripheral.sendFunds` to send locked funds to any address.

Path: `./contracts/LendingPoolPeripheral.vy` :
`setLoansPeripheralAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once.

Status: **Mitigated** (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

7. Highly Permissive Role Access

The owner could unexpectedly use *LendingPoolPeripheral.setLiquidityControlsAddress* to set an implementation that could break the system or impact users' assets.

Example: the new implementation could block *LendingPoolPeripheral.withdraw* for any set of users.

Path: `./contracts/LendingPoolPeripheral.vy` :
`setLiquidityControlsAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once.

Status: *Mitigated* (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

8. Highly Permissive Role Access

The owner could unexpectedly use *LiquidationsCore.setLiquidationsPeripheralAddress* to set an implementation that could break the system or impact users' assets.

Example: the new implementation could call *LiquidationsCore.addLiquidation* to create liquidations on loans that have not defaulted, and set arbitrary liquidation parameters (e.g. disable grace/lender/auction periods).

Path: `./contracts/LiquidationsCore.vy` :
`setLiquidationsPeripheralAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once.

Status: *Mitigated* (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

9. Highly Permissive Role Access

The owner could unexpectedly use *LiquidationsPeripheral.setLiquidationsCoreAddress* to set an implementation that could break the system or impact users' assets.

Example: The new implementation could return liquidation objects which include NFTs that are different from the original (e.g. some worthless pre-setup assets). A borrower could do a buyback during the grace period and receive not what it expected.

Path: `./contracts/LiquidationsPeripheral.vy` :
`setLiquidationsCoreAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once.

Status: **Mitigated** (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

10. Highly Permissive Role Access

The owner could unexpectedly use `LiquidationsPeripheral.addLoansCoreAddress` to set an implementation that could break the system or impact users' assets.

Example: The new implementation could return loan objects, including collateral objects, which include NFTs that are different from the original (e.g. some worthless pre-setup assets). A borrower could do a buyback during the grace period and receive not what it expected.

Path: `./contracts/LiquidationsPeripheral.vy : addLoansCoreAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once per ERC20 token.

Status: **Mitigated** (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

11. Highly Permissive Role Access

The owner could unexpectedly use `LiquidationsPeripheral.setCollateralVaultPeripheralAddress` to set an implementation that could break the system or impact users' assets.

Example: The new implementation could implement `transferCollateralFromLiquidation` in a way that does not transfer the collaterals at all (e.g. in exchange for a borrower buyback).

Path: `./contracts/LiquidationsPeripheral.vy : setCollateralVaultPeripheralAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once per ERC20 token.

Status: **Mitigated** (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

12. Highly Permissive Role Access

The owner could unexpectedly use `LiquidityControls.changeLockPeriodConditions` to block lender withdrawals.

Path: `./contracts/LiquidityControls.vy : changeLockPeriodConditions()`

Recommendation: Document this possibility explicitly or set reasonable limits for the parameter.

Status: **Fixed** (9e860868a0f47217d57c8c643c999b0cf732672e)

13. Highly Permissive Role Access

The owner could unexpectedly use `Loans.changeInterestAccrualPeriod` to instantly increase the amounts owed by borrowers without bound.

Path: `./contracts/Loans.vy : changeInterestAccrualPeriod()`

Recommendation: Document this possibility explicitly. Set reasonable limits for the parameter. Loan conditions should not change on the fly - this parameter could be copied into `Loan` object so that the calculations always use the original value.

Status: `Mitigated` (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

14. Highly Permissive Role Access

The owner could unexpectedly use `Loans.setLendingPoolPeripheralAddress` to set an implementation that could break the system or impact users' assets.

Example: The new implementation could implement `sendFunds` in a way that does not transfer funds to a borrower.

Path: `./contracts/Loans.vy : setLendingPoolPeripheralAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once.

Status: `Mitigated` (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

15. Highly Permissive Role Access

The owner could unexpectedly use `Loans.setCollateralVaultPeripheralAddress` to set an implementation that could break the system or impact users' assets.

Example: The new implementation could implement `transferCollateralFromLoan` in a way that does not transfer collateral to a borrower when a loan is paid.

Path: `./contracts/Loans.vy : setCollateralVaultPeripheralAddress()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once.

Status: `Mitigated` (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

16. Highly Permissive Role Access

The owner could unexpectedly use `LoansCore.setLoansPeripheral` to set an implementation that could break the system or impact users' assets.

Example: The new implementation could call `LoansCore.updateDefaultedLoan` to force default/liquidations on any loan.

Path: `./contracts/LoansCore.vy : setLoansPeripheral()`

Recommendation: Document this possibility explicitly or allow the setter to be called only once.

Status: **Mitigated** (Client's notice: Added to public documentation as a known issue and would be fixed in the further iterations to make system immutable)

17. Funds Lock

`LendingPoolPeripheral` contract can receive native tokens with default payable function, but there is no mechanism for their withdrawals, which leads to native tokens lock in the contract.

Path: `./contracts/LendingPoolPeripheral.vy : __default__()`

Recommendation: Remove the default payable function or add the possibility to withdraw native tokens.

Status: **Fixed** (5dcd9f978960f4d336c026041af44e8d95324549 : The fix is only applicable for Ethereum Mainnet)

■ ■ Medium

1. Undocumented Behavior; Requirements Violation

From provided documentation: *lock period represents the lock period applicable for deposits in lending pools, i.e. for each new deposit, it can't be withdrawn before the lock period finishes. If the lender already has an ongoing lock period, a new deposit won't extend the lock period.*

Existing logic will lock previous deposits in case of adding a new deposit when there are no ongoing lock periods, but according to the provided documentation, it is not obvious.

Path: `./contracts/LendingPoolCore.vy : deposit()`

Recommendation: Check if existing logic is correct, update the documentation to highlight such behavior.

Status: **Fixed** (9e860868a0f47217d57c8c643c999b0cf732672e)

2. Invalid Event Tracking

Incorrect events are tracked, which breaks state observability.

`MaxLoansPoolShareFlagChanged` event is tracked, when `maxCollectionShareEnabled` is updated, instead of `MaxCollectionShareFlagChanged`.

`MaxLoansPoolShareChanged` event is tracked, when `maxCollectionShare` is updated, instead of `MaxCollectionShareChanged`.

Path: `./contracts/CollateralVaultPeripheral.vy: changeMaxCollectionShareConditions()`

Recommendation: Track correct events.

Status: **Fixed** (9e860868a0f47217d57c8c643c999b0cf732672e)

3. Missing Access Control

`LiquidationsPeripheral.liquidateNFTX` can be called by anybody. No severe consequences were found, yet it is unlikely that the dApp architecture needs anybody to call the function: it seems that this function is meant to be called by the dApp code. It looks like the authors just forgot to add the access control.

Example of an unwanted consequence: somebody could trigger the liquidation at a less favorable state of the market than the interested parties would choose.

Path: `./contracts/LiquidationsPeripheral.vy : liquidateNFTX()`

Recommendation: Restrict the callers to some system addresses.

Status: **Fixed** (9e860868a0f47217d57c8c643c999b0cf732672e)

4. Excessive Permissions Required

The check in `Loans._areCollateralsApproved` requires that a user approves `Loans` to do anything with any token that belongs to an NFT contract that belongs to a loan the user wants to open.

Path: `./contracts/Loans.vy : _areCollateralsApproved`

Recommendation: Require approvals at token granularity. The contract should renounce itself as the approved address for a token (by setting the approved address to zero) once it releases the token.

Status: **Fixed** (9e860868a0f47217d57c8c643c999b0cf732672e)

5. Signed Message Replay Attack

Signature verification is often used for access checks. It should not be possible to use the same signed message twice.

The current implementation of signature allows reusing it multiple times before `_deadline` is reached.

Also, the borrower's address is not part of the signature and is recommended to be added.

Path: `./contracts/Loans.vy : reserve()`

Recommendation: Add nonce to the signature.

Status: **Fixed** (5dcd9f978960f4d336c026041af44e8d95324549)

www.hacken.io

■ Low

1. Redundant Operation

Existing implementation of the `onERC721Received` function converts output twice, which is redundant and could be optimized by the specifying expected `output_type` of the `method_id` function.

Path: `./contracts/CollateralVaultCore.vy: onERC721Received()`

Recommendation: Remove `convert` function and use `output_type=bytes4` instead.

Status: Fixed (9e860868a0f47217d57c8c643c999b0cf732672e)

2. Floating Compiler Version

Locking the compiler version helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Path: All files

Recommendation: Remove `^` from compiler version.

Status: Fixed (9e860868a0f47217d57c8c643c999b0cf732672e)

3. Duplicate Code

- struct `TopStats:` `contracts/LoansCore.vy`
`interfaces/ILoansCore.vy`
- struct `Liquidation:` `contracts/LiquidationsCore.vy`
`contracts/LiquidationsPeripheral.vy`
`interfaces/ILiquidationsCore.vy`
`interfaces/ILiquidationsPeripheral.vy`
- struct `Collateral:` `contracts/LiquidationsPeripheral.vy`
`contracts/LoansCore.vy` `contracts/Loans.vy`
`interfaces/ILoansCore.vy` `interfaces/ILoans.vy`
`interfaces/ILiquidationsPeripheral.vy`
- struct `Loan:` `contracts/LiquidationsPeripheral.vy`
`contracts/LoansCore.vy` `contracts/Loans.vy`
`interfaces/ILoansCore.vy` `interfaces/ILoans.vy`
`interfaces/ILiquidationsPeripheral.vy`
- struct `InvestorFunds:` `contracts/LiquidationsPeripheral.vy`
`contracts/LendingPoolCore.vy` `contracts/LendingPoolPeripheral.vy`
`interfaces/ILendingPoolCore.vy`
`interfaces/ILendingPoolPeripheral.vy`
`interfaces/ILiquidationsPeripheral.vy`
- event `GracePeriodDurationChanged:`
`contracts/LiquidationsPeripheral.vy`
`interfaces/ILiquidationsPeripheral.vy`
- event `LendersPeriodDurationChanged:`
`contracts/LiquidationsPeripheral.vy`
`interfaces/ILiquidationsPeripheral.vy`

- event *AuctionPeriodDurationChanged:*
contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *LiquidationsCoreAddressSet:*
contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *LendingPoolPeripheralAddressAdded:*
contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *LendingPoolPeripheralAddressRemoved:*
contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *NFTXVaultFactoryAddressSet:*
contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *NFTXMarketplaceZapAddressSet:*
contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *SushiRouterAddressSet:*
contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *LiquidationAdded:* contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *LiquidationRemoved:* contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *NFTPurchased:* contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *AdminWithdrawal:* contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsPeripheral.vy
- event *CollateralVaultCoreAddressSet:*
contracts/CollateralVaultPeripheral.vy
interfaces/ICollateralVaultPeripheral.vy
- event *LoansPeripheralAddressAdded:*
contracts/CollateralVaultPeripheral.vy
interfaces/ICollateralVaultPeripheral.vy
- event *LoansPeripheralAddressRemoved:*
contracts/CollateralVaultPeripheral.vy
interfaces/ICollateralVaultPeripheral.vy
- event *CollateralStored:* contracts/CollateralVaultPeripheral.vy
interfaces/ICollateralVaultPeripheral.vy
- event *CollateralFromLoanTransferred:*
contracts/CollateralVaultPeripheral.vy
interfaces/ICollateralVaultPeripheral.vy
- event *CollateralFromLiquidationTransferred:*
contracts/CollateralVaultPeripheral.vy
interfaces/ICollateralVaultPeripheral.vy
- event *OperatorApproved:* contracts/CollateralVaultPeripheral.vy
interfaces/ICollateralVaultPeripheral.vy
- event *MaxPoolShareFlagChanged:* contracts/LiquidityControls.vy
interfaces/ILiquidityControls.vy
- event *MaxPoolShareChanged:* contracts/LiquidityControls.vy
interfaces/ILiquidityControls.vy

- event *MaxLoansPoolShareFlagChanged:*
 contracts/LiquidityControls.vy
 interfaces/ILiquidityControls.vy
- event *MaxLoansPoolShareChanged:* contracts/LiquidityControls.vy
 interfaces/ILiquidityControls.vy
- event *MaxCollectionShareFlagChanged:*
 contracts/LiquidityControls.vy interfaces/ILiquidityControls.vy
- event *MaxCollectionShareChanged:* contracts/LiquidityControls.vy
 interfaces/ILiquidityControls.vy
- event *LockPeriodFlagChanged:* contracts/LiquidityControls.vy
 interfaces/ILiquidityControls.vy
- event *LockPeriodDurationChanged:* contracts/LiquidityControls.vy
 interfaces/ILiquidityControls.vy
- event *MaxCapitalEfficiencyChanged:*
 contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *ProtocolWalletChanged:* contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *ProtocolFeesShareChanged:*
 contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *WhitelistStatusChanged:*
 contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *WhitelistAddressAdded:* contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *WhitelistAddressRemoved:*
 contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *InvestingStatusChanged:*
 contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *Deposit:* contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *Withdrawal:* contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *FundsTransfer:* contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *FundsReceipt:* contracts/LendingPoolPeripheral.vy
 interfaces/ILendingPoolPeripheral.vy
- event *MaxAllowedLoansChanged:* contracts/Loans.vy
 interfaces/ILoans.vy
- event *MaxLoansChanged:* contracts/Loans.vy interfaces/ILoans.vy
- event *MaxLoanDurationChanged:* contracts/Loans.vy
 interfaces/ILoans.vy
- event *MaxLoanAmountChanged:* contracts/Loans.vy
 interfaces/ILoans.vy
- event *InterestAccrualPeriodChanged:* contracts/Loans.vy
 interfaces/ILoans.vy
- event *CollateralToWhitelistAdded:* contracts/Loans.vy
 interfaces/ILoans.vy

- event *CollateralToWhitelistRemoved:*
contracts/Loans.vy interfaces/ILoans.vy
- event *WalletsWhitelistStatusChanged:* contracts/Loans.vy
interfaces/ILoans.vy
- event *WhitelistedWalletAdded:* contracts/Loans.vy
interfaces/ILoans.vy
- event *WhitelistedWalletRemoved:* contracts/Loans.vy
interfaces/ILoans.vy
- event *LoanCreated:* contracts/Loans.vy interfaces/ILoans.vy
- event *LoanPayment:* contracts/Loans.vy interfaces/ILoans.vy
- event *LoanPaid:* contracts/Loans.vy interfaces/ILoans.vy
- event *LoanDefaulted:* contracts/Loans.vy interfaces/ILoans.vy
- event *LoansCoreAddressAdded:* contracts/LiquidationsCore.vy
contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsCore.vy
interfaces/ILiquidationsPeripheral.vy
- event *LoansCoreAddressRemoved:* contracts/LiquidationsCore.vy
contracts/LiquidationsPeripheral.vy
interfaces/ILiquidationsCore.vy
interfaces/ILiquidationsPeripheral.vy
- event *LendingPoolPeripheralAddressSet:*
contracts/LendingPoolCore.vy contracts/Loans.vy
interfaces/ILendingPoolCore.vy interfaces/ILoans.vy
- event *LoansPeripheralAddressSet:*
contracts/LendingPoolPeripheral.vy contracts/LoansCore.vy
interfaces/ILendingPoolPeripheral.vy interfaces/ILoansCore.vy
- event *LiquidityControlsAddressSet:*
contracts/LendingPoolPeripheral.vy contracts/Loans.vy
interfaces/ILendingPoolPeripheral.vy interfaces/ILoans.vy
- event *ContractStatusChanged:* contracts/LendingPoolPeripheral.vy
contracts/Loans.vy interfaces/ILendingPoolPeripheral.vy
interfaces/ILoans.vy
- event *ContractDeprecated:* contracts/LendingPoolPeripheral.vy
contracts/Loans.vy interfaces/ILendingPoolPeripheral.vy
interfaces/ILoans.vy
- event *CollateralVaultPeripheralAddressSet:*
contracts/LiquidationsPeripheral.vy
contracts/CollateralVaultCore.vy contracts/Loans.vy
interfaces/ICollateralVaultCore.vy interfaces/ILoans.vy
interfaces/ILiquidationsPeripheral.vy
- event *LiquidationsPeripheralAddressSet:*
contracts/LiquidationsCore.vy
contracts/CollateralVaultPeripheral.vy
contracts/LendingPoolPeripheral.vy contracts/Loans.vy
interfaces/ICollateralVaultPeripheral.vy
interfaces/ILiquidationsCore.vy
interfaces/ILendingPoolPeripheral.vy interfaces/ILoans.vy
- event *OwnershipTransferred:* contracts/LiquidationsCore.vy
contracts/LiquidationsPeripheral.vy
contracts/CollateralVaultCore.vy
contracts/CollateralVaultPeripheral.vy
contracts/LendingPoolCore.vy contracts/LendingPoolPeripheral.vy
contracts/LoansCore.vy contracts/Loans.vy

- interfaces/ILendingPoolCore.vy
- interfaces/ICollateralVaultPeripheral.vy
- interfaces/ILiquidationsCore.vy
- interfaces/ILendingPoolPeripheral.vy
- interfaces/ICollateralVaultCore.vy interfaces/ILoansCore.vy
- interfaces/ILoans.vy interfaces/ILiquidationsPeripheral.vy
- event *OwnerProposed*: contracts/LiquidationsCore.vy
- contracts/LiquidationsPeripheral.vy
- contracts/CollateralVaultCore.vy
- contracts/CollateralVaultPeripheral.vy
- contracts/LendingPoolCore.vy contracts/LendingPoolPeripheral.vy
- contracts/LoansCore.vy contracts/Loans.vy
- interfaces/ILendingPoolCore.vy
- interfaces/ICollateralVaultPeripheral.vy
- interfaces/ILiquidationsCore.vy
- interfaces/ILendingPoolPeripheral.vy
- interfaces/ICollateralVaultCore.vy interfaces/ILoansCore.vy
- interfaces/ILoans.vy interfaces/ILiquidationsPeripheral.vy
- *ownerIndexed* and *proposedOwnerIndexed* are duplicated with *owner* and *proposedOwner* duplication inside *OwnershipTransferred* and *OwnerProposed* events.

Recommendation: Do not duplicate code.

Status: **Reported**

4. Unused Code

- contracts/LoansCore.vy: *updateInvalidLoan(..)* is never called from the periphery.
- contracts/LoansCore.vy: *updateCanceledLoan(..)* is never called from the periphery.
- contracts/LiquidationsCore.vy: *loansCoreAddresses* - field is only written.
- ./contracts/CollateralVaultPeripheral.vy: *CollateralVaultCoreAddressSet* - unused event.
- ./contracts/Loans.vy: *MaxAllowedLoansChanged* - unused event.
- ./contracts/LendingPoolCore.vy: *totalFundsInvested* variable is duplicated by *fundsInvested* and could be removed.
- contracts/Loans.vy : *_withinCollectionShareLimit()* uses *i* variable which could be removed. There is no need to init *self.collectionsAmount[collateral.contractAddress]* for the first time and *+=* operation could be used for the initialization as well.

Recommendation: Remove the unused code or make use of it.

Status: **Fixed** (9e860868a0f47217d57c8c643c999b0cf732672e)

5. Commented Code Parts

Path: ./contracts/Loans.vy:525

Recommendation: Commented-out code may confuse a reader; it is better to remove it.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Status: Reported

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, Consultant cannot guarantee the explicit security of the audited smart contracts.