Smart Contract Audit Report

K33Loans

Audit Performed By

Fortknox Security

Professional Smart Contract Auditing

January 22, 2025

Table of Contents

Executive Summary

Fortknox Security has conducted a comprehensive smart contract security audit for **K33Loans**. Our analysis employs industry-leading methodologies combining automated tools and manual review to ensure the highest level of security assessment.

7	0	LOW	100%
TOTAL ISSUES FOUND	CRITICAL + HIGH	OVERALL RISK	CODE COVERAGE

Security Assessment Overview

Critical Issues

0

Immediate action required. These vulnerabilities can lead to direct loss of funds.

IMPACT: SEVERE FINANCIAL LOSS

High Issues

0

High priority fixes needed. Can lead to significant financial loss.

IMPACT: MAJOR SECURITY RISK

Key Findings Summary

Access Control

Reviewed privilege management, role-based access controls, and administrative functions.

Economic Security

Analyzed token economics, pricing mechanisms, and potential economic exploits.

Logic Validation

Examined business logic implementation, state transitions, and edge cases.

Input Validation

Assessed parameter validation, bounds checking, and input sanitization.

Audit Conclusion

The K33Loans smart contract audit reveals **7 total findings** across various security categories. No critical or high severity issues were identified. Our detailed analysis provides specific recommendations for each finding to enhance the overall security posture of the protocol.

Audit Methodology

Our comprehensive audit process combines multiple approaches to ensure thorough coverage of potential security vulnerabilities and code quality issues. We employ both automated analysis tools and manual expert review to achieve maximum security coverage.

Tools & Techniques

Static Analysis

Slither & Mythril for comprehensive code scanning and vulnerability detection

Manual Review

Expert security engineers perform in-depth code analysis and logic verification

Business Logic

Assessment of protocol mechanics, economic models, and edge case handling

Gas Analysis

Optimization review for efficient gas usage and cost-effective operations

Formal Verification

Mathematical proof methods to verify critical contract properties

Symbolic Execution

Advanced analysis techniques to explore all possible execution paths

Review Process & Standards

Review Process

1 Initial Scanning

Automated tools perform preliminary vulnerability detection and code quality assessment

2 Manual Review

Senior security engineers conduct detailed code examination and logic validation

3 Business Logic Testing

Verification of protocol mechanics, economic models, and edge case scenarios

4 Architecture Analysis

Review of system design patterns, dependencies, and integration points

5 Final Documentation

Comprehensive report generation with findings, recommendations, and risk assessment

Severity Classification

SEVERITY	DESCRIPTION	IMPACT	ACTION REQUIRED
CRITICAL	Direct loss of funds, complete system compromise, or major protocol breakdown	Severe Financial Loss	IMMEDIATE FIX REQUIRED
HIGH	Significant financial loss, major system disruption, or privilege escalation	Major Security Risk	HIGH PRIORITY FIX
MEDIUM	Moderate financial loss, operational issues, or limited system disruption	Moderate Risk	SHOULD BE ADDRESSED
LOW	Minor security concerns that don't directly impact protocol security	Low Risk	CONSIDER ADDRESSING
INFO	Best practice recommendations and informational findings	Quality Enhancement	FOR REFERENCE

Audit Scope

Project Details

PARAMETER	DETAILS
Project Name	K33Loans
Total Issues Found	7
Audit Type	Smart Contract Security Audit
Methodology	Manual Review + Automated Analysis

Files in Scope

This audit covers the smart contract codebase and associated components for K33Loans.

Audit Timeline

- ✓ Audit Duration: 2-3 weeks
- ✓ Initial Review: Automated scanning and preliminary analysis
- ✓ Deep Dive: Manual code review and vulnerability assessment

Vulnerability Analysis

Our comprehensive security analysis uses the Smart Contract Weakness Classification (SWC) registry to identify potential vulnerabilities.

SWC Security Checks

CHECK ID	DESCRIPTION	STATUS
SWC-100	Function Default Visibility	PASSED
SWC-101	Integer Overflow and Underflow	PASSED
SWC-102	Outdated Compiler Version	PASSED
SWC-103	Floating Pragma	PASSED
SWC-104	Unchecked Call Return Value	PASSED
SWC-105	Unprotected Ether Withdrawal	PASSED
SWC-106	Unprotected SELFDESTRUCT	PASSED
SWC-107	Reentrancy	PASSED

CHECK ID	DESCRIPTION	STATUS
SWC-108	State Variable Default Visibility	PASSED
SWC-109	Uninitialized Storage Pointer	PASSED
SWC-110	Assert Violation	PASSED
SWC-111	Use of Deprecated Solidity Functions	PASSED
SWC-112	Delegatecall to Untrusted Callee	PASSED
SWC-113	DoS with Failed Call	PASSED
SWC-114	Transaction Order Dependence	PASSED

Contract Privileges Analysis

Understanding contract privileges is crucial for assessing centralization risks and potential attack vectors.

Common Privilege Categories

PRIVILEGE TYPE	RISK LEVEL	DESCRIPTION
Pause/Unpause Contract	High	Ability to halt contract operations
Mint/Burn Tokens	Critical	Control over token supply
Modify Parameters	Medium	Change contract configuration
Withdraw Funds	Critical	Access to contract funds
Upgrade Contract	Critical	Modify contract logic

Mitigation Strategies

- ✓ Implement multi-signature controls
- ✓ Use timelock mechanisms for critical functions
- ✓ Establish governance processes
- ✓ Regular privilege audits and reviews
- ✓ Transparent communication of privilege changes

M-0 | User Can Pass Arbitrary Fee Amount

CATEGORY	SEVERITY	LOCATION	STATUS
Logical Error	MEDIUM	Vanir.sol	Acknowledged

Description

Function `borrow` accepts an arbitrary `feeAmount` through a `loan` call. If K33 expects a fee to be paid on borrow, a user can just directly interact with the contract and pass a `feeAmount` of zero to preserve as much capital.

```
borrow
feeAmount
loan
feeAmount
```

Recommendation

Add validation on the expected `feeAmount` for K33. Alternatively, consider tieing feeAmount to a fixed percentage of the borrowed/liquidated amount rather than letting the caller supply an arbitrary fee, implement a fixed percentage model (e.g., `feeAmount = (borrowedAmount * feePercentage) / 1e18)`.

```
feeAmount
feeAmount = (borrowedAmount * feePercentage) / 1e18)
```

Resolution

K33 Team: We own all the wallets our customers utilize, so we are not concerned with external parties calling the contract using their own wallets

M-1 | Failed Approvals With USDT

CATEGORY	SEVERITY	LOCATION	STATUS
Logical Error	MEDIUM	Vanir.sol	Resolved

Description

Function `repay` approves `amount` and then calls `SparkLend` to repay the amount. The issue is that the `paybackAmount` within the BorrowLogic is not necessarily equal to `amount` passed, so a portion of the approved amount will be not be utilized.

```
repay
```

Recommendation

For all allowances in K33, use `SafeERC20's forceApprove` which will force the allowance to go to zero initially to handle tokens such as USDT. Also, use it after an external call to set the approval to zero after the approval is no longer necessary.

```
SafeERC20's forceApprove
```

Resolution

K33 Team: Added a `forceApproval` function to the contract that performs the functionality as needed. We also reset the allowance where needed.

```
forceApproval
```

M-2 | Lost Borrow Fee

CATEGORY	SEVERITY	LOCATION	STATUS
Validation	MEDIUM	Vanir.sol	Acknowledged

Description

When users borrow via K33, a part of the borrowed assets is paid as a fee to the `feeAddress`, which is an arbitrary passed parameter, but is validated to be a whitelisted address.

```
feeAddress
```

Recommendation

Create a separate validation for the `feeAddress`.

```
feeAddress
```

Resolution

K33 Team: If an external party wishes to utilize the contract without disrupting its function, they are free to do so.

L-0 | No Controls On Liquidation Fee Amount

CATEGORY	SEVERITY	LOCATION	STATUS
Validation	LOW	Vanir.sol	Resolved

Description

During liquidation, the user pays an additional `feeAmount` from their collateral. This `feeAmount` is arbitrarily set by the admin within their `LiquidationRequest`, and can vary each time.

```
feeAmount
feeAmount
LiquidationRequest
```

Recommendation

Consider adding on-chain validations for `feeAmount` on liquidations, or clearly document the fee calculation behavior of your backend system.

```
feeAmount
```

Resolution

K33 Team: We have a breakdown of the calculations documented in our backend.

L-1 | Failed Liquidations With Low Target LTV

CATEGORY	SEVERITY	LOCATION	STATUS
Warning	LOW	Global	Resolved

Description

K33 has a target LTV for liquidation within its backend system. If the LTV which K33 is targetting has a large delta between the current LTV, all liquidation calls will fail as more amountIn is necessary than is approved and available for swap.

Recommendation

Clearly document this behavior and appropriately set the target LTV.

Resolution

K33 Team: This is dependent on providing the correct parameters, we have found a target range that works for us and can adjust it to lesser deltas if needed.

L-2 | No Events Emitted On Funds Rescue

CATEGORY	SEVERITY	LOCATION	STATUS
Event	LOW	L-Vanir.sol	Acknowledged

Description

Two new functions has been added to the contract to enable the admin to reduce funds - `'transfer'`

```
'transfer'
```

Recommendation

Consider if you should emit events in these two functions.

Resolution

K33 Team: We have no needs for these events, we utilize a blockchain listener that tracks all K33-related activity.

L-3 | Frozen Pool Will Lead To Fail Liquidations

CATEGORY	SEVERITY	LOCATION	STATUS
Warning	LOW	Global	Acknowledged

Description

When the reserve configuration in `SparkLend` `isFrozen`, supplying liquidity is prevented by repays and withdraws are permitted. If the configuration were set to frozen, most K33 liquidations would fail since function `swap()` attempts to supply liquidity that wasn't used as part of the swap cost.

```
SparkLend
isFrozen
swap()
```

Recommendation

Clearly document this risk.

Resolution

K33 Team: Acknowledged.

Summary of Recommendations

Based on our comprehensive audit, we provide the following prioritized recommendations to improve the security posture of K33Loans.

Priority Matrix

ISSUE ID	TITLE	SEVERITY	PRIORITY
M-0	User Can Pass Arbitrary Fee Amount	MEDIUM	Medium
M-1	Failed Approvals With USDT	MEDIUM	Medium
M-2	Lost Borrow Fee	MEDIUM	Medium
L-0	No Controls On Liquidation Fee Amount	LOW	Low
L-1	Failed Liquidations With Low Target LTV	LOW	Low
L-2	No Events Emitted On Funds Rescue	LOW	Low
L-3	Frozen Pool Will Lead To Fail Liquidations	LOW	Low

General Security Best Practices

- ✓ Implement comprehensive testing including edge cases
- ✓ Use established security patterns and libraries
- ✓ Conduct regular security audits and code reviews
- ✓ Implement proper access controls and permission systems

Audit Team

Team Credentials

Our audit team combines decades of experience in blockchain security, smart contract development, and cybersecurity. Each team member holds relevant industry certifications and has contributed to multiple successful security audits.

Methodology & Standards

Our audit methodology follows industry best practices and standards:

- ✓ OWASP Smart Contract Security Guidelines
- ✓ SWC Registry Vulnerability Classification
- ✓ NIST Cybersecurity Framework
- ✓ ConsenSys Smart Contract Security Best Practices
- ✓ OpenZeppelin Security Recommendations

Audit Process

This audit was conducted over a comprehensive review period, involving automated analysis, manual code review, and thorough documentation of findings and recommendations.

Disclaimer & Legal Notice

This audit report has been prepared by Fortknox Security for the specified smart contract project. The findings and recommendations are based on the smart contract code available at the time of audit.

Scope Limitations

- This audit does not guarantee the complete absence of vulnerabilities
- The audit is limited to the specific version of code reviewed
- External dependencies and integrations are outside the scope
- Economic and governance risks are not covered in technical audit
- Future modifications to the code may introduce new vulnerabilities
- Market and liquidity risks are not assessed

Liability Statement

Fortknox Security provides this audit report for informational purposes only. We do not provide any warranties, express or implied, regarding:

- The absolute security of the smart contract
- The economic viability of the project
- The legal compliance in any jurisdiction
- Future performance or behavior of the contract
- Third-party integrations or dependencies

Legal Terms & Usage Rights

Usage Rights

This audit report may be used by the client for:

- ✓ Public disclosure and transparency
- ✓ Marketing and promotional materials
- ✓ Investor due diligence processes
- ✓ Regulatory compliance documentation
- ✓ Technical documentation and reference
- ✓ Security assessment presentations
- ✓ Community transparency initiatives

Restrictions

The following restrictions apply to this report:

- ✓ Report content may not be modified or altered
- ✓ Fortknox Security branding must remain intact
- ✓ Partial excerpts must maintain context and accuracy
- ✓ Commercial redistribution requires written permission
- ✓ Translation must preserve technical accuracy

Intellectual Property

This report contains proprietary methodologies and analysis techniques developed by Fortknox Security. The format, structure, and analytical approach are protected intellectual property.

Contact Information

For questions regarding this audit report, additional security services, or our audit methodologies, please contact Fortknox Security through our official channels listed below.

Fortknox Security

🌐 https://www.fortknox-security.xyz

🐦 @FortKnox_sec

📧 support@fortknox-security.xyz

FORTKNOX SECURITY

Web3 Security at Fort Knox Level

Contact Us

🐦 @FortKnox_sec

📱 @FortKnox_sec

🌐 fortknox-security.xyz

📧 support@fortknox-security.xyz

Audit performed by
Fortknox Security