Kryptoin Smart Contract final Audit Report

Scope of Audit

The scope of this audit was to analyze and document the **Kryptoin** smart contract codebase,which is fork of aave Protocol for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistical analysis, Theo.

Issue Categories

Every issue in this report has been assigned a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high-severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arises because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impacts and are just warnings that can remain unfixed. It would be better to fix these issues at some point in the future.

Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

The number of security issues per severity.

TYPE	HIGH	MEDIUM	LOW	INFORMATIONAL
Open	0	0	0	1
Acknowledged	0	0	1	0
Closed	0	0	4	0

Introduction

During the period of **1st April 2023** to **7th June 2023** Initial Audit is completed

QuillAudits Team performed a security audit for the **Kryptoin** smart contract.

Code Base:
https://github.com/Proxy-Protocol/Redux-BTC-Markets/tree/redux-v2/contracts

Fixed In:
https://github.com/Proxy-Protocol/Redux-BTC-Markets/commit/6f2548a05eda43284bde23de5f2f6b919f93e03c

Issues Found – Code Review / Manual Testing

High Severity Issues

None

Medium Severity Issues

None

Low Severity Issues

L.1 Inherited OwnableUpgradeable uses single-step ownership transfer

Description
During the code review, It has been noticed that mostly all contracts use single-step ownership transfer on the OwnableUpgradeable contract.

Recommendation
Consider using the Ownable2StepUpgradable contract in the implementation.

Status: Acknowledge

L.2 Remove garbage

Description
In **Ptoken.sol** contract hardhat/console.sol is imported.

Recommendation
In general, hardhat/console.sol is used only for testing and debugging, the deployed version of the contracts should not contain that.
We recommend removing the import.

Status: fixed

L.3 Sanity checks are missing

Description
zero sanity checks missing in all functions of **PrxyTreasury.sol**

Recommendation
Add sanity checks for the input parameters and it's always better to take precautions instead of trusting the owner.

Status: fixed

L.4 SafeTransfer Methods missing

Description
It is good to add a require() statement that checks the return value of token transfers or to use something like a safeTransfer/safeTransferFrom unless one is sure the given token reverts in case of a failure. Failure to do so will cause silent failures of transfers and affect token accounting in the contract.

Recommendation:
Replace transferFrom() with safeTransferFrom() as ERC20 token implementation can be any. If transferFrom() does not return a value (e.g., USDT), the transaction reverts because of a decoding error. Revert without error.

Status: fixed

L.5 Setters are missing

Description
In **PrxyTreasury.sol** setters for changing the owner and pool address are missing.

Recommendation:
Add setters for pool and owner such that if in future the rights need to transfer to multi-sig it can be done easily.

Status: fixed

Informational Issues

I.1 Recommendations and Gas optimizations

- For test stake use smock Library.
- Gas optimization
    - The pre-increment operation is cheaper (about 5 GAS per iteration) so use ++i instead of i++ or i+= 1 in for loop. We recommend using pre-increment in all the for loops.
    - != 0 costs 6 less GAS compared to > 0 for unsigned integers in require statements with the optimizer enabled. We recommend using !=0 instead of > 0 in all the contracts.
    - In for loop the default value initialization to 0 should be removed from all the for loops.
    - In the EVM, there is no opcode for non-strict inequalities (>=, <=) and two operations are performed (> + =.) Consider replacing >= with the strict counterpart >. Recommend following the inequality with a strict one.
    - All the public functions which are not used internally need to be converted to external.

Status: **Partially fixed**

Functional Tests

Some tests performed are mentioned below:

- Initial setup checked. ✅
- Deposit and supply positive and negative scenarios. ✅
- Withdraw and borrow positive and negative scenarios. ✅
- Liquidation positive and negative scenarios. ✅
- Flash loan disabled. ✅
- Treasury prxyToken transfers and deposits. ✅
- claimPrxy interest calculation. ✅
- Prxy supply and borrow mode scenarios. ✅

Closing Summary

4 low issues are fixed and 1 informational issue is partially fixed. The code is good to deploy on the mainnet.

Disclaimer:

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **Kryptoin**. This audit does not provide a security or correctness guarantee for the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multi-step process. One audit cannot be considered enough. We recommend that the **Kryptoin** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.