PantherFi Smart Contract
Audit Report

Fri Apr 05 2024

contact@scalebit.xyz https://twitter.com/scalebit_

ScaleBit

PantherFi Smart Contract Audit Report

1 Executive Summary

1.1 Project Information

Description	PantherFi is a decentralized platform that merges DeFi and AI computing. This fusion introduces new computational asset classes and revenue streams to the DeFi ecosystem, while simultaneously offering decentralized financial services to AI computing power
Type	Lending
Auditors	ScaleBit
Timeline	Wed Mar 06 2024 - Tue Mar 19 2024
Languages	Solidity
Platform	Ethereum
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/PantherFi/lending/
Commits	9a4b92ae5e27e772941f5a702c1c004d7facfb4b 8c9174b697a2168dadf201df96dab27920ddd3d4 fa8f8cc0a8695a005c308d15218f5430242e033a 0841793964c909da7a035ce99b82bd90cb269b3d 4dd79afa7d16fc3df29dc0021a562dedce6a9145

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
CE2	src/CErc20.sol	ef46042d1de3f97fd07697b160f1aa8eb05d832f
PFI	src/Governance/PantherFi.sol	58e6d9545c5fa9266c490050729aafb08b93a2b2
PPO	src/PythPriceOracle.sol	d4e9a7c6f7cf14b4423a73c4c7f0317e241c8b3a
PIN	src/PythInteraction.sol	6bd70da81726ef34f78607d67903b73cef2595fe
SCTIS	src/CTokenInterfaces.sol	89c6b922860d002d6651b0d67481a85a8aeeb930
CTO	src/CToken.sol	3584e3e222b33994044feb564cecbde81bf84994
CIN	src/ComptrollerInterface.sol	f3111fea47ba24fe3d45f0cf614dc16ce570978c
CST	src/ComptrollerStorage.sol	87fe6a9ab7a0d3c5a7af723ede922987020682da
CNTU	src/CNativeTokenUpgradeable.sol	a72acdae2c7b56722f8c119b0d0052e8589960f9
COM	src/Comptroller.sol	6b4b1f8d57583475d08540e8086bf03989ac0530

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	2	1
Informational	0	0	0
Minor	1	1	0
Medium	0	0	0
Major	2	1	1
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by PantherFi to identify any potential issues and vulnerabilities in the source code of the PantherFi smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 3 issues of varying severity, listed below.

ID	Title	Severity	Status
CTO-1	First Deposit Bug	Major	Fixed
PFI-1	Centralization Risk	Major	Acknowledged
PPO-1	Lack of Events Emit	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the PantherFi Smart Contract :

Admin

- The Admin can update the parameters of the interest rate model through `updateJumpRateModel()`.
- The Admin can initialize the money market through `initialize()`.
- The Admin can begin the transfer of admin rights through `_setPendingAdmin()`.
- The Admin can accept the transfer of admin rights through `_acceptAdmin()`.
- The Admin can set a new comptroller for the market through `_setComptroller()`.
- The Admin can accrue interest and set a new reserve factor for the protocol through `_setReserveFactor()`.
- The Admin can sweep accidental `ERC-20` transfers to this contract through `sweepToken()`.
- The Admin can accrue interest and reduce reserves through `_reduceReserves()`.
- The Admin can accrue interest and update the interest rate model through `_setInterestRateModel()`.
- The Admin can set the `allowPureInteraction` through `_setAllowPureInteraction()`.
- The Admin can set the `Ctoken` Price id through `addCtokenPriceId()` and `setCtokensPriceIds()`.
- The Admin can set a new price oracle for the comptroller through `_setPriceOracle()`.
- The Admin can set the close factor used when liquidating borrows through `_setCloseFactor()`.
- The Admin can set the collateral factor for a market through `_setCollateralFactor()`.
- The Admin can set liquidation incentive through `_setLiquidationIncentive()`.
- The Admin can add the market to the markets mapping and set it as listed through `_supportMarket()`.

- The Admin can set the given borrow caps for the given `cToken` markets through `_setMarketBorrowCaps()`.
- The Admin can change the `borrowCapGuardian` through `_setBorrowCapGuardian()`.
- The Admin can change the `pauseGuardian` through `_setPauseGuardian()`.
- The Admin can change the `mintGuardianPaused` through `_setMintPaused()`.
- The Admin can change the `borrowGuardianPaused` through `_setBorrowPaused()`.
- The Admin can change the `transferGuardianPaused` through `_setTransferPaused()`.
- The Admin can change the `seizeGuardianPaused` through `_setSeizePaused()`.
- The Admin can change brains through `_become()`.
- The Admin can fix the bad accruals through `fixBadAccruals()`.
- The Admin can set `COMP` borrow and supply speeds for the specified markets through `_setIncentiveSpeeds()`.
- The Admin can set `COMP` speed for a single contributor through `_setContributorIncentiveSpeed()`.
- The Admin can set the incentive token through `_setIncentiveToken()`.
- The Admin can open the claim incentive through `_openClaimIncentive()`.

User

- The User can approve `spender` to transfer up to `amount` from `src` through `approve()`.
- The User can transfer `amount` tokens from `msg.sender` to `dst` through `transfer()`.
- The User can transfer `amount` tokens from `src` to `dst` through `transferFrom()`.
- The User can delegate votes from `msg.sender` to `delegatee` through `delegate()`.
- The User can delegate votes from signatory to `delegatee` through `delegateBySig()`.
- The User can suppliy assets into the market and receives cTokens in exchange through `mint()`.
- The User can redeem cTokens in exchange for the underlying asset through `redeem()`.

- The User can redeem cTokens in exchange for a specified amount of underlying asset through `redeemUnderlying()`.
- The User can borrow assets from the protocol to their own address through `borrow()`.
- The User can repay their own borrow through `repayBorrow()`.
- The User can repay a borrow belonging to borrower through `repayBorrowBehalf()`.
- The User can liquidate the borrowers collateral through `liquidateBorrow()`.

4 Findings

CTO-1 First Deposit Bug

Severity: Major

Status: Fixed

Code Location:

src/CToken.sol#487

Descriptions:

The exchange rate contains a critical bug that can be exploited to steal funds of initial depositors of a freshly deployed `CToken` market. The formulas can be simplified and written as below:

```
Exchange rate = underlying.balanceOf(CToken) * 1e18 / CToken.totalSupply()


CToken amount = User deposit amount / Exchange rate
```

if the exchange rate can be increased to a value greater than the user's deposit, the `CToken` output amount comes out to be 0.

As the exchange rate is dependent upon the ratio of CToken's total supply and the underlying token balance of the `CToken` contract, the attacker can craft transactions to manipulate the exchange rate.

Steps to attack:

Once the `CToken` has been deployed and added to the lending protocol, the attacker mints the smallest possible amount of CTokens.

Then the attacker does a plain underlying token transfer to the CToken contract, artificially inflating the underlying.balanceOf(CToken) value.

Due to the above steps, during the next legitimate user deposit, the mintTokens value for the user will become less than 1 and essentially be rounded down to 0 by Solidity. Hence the user gets 0 CTokens against his deposit and the CToken's entire supply is held by the Attacker.

The Attacker can then simply redeem his `CToken` balance for the entire underlying token balance of the `CToken` contract.

The same steps can be performed again to steal the next user's deposit.

It should be noted that the attack can happen in two ways:

The attacker can simply execute steps 1 and 2 as soon as the `CToken` gets added to the lending protocol. The attacker watches the pending transactions of the network and front runs the user's deposit transaction by executing steps 1 and 2 and then back runs it with step 3.

A sophisticated attack can impact all initial user deposits until the lending protocols owners and users are notified and contracts are paused. Since this attack is a replicable attack it can be performed continuously to steal the deposits of all depositors that try to deposit into the new `CToken` contract.

The loss amount will be the sum of all deposits done by users into the `CToken` multiplied by the underlying token's price.

Suggestion:

The fix to prevent this issue would be to enforce a minimum deposit that cannot be withdrawn. This can be done by minting small amount of CToken units to 0x00 address on the first deposit.

```
function mintFresh(address minter, uint mintAmount) internal {
  // ...
  Exp memory exchangeRate = Exp({mantissa: exchangeRateStoredInternal()});

  uint actualMintAmount = doTransferIn(minter, mintAmount);

  uint mintTokens = div_(actualMintAmount, exchangeRate);

  /// THE FIX
  if (totalSupply == 0) {
     totalSupply = 1000;
     accountTokens[address(0)] = 1000;
     mintTokens -= 1000;
  }
```

```
    totalSupply = totalSupply + mintTokens;
    accountTokens[minter] = accountTokens[minter] + mintTokens;
    // ...
}
```

Instead of a fixed 1000 value an admin-controlled parameterized value can also be used to control the burn amount on a per CToken basis.

Alternatively, a quick fix would be that the protocol owners perform the initial deposit themselves with a small amount of underlying tokens and then burn the received CTokens permanently by sending them to a dead address.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

PFI-1 Centralization Risk

Severity: Major

Status: Acknowledged

Code Location:

src/Governance/PantherFi.sol

Descriptions:

Centralization risk was identified in the smart contract.

- The Admin can mint 100 million tokens to any address when initialized.

Suggestion:

It is recommended to take measures to mitigate this issue.

Resolution:

The client replies that The original code of the `Compound` project is like this, and the project team has not made any modifications. It will extract all the `Pantherfi` tokens and release them through a multi-signature account.

PPO-1 Lack of Events Emit

Severity: Minor

Status: Fixed

Code Location:

src/PythPriceOracle.sol#28,38,61

Descriptions:

The smart contract lacks appropriate events for monitoring sensitive operations, which could make it difficult to track sensitive actions or detect potential issues.

Suggestion:

It is recommended to emit events for those sensitive functions.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.