Security Assessment

Bend

Apr 1st, 2022

Table of Contents

Summary

This report has been prepared for Bend to discover issues and vulnerabilities in the source code of the Bend project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Bend
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/bnftdao/bnft-protocol https://github.com/BendDAO/bend-protocol https://github.com/BendDAO/bend-incentive
Commit	4ffbab2f162391247a393c7f4f764530b932de83 a8d7b4246b4982994d00a2fc9b97024d8120ebeb b0620c3e6d41df69358f6765fbca089ecc59ebeb

Audit Summary

Delivery Date	Apr 01, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	0	0	0	0	0	0	0
Major	6	0	0	4	0	0	2
Medium	5	0	0	1	0	0	4
Minor	2	0	0	0	0	0	2
Informational	13	0	0	3	0	0	10
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
IIC	bend-incentive/contracts/incentives/interfaces/IIncentivesController.sol	ad29ec69301b84c53b83c7266db33624bebf1aaeb3c0f590ffccba72da9ad595
ISB	bend-incentive/contracts/incentives/interfaces/IScaledBalanceToken.sol	e0efd10fd17e1683044ab7b67d5d6dd57451f1efa25948ef35798c58d593754d
BPI	bend-incentive/contracts/incentives/BendProtocolIncentivesController.sol	fc5109367f3c9d17ffcd50b09e9fc9dd6ea336c1796177341561fcea5208d8f9
DMC	bend-incentive/contracts/incentives/DistributionManager.sol	ec167114e34bb636b376576c313d2ba227deee9e6e58adf50356ad143bbbec4d
DTC	bend-incentive/contracts/incentives/DistributionTypes.sol	6b42cc6e2ec6b2cb06463201eb1d835e9785c2696c4a34c061746b9ca14cd949
ERC	bend-incentive/contracts/libs/ERC20Detailed.sol	817931354353d411235aa4c0db2cc4ca8a3d553916b9a0c79669f6dc33fccb5d
IVC	bend-incentive/contracts/token/interfaces/IVault.sol	3abadf740d32597140c3302504f984063a3c81d4c01a99595ab6d4c5a871628a
BTC	bend-incentive/contracts/token/BendToken.sol	fccd6985dacd3cab85601f8bc3464c5c4236d5cda517fcb61b112090eea11e28
VCK	bend-incentive/contracts/token/Vault.sol	f3e2ba169e34a1d6091d246cdfd1906014ab5c03fbc2693bf3a49ee0b7ce4aec
ISW	bend-incentive/contracts/vote/interfaces/ISmartWalletChecker.sol	7d5629bab2fdacede8d3fc2fa716443db8482de76120e248d77dc8e37aca3840
VBC	bend-incentive/contracts/vote/VeBend.sol	43bd0c9839421ca33c0138dcd3636cb2648bbaf78f666e55792511d89ac2dd59
BTA	bend-protocol/contracts/deployments/BTokensAndBNFTsHelper.sol	c91d6a4e84b22c7dfb4efc14dd316ea21fbb264fc42fe02f8e4f05c12a443cc9
IBN	bend-protocol/contracts/interfaces/IBNFT.sol	16bde64d971b367044fd50687f4c62d655247c78e15edc670094aac9b56c67fa
IBF	bend-protocol/contracts/interfaces/IBNFTRegistry.sol	0b450a37d2257b4a12c223087860ccf6f86b74f2703a0983fdc1628112fb688e

ID	File	SHA256 Checksum
IBT	bend-protocol/contracts/interfaces/IBToken.sol	abd94e9662b113c1d15451c467229e88c0ce15e15ff69fceff43cc65b5111bf7
IDT	bend-protocol/contracts/interfaces/IDebtToken.sol	944a49920842b5c1d6ef3dfeb60bb44e9b09289f01953e6c12d895b915d24d05
IER	bend-protocol/contracts/interfaces/IERC20Detailed.sol	b0a26dc3ce7f2465ad5fa684f6b9bb2209f819c9f69d84211b1f52304fd28acf
IEC	bend-protocol/contracts/interfaces/IERC721Detailed.sol	c6636244dc4759b3e0c40c32698ba6772453ff0b165ec0ca6cf097286516842b
IFL	bend-protocol/contracts/interfaces/IFlashLoanReceiver.sol	6e172c67758edf98eeeaaca8ec06868ec33c604fa1bbe20c17f7de14a8f553a3
IIK	bend-protocol/contracts/interfaces/IIncentivesController.sol	e21bf9b76d099b8fda2b6e12c7f10c9942c49ca3ed965808985c9a9fb9de8c0e
IIR	bend-protocol/contracts/interfaces/IInterestRate.sol	e91f29d342715feebd513d17cbf90c15d79c2107b54cbb5411fd70129b316682
ILP	bend-protocol/contracts/interfaces/ILendPool.sol	fb38b5bf7cad06c1d1997f79ff271700de4a78513a603e1bdbe3562f80773cfe
ILA	bend-protocol/contracts/interfaces/ILendPoolAddressesProvider.sol	63d315220d391b20847d41b47de7241b0d0573d98984654e29da4a86d1fafeca
ILR	bend-protocol/contracts/interfaces/ILendPoolAddressesProviderRegistry.sol	b291d5b7f8899cc99ba8a4a35538bb19c28549d7cbb354287fdf2db5521ea33e
ILC	bend-protocol/contracts/interfaces/ILendPoolConfigurator.sol	df64bb4cb52e2865b9c8a425c0ae8261b46a303ae9c01de91b8f5bee4bda5c46
ILL	bend-protocol/contracts/interfaces/ILendPoolLiquidator.sol	14eb13a6febdfcfdcb3a5e8ffe0e092699c830ae800e172214a475348da6761b
ILK	bend-protocol/contracts/interfaces/ILendPoolLoan.sol	62de555b78d24a45ae36b4ef5fe4a9ecb21c34b462c31d8aaaab2ad56c61bbe3
INO	bend-protocol/contracts/interfaces/INFTOracle.sol	0d8b0522072d846086d87775199fe7b01eb04cd5c92054ef2c17573cf74331ba
ING	bend-protocol/contracts/interfaces/INFTOracleGetter.sol	9028c2e679ac5928e47db5759d8b41f9e3dd22af991d9a31c771270f259094b3

ID	File	SHA256 Checksum
IPG	bend-protocol/contracts/interfaces/IPunkGateway.sol	c10c2bc141f741c5aaec276ff11531e6c02eb397c094da9d787bb3807de73d9c
IPC	bend-protocol/contracts/interfaces/IPunks.sol	c7c2d0fe51903324e10a070408aa4272702cedbe9a49b0e8927147ba7f50b31e
IRO	bend-protocol/contracts/interfaces/IReserveOracleGetter.sol	b3c6bfbaa31176e809a3ae1e85b7270f4f432667b9d184de48ae15b83ba78c2e
IST	bend-protocol/contracts/interfaces/IScaledBalanceToken.sol	a0e7e2c1b7a25d325c5e4155558f58a4d617698ae4020725cda5ccb25b73abe6
IUP	bend-protocol/contracts/interfaces/IUiPoolDataProvider.sol	3ff0d0625af70445bb4fc16d80d74289caa308753cedded12c59f4ebc94a6f62
IWE	bend-protocol/contracts/interfaces/IWETH.sol	419d5909ff3f2312c27369d7300eb46555a46c46cc7767c936254c0d8256af85
IWT	bend-protocol/contracts/interfaces/IWETHGateway.sol	9855a2af5d6122b4e79fc41e4e7e4e71c16c62ee201b497ac522bf8ef7e56d7e
IWP	bend-protocol/contracts/interfaces/IWrappedPunks.sol	44aff6c0226da95b62b4a6e590f089721081db8f36f66765aebfb728348669e7
NCC	bend-protocol/contracts/libraries/configuration/NftConfiguration.sol	1c8f0c16e2ba3cdd0d19ecc5acb7f802253993a75f5629dcdc08a2576a3c9c97
RCC	bend-protocol/contracts/libraries/configuration/ReserveConfiguration.sol	fe92c46456347a2d32165320353c8ca2175bed01ecfef7c0d77efed33e9ff7a3
ECK	bend-protocol/contracts/libraries/helpers/Errors.sol	a950909f078ae2fd17822dbd089ca7f95abd1769ddee107d6ec21b9fd40d7025
GLC	bend-protocol/contracts/libraries/logic/GenericLogic.sol	7521f671929fd70133dc8ada4d31bc817adb36ea4d36356eb52f04f4c32694c9
NLC	bend-protocol/contracts/libraries/logic/NftLogic.sol	266e9560406d67cf9e83361f225139d6de298433c038bab34420e88234053f4f
RLC	bend-protocol/contracts/libraries/logic/ReserveLogic.sol	a0b0b905d15e37f0fec6541ad1d2b7a535257ece73ce17f6d3dc4721bc55598c
VLC	bend-protocol/contracts/libraries/logic/ValidationLogic.sol	154b0d877644d549b9b3f392db48b02fb0c45c1abc1caab0dbf268147d600d85

ID	File	SHA256 Checksum
MUC	bend-protocol/contracts/libraries/math/MathUtils.sol	594d87b7bc28ecac6b407d910d7b71f0d9498606265f5296172a7d6ff295c154
PMC	bend-protocol/contracts/libraries/math/PercentageMath.sol	88bceff169a2149859e8cfbe4292d1dda78a3e8343022022bc3a27e4b8c7220f
WRM	bend-protocol/contracts/libraries/math/WadRayMath.sol	3cc5937901290db64aa0791880c3eb0ed903045be855bf8af071b5617e60ef91
BPA	bend-protocol/contracts/libraries/proxy/BendProxyAdmin.sol	e47ab008f6328f56b5595bcf070d19c9b765fae2aca7dfde759ae0b4aec29c11
BUP	bend-protocol/contracts/libraries/proxy/BendUpgradeableProxy.sol	979ab5513b4f90d447c7ed0d294d90308b8979ac1468d5b9e6b0550e694dbbb5
DTK	bend-protocol/contracts/libraries/types/DataTypes.sol	83ac695576f637b7e533057859e07419f9895a6fe850675e7d8e883c047b2f9b
BCC	bend-protocol/contracts/misc/BendCollector.sol	46cf7bec92cc67265a18ddf0ba545018a0363658a2a0cf37c48ddb960c98d237
BPD	bend-protocol/contracts/misc/BendProtocolDataProvider.sol	0f35c84d05098e327c88738dacca8edeb2e1c6c34ff0aeffb835b13b6f50c287
UPD	bend-protocol/contracts/misc/UiPoolDataProvider.sol	4ae2d0d60f32093227a28967874406159c41fa73111568168a56ac5a0ca66f6e
WBP	bend-protocol/contracts/misc/WalletBalanceProvider.sol	65d295aecab383f1a727527b1bd0b6a2cdb7359f47c9a661a377aca9fe2117a8
BTK	bend-protocol/contracts/protocol/BToken.sol	c98b685a25e5c87c3b8f08a5327856e7fdeb94f6c09a048274e71e81d389d972
DTP	bend-protocol/contracts/protocol/DebtToken.sol	080006d630a091be985934111f118c2ff00a2945117148d593945b236cd8a73b
ETR	bend-protocol/contracts/protocol/EmergencyTokenRecovery.sol	997882fdc238ee517561b41f7b8972f15d76872c2972abd53afd0618fa093029
IEK	bend-protocol/contracts/protocol/IncentivizedERC20.sol	bfdac58b9e5b5e9972ed556c95c5368edc6ac3b98c26d8a92b9dba480685ac19
IRC	bend-protocol/contracts/protocol/InterestRate.sol	23f4cc877a3e5cfcbffef52fa6f767303068840945ee1fc3b31594f802b02054

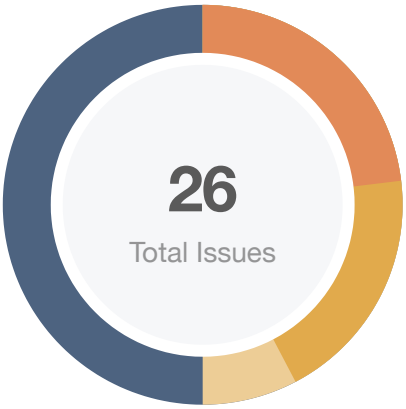
ID	File	SHA256 Checksum
LPC	bend-protocol/contracts/protocol/LendPool.sol	87025317a198c01b72277287c0553a29763d4d21ca2c9ad0304addf41acd3ad5
LPA	bend-protocol/contracts/protocol/LendPoolAddressesProvider.sol	5bd01de467a866bcfa03aaab03f946b381aeb8a78f84860b4c5d00d507dbc4d7
LPP	bend-protocol/contracts/protocol/LendPoolAddressesProviderRegistry.sol	0f3433d9ff04629a34d672b933caea0f35c471779349b786668f917b53877320
LPK	bend-protocol/contracts/protocol/LendPoolConfigurator.sol	3f7c7235de421cc883fee3992d0d337436d711805a4eb2ef72bd4a861d4b3f7e
LPL	bend-protocol/contracts/protocol/LendPoolLiquidator.sol	727557d58007afd4c74013997ed99ab75ccfdcad662cf51564eab2fca5c6c82a
LLC	bend-protocol/contracts/protocol/LendPoolLoan.sol	7d891ebc7da25c354ebf8b8f6862b5ae0c235633400b37f31c437cb358c9a386
LPS	bend-protocol/contracts/protocol/LendPoolStorage.sol	98da537a21ef0a81da9c1cb4ddc47584f23015810a2f297326e19c5cecbbfdf3
NFT	bend-protocol/contracts/protocol/NFTOracle.sol	085a83d2f7218f37ff1ef1815fbb1fce2d940ce7cb87c74c4972c80c59c4baff
PGC	bend-protocol/contracts/protocol/PunkGateway.sol	a8e6250a12f928516cadd927a9769ffb3377a755e0a251cdf2834ae71422478f
ROC	bend-protocol/contracts/protocol/ReserveOracle.sol	9155c80b126c3998b857d89a0f4b4ade0d24ee6609d3fb5e158bb20255d9561f
WET	bend-protocol/contracts/protocol/WETHGateway.sol	9d2046db6b55f18210cbdb3e78d52a288f92e82cbd03b0809959623d4af8c8c3
BCK	bend-protocol/contracts/utils/BlockContext.sol	9ab7a788aa457f3806d02b592158f0d3cb085bbe3b0c300ebfaef395b5fb1056
IBC	bnft-protocol/contracts/interfaces/IBNFT.sol	329b26ca46b1e8d8a95a44225bad27d9411f66cbcf719cfe7c1fe17d8803aa8f
IBR	bnft-protocol/contracts/interfaces/IBNFTRegistry.sol	d82cff4e281079cef4e671aafecfa244b9be0783862190393bceece8750d35bc
IED	bnft-protocol/contracts/interfaces/IERC721Detailed.sol	03150141c23e0f12030cf1231ffa63194a0548089b73061feea9288dd774d5a8

ID	File	SHA256 Checksum
IFR	bnft-protocol/contracts/interfaces/IFlashLoanReceiver.sol	185758549f6a5a7337c2337ad8788484e971bfa10d3eb33761a0c7ae35238c98
BNF	bnft-protocol/contracts/libraries/BNFTProxyAdmin.sol	efc41b718eee605f14ecebfb1adea2c0181678180dcc92598f0e88277de8f379
BNT	bnft-protocol/contracts/libraries/BNFTUpgradeableProxy.sol	8ef093f2436ec7a05178aa8e84125a0db36837d9677011c0eedf2bfe94a4fdbd
AFL	bnft-protocol/contracts/misc/AirdropFlashLoanReceiver.sol	95e9b6d18bce537f784c0949333c1fdf77b9da949577bf937ba679be8955096d
BNC	bnft-protocol/contracts/protocol/BNFT.sol	88477da4961181df33c7d1fae2962e3d7bb6d9fc2776178067e90cb65076d1f5
BNR	bnft-protocol/contracts/protocol/BNFTRegistry.sol	5694ad31984688e8eb7110d694119bb16b2cc9d8b80d17ef9dad8060b528c499

Financial Models

Financial models of blockchain protocols need to be resilient to attacks. It needs to pass simulations and verifications to guarantee the security of the overall protocol. Financial models are not in the scope of the audit.

Findings



Critical	**0** (0.00%)
Major	**6** (23.08%)
Medium	**5** (19.23%)
Minor	**2** (7.69%)
Informational	**13** (50.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Potential Price Oracle Risk	Control Flow	Major	Acknowledged
GLOBAL-02	Third Party Dependencies	Volatile Code	Medium	Acknowledged
GLOBAL-03	Risk of Supported NFT Digital Assets	Logical Issue	Medium	Resolved
GLOBAL-04	Unlocked Compiler Version	Language Specific	Informational	Resolved
GLOBAL-05	Financial Models	Logical Issue	Informational	Acknowledged
AFL-01	Function Visibility Optimization	Gas Optimization	Informational	Resolved
BNC-01	No Fee for Flashloan	Volatile Code	Minor	Resolved
BNR-01	Lack of Access Restriction	Control Flow	Minor	Resolved
BPI-01	Lack of `msg.sender` Validation	Volatile Code	Informational	Resolved
BTC-01	Initial Token Distribution	**Centralization / Privilege**	**Medium**	Resolved
CKP-01	Centralization Related Risks	**Centralization / Privilege**	**Major**	Acknowledged
CKP-02	Redundant Statements	Volatile Code	Informational	Resolved
CKP-03	Variables That Could Be Declared as Immutable	Gas Optimization	Informational	Resolved
CKP-04	Missing Emit Events	Coding Style	Informational	Resolved

ID	Title	Category	Severity	Status
CON-01	Incorrect Borrower Validation	Logical Issue	Major	Resolved
CON-02	Declaration Naming Convention	Coding Style	Informational	Resolved
COR-01	Functions With `_` as Name Prefix Are Not `private` or `internal`	Coding Style	Informational	Resolved
COT-01	Any User Can Claim Airdrop Tokens	Control Flow	Medium	Resolved
LPL-01	Auction Could Fail Due to Price Fluctuations	Control Flow	Major	Acknowledged
LPL-02	Lack of Validation in the Functions `redeem()/liquidate()`	Logical Issue	Informational	Acknowledged
NFT-01	NFT Asset Price Not Based On Token ID	Control Flow	Informational	Acknowledged
PRT-01	Potential risks in the `DebtToken.mint()`	Logical Issue	Major	Resolved
PRT-02	Centralization Risks in Control Flow	**Centralization / Privilege**	**Major**	Acknowledged
VBC-01	Usage of Require Instead of Assert	Language Specific	Informational	Resolved
VBC-02	Missing Error Messages	Coding Style	Informational	Resolved
WBP-01	Locked Ether	Language Specific	Medium	Resolved

GLOBAL-01 | Potential Price Oracle Risk

Category	Severity	Location	Status
Control Flow	Major	Global	Acknowledged

Description

The `Bend` protocol allows users to borrow assets using NFT as the collateral. If an attacker manipulates the NFT price in the third-party market, so that the NFT price rises to a very high level, the attacker can borrow assets from the pool that exceed the true value of the collateral.

In the end, the `Bend` protocol could not recover the borrowed assets through auction liquidation. In the worse case, the pool will not have enough assets for the depositors to withdraw their tokens, and the depositors will suffer unexpected loss.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

The team acknowledged this issue and they stated the following:

1. The team will prune too low or too high floor price, following the rules of `abs(percent)<=N%`.
2. Auction price must be greater than the total debt value(principal + interest).

GLOBAL-02 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Medium	Global	Acknowledged

Description

The contract is serving as the underlying entity to interact with third-party PUNK, Opensea, SmartWalletChecker, NFT protocols. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to arbitrarily mint assets, lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of Bend protocol requires interaction with PUNK, Opensea, SmartWalletChecker, NFT, etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

The team acknowledged this issue and they stated the following:

1. Only BlueChip NFTs will be listed in Bend protocol through community voting.
2. The team will constantly monitor the statuses of 3rd parties.

GLOBAL-03 | Risk Of Supported NFT Digital Assets

Category	Severity	Location	Status
Logical Issue	Medium	Global	Resolved

Description

In function `borrow()`, there is no restriction on the type of collateral NFT assets to be pledged.

However, it is hard for borrowers to estimate the real value of any type of NFTs.

Besides, the liquidity and audience of the NFT assets will greatly influence the outcome of the auction when the borrower's collateral assets are to be liquidated.

Recommendation

We recommend using a whitelist for NFT assets to be pledged in the contract to mitigate the potential risks.

Alleviation

The team confirmed the current implementation aligns with the original project design, and stated the following:

1. They are using a whitelist for bluechip NFT collections through community voting.
2. They are using floor price and collateral ratio to limit the max borrow amount of NFT assets.

GLOBAL-04 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	Global	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to different compiler versions. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.8.0` the contract should contain the following line:

```
pragma solidity 0.8.0;
```

Alleviation

The team heeded our advice and resolved this issue in commits:

- https://github.com/BendDAO/bend-protocol/commit/68d2c11b720bf2b83e34bd4f8d58a9590e8bd864
- https://github.com/BendDAO/bend-incentive/commit/95bdb11b6dabef2310bc40b673e5c27a4ea8062d
- https://github.com/BoundNFT/boundnft-protocol/commit/98c0e4958ce7e0d8c987f0fd7257c79d86e362b9

GLOBAL-05 | Financial Models

Category	Severity	Location	Status
Logical Issue	Informational	Global	Acknowledged

Description

Bend is a decentralized non-custodial NFT-backed borrowing and lending protocol where users can participate as depositors or borrowers.

The liquidation will be triggered when the threshold price is smaller than borrowing with interest, it has the following potential issues:

1. When the NFT price goes lower and the accumulated debt goes higher than the value of the NFT, as a result, no one will bid to the liquidate auction. The only way is to wait for the NFT price to rise.
2. If the NFT price keeps lower than the debt for a long period, the interest keeps accumulating, making the debt higher and higher. As a result, it is possible that the NFT will keep unclaimed because the increase in debt could be higher than the increase in the price of NFT.
3. If the NFT price recovers above the liquidation line, it is no longer eligible for liquidation auction and cannot be liquidated. These problems may recur if the lender does not return the principal.

The borrower adds the NFT to Bend and takes the underlying asset, he can get the benefits when the NFT price rises. However, the borrower can keep the underlying asset without repayment if the NFT price falls. Due to liquidation issues described above, the NFT may not be successfully liquidated, and the Bend protocol loses the underlying assets. As a result, the depositor will suffer unexpected losses. The risk-liabilities mismatch between the borrower and the depositor.

Due to the abovementioned points, the protocol is heavily exposed to the failure of supported token systems as well as market fluctuations.

Financial models of blockchain protocols need to be resilient to attacks. It needs to pass simulations and verifications to guarantee the security of the overall protocol. Financial models are not in the scope of the audit.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

The team acknowledged this issue and they stated the following:

1. The essence of this issue is the liquidity of NFT. NFT is an atomic asset, and its liquidity is not as good as the ERC20 tokens. Bend protocol will use a relatively higher interest rate to guide more ETH into the NFT market to improve the whole market liquidity. As the entire NFT market size becomes larger and has more participants, the liquidity will become better.
2. Bend protocol only accepts high-quality blue-chip NFTs, these NFTs have at least experienced multiple up-down cycles, and have been recognized by the market.
3. When the price of NFT falls, even if the floor price is lower than the total amount of debt, there are still some liquidators who believe that the NFT will subsequently rise and choose to buy NFT at a relatively low price at this time.
4. Bend protocol will monitor the NFT in liquidation. After a certain period of time, it will deal with those NFTs which are unable to be sold in auctions. Bend protocol will use the vault reserves to participate in the liquidation of the purchase of NFT, via the community vote. After the NFT price rises, Bend protocol will sell NFT to obtain funds to re-inject into the vault.
5. The team considers that using the auction mechanism to liquidate is the optimal solution at the current stage. And they will keep optimizing and upgrading the liquidation mechanism.

AFL-01 | Function Visibility Optimization

Category	Severity	Location	Status
Gas Optimization	Informational	bnft-protocol/contracts/misc/AirdropFlashLoanReceiver.sol: 28	Resolved

Description

The following functions are declared as `public`, contain array function arguments, and are not invoked in any of the contracts contained within the project's scope. The functions that are never called internally within the contract should have external visibility.

Recommendation

We advise that the functions' visibility specifiers are set to `external` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

Alleviation

The team heeded our advice and resolved this issue in commit https://github.com/BendDAO/bend-protocol/commit/68d2c11b720bf2b83e34bd4f8d58a9590e8bd864.

BNC-01 | No Fee For Flashloan

Category	Severity	Location	Status
Volatile Code	Minor	bnft-protocol/contracts/protocol/BNFT.sol: 143	Resolved

Description

The function `BNFT.flashLoan()` allows the users to flash loan the escrowed NFT without any fee, and the depositors cannot benefit from the flash loan.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

The team confirmed the current implementation aligns with the original project design, and they stated the following:

There is a restriction that only allows the borrowers to flashloan their own NFTs, not others' NFTs.

BNR-01 | Lack Of Access Restriction

Category	Severity	Location	Status
Control Flow	Minor	bnft-protocol/contracts/protocol/BNFTRegistry.sol: 61	Resolved

Description

The function `createBNFT()` in the aforementioned line can be called by any contract, as it has no access restriction. This enables anyone to register a new BNFT.

Recommendation

We recommend adding the `onlyOwner` modifier for the `createBNFT()`.

Alleviation

The team confirmed the current implementation aligns with the original project design, and stated the following:

The BoundNFT is permissionless, it's the base of NFT liquidity protocols.

BPI-01 | Lack Of `msg.sender` Validation

Category	Severity	Location	Status
Volatile Code	Informational	bend-incentive/contracts/incentives/BendProtocolIncentivesController.sol: 88	Resolved

Description

The function `handleAction()` is missing a sanity check to ensure the `msg.sender` is the configured token contract in `_assets` array.

Recommendation

We recommend adding a validation for the `msg.sender`.

Alleviation

The team heeded our advice and resolved this issue in commit https://github.com/BendDAO/bend-incentive/commit/95bdb11b6dabef2310bc40b673e5c27a4ea8062d.

BTC-01 | Initial Token Distribution

Category	Severity	Location	Status
Centralization / Privilege	**Medium**	bend-incentive/contracts/token/BendToken.sol: 20	Resolved

Description

All of the deployer specified `_amount` tokens are sent to the contract deployer when deploying the contract. This could be a centralization risk as the deployer can distribute all tokens without obtaining the consensus of the community.

Recommendation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Alleviation

The team confirmed the current implementation aligns with the design and they stated the following:

"The initial token distribution plan is documented as below:

https://docs.benddao.xyz/portal/governance/bendenomics#bend-token-distribution

The transaction to distribute the token is as below:

https://etherscan.io/token/0x0d02755a5700414b26ff040e1de35d337df56218?a=0x16729fad4dfd9f7c50f3b52a5deaf842d2c609b7 "

CKP-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	Major	bend-protocol/contracts/deployments/BTokensAndBNFTsHelper.sol bnft-protocol/contracts/protocol/BNFTRegistry.sol bnft-protocol/contracts/protocol/BNFT.sol bend-incentive/contracts/vote/VeBend.sol bend-protocol/contracts/protocol/ReserveOracle.sol bend-protocol/contracts/protocol/PunkGateway.sol bend-protocol/contracts/protocol/WETHGateway.sol bend-protocol/contracts/protocol/NFTOracle.sol bend-protocol/contracts/protocol/LendPoolConfigurator.sol bend-protocol/contracts/protocol/LendPoolAddressesProviderRegistry.sol bend-protocol/contracts/protocol/LendPoolAddressesProvider.sol bend-protocol/contracts/protocol/EmergencyTokenRecovery.sol bend-protocol/contracts/protocol/BToken.sol bend-protocol/contracts/protocol/DebtToken.sol bend-protocol/contracts/protocol/LendPool.sol bend-protocol/contracts/protocol/LendPoolLoan.sol bend-incentive/contracts/token/Vault.sol	ⓘ Acknowledged

Description

In the contract `BTokensAndBNFTsHelper`, the role `owner` has authority over the following functions:

- function `configureReserves()`, to configure reserves.
- function `configureNfts()`, to configure NFTs.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

In the contract `BNFTRegistry`, the role `owner` has authority over the following functions:

- function `setBNFTGenericImpl()`, to set the BNFT contract.
- function `createBNFTWithImpl()`, to create BNFT proxy with already deployed implement, then initialize it.

- function `upgradeBNFTWithImpl()`, to update BNFT proxy with already deployed implement, then initialize it.
- function `addCustomeSymbols()`, to add custom symbol for some special NFTs like CryptoPunks.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

In the contract `BNFT`, the role `owner` has authority over the following functions:

- function `renounceOwnership()`.
- function `transferOwnership()`.
- function `claimERC20Airdrop()`, to claim ERC20 tokens. The claimable token cannot be the `_underlyingAsset`.
- function `claimERC721Airdrop()`, to claim ERC721 tokens. The claimable token cannot be the `_underlyingAsset`.
- function `claimERC1155Airdrop()`, to claim ERC1155 tokens. The claimable token cannot be the `_underlyingAsset`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

In the contract `VeBend`, the role `owner` has authority over the following function:

- function `commitSmartWalletChecker()`, to change the `smartWalletChecker` contract, which is used to check if the lock creator is from a whitelisted smart contract.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

In the contract `ReserveOracle`, the role `owner` has authority over the following functions:

- function `setAggregators()`, to set Chainlink aggregators.
- function `addAggregator()`, to add Chainlink aggregator.
- function `removeAggregator()`, to remove Chainlink aggregator.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

In the contract `WETHGateway`, the role `owner` has authority over the following function:

- function `authorizeLendPoolNFT()`, to grant approval of NFT asset to the `lendPool`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

In the contract `PunkGateway`, the role `owner` has authority over the following functions:

- function `authorizeLendPoolERC20()`, to grant maximum approval of ERC20 token to the `lendPool`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

In the contract `NFTOracle`, the role `admin` has authority over the following function:

- function `setAssetData()`, to set the NFT asset price, timestamp and round ID.

Any compromise to the `admin` account may allow a hacker to take advantage of this authority.

In the contract `LendPoolConfigurator`, the role `poolAdmin` has authority over the following functions:

- function `batchInitReserve()`, to initialize reserves in batch.
- function `batchInitNft()`, to initialize NFT in batch.
- function `updateBToken()`, to update the `bToken` implementation for the reserve.
- function `updateDebtToken()`, to update the debt token implementation for the asset.
- function `enableBorrowingOnReserve()`, to enable borrowing on a reserve.
- function `disableBorrowingOnReserve()`, to disable borrowing on a reserve.
- function `activateReserve()`, to activate a reserve.
- function `deactivateReserve()`, to deactivate a reserve.
- function `freezeReserve()`, to freeze a reserve. A frozen reserve doesn't allow any new deposit, borrow or rate swap but allows repayments, liquidations, rate rebalances and withdrawals.
- function `unfreezeReserve()`, to unfreeze a reserve.
- function `setReserveFactor()`, to update the reserve factor of a reserve.
- function `setReserveInterestRateAddress()`, to set the interest rate strategy of a reserve.
- function `configureNftAsCollateral()`, to configure the NFT collateralization parameters. All the values are expressed in percentages with two decimals of precision. A valid value is 10000, which means 100.00%.
- function `activateNft()`, to activate a NFT.
- function `deactivateNft()`, to deactivate a NFT.
- function `freezeNft()`, to freeze a NFT. A frozen NFT doesn't allow any new borrow but allows repayments, liquidations.
- function `unfreezeNft()`, to unfreeze a NFT.
- function `configureNftAsAuction()`, to configure the NFT auction parameters.
- function `setMaxNumberOfReserves()`.
- function `setMaxNumberOfNfts()`.
- function `getTokenImplementation()`, to get the implementation contract address of a proxy.

In the contract `LendPoolConfigurator`, the role `emergencyAdmin` has authority over the following function:

- function `setPoolPause()`, to pause or unpause all the actions of the protocol, including `bToken` transfers.

Any compromise to the `poolAdmin`, `emergencyAdmin` accounts may allow a hacker to take advantage of this authority.

In the contract `LendPoolAddressesProviderRegistry`, the role `owner` has authority over the following functions:

- function `registerAddressesProvider()`, to register an address provider.
- function `unregisterAddressesProvider()`.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

In the contract `LendPoolAddressesProvider`, the role `owner` has authority over the following functions:

- function `setMarketId()`, to set the `_marketId`.
- function `setAddressAsProxy()`, to set the implementation for a proxy registered with certain ID.
- function `setAddress()`, to set an address for an ID.
- function `setLendPoolImpl()`, to update the implementation of the `LendPool`, or creates the proxy setting the new `pool` implementation on the first time calling it.
- function `setLendPoolConfiguratorImpl()`, to update the implementation of the `LendPoolConfigurator`, or creates the proxy setting the new `configurator` implementation on the first time calling it.
- function `setPoolAdmin()`.
- function `setEmergencyAdmin()`.
- function `setReserveOracle()`.
- function `setNFTOracle()`.
- function `setLendPoolLoanImpl()`, to update the implementation of the `LendPoolLoan`, or creates the proxy setting the new implementation on the first time calling it.
- function `setBNFTRegistry()`, to set the `BNFTRegistry` address.
- function `setLendPoolLiquidator()`, to set the `LendPoolLiquidator` address.
- function `setIncentivesController()`.
- function `setUIDataProvider()`.
- function `setBendDataProvider()`.
- function `setWalletBalanceProvider()`.
- function `getImplementation()`, to get the implementation address of a proxy.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

In the contract `EmergencyTokenRecovery`, the role `owner` has authority over the following functions:

- function `emergencyERC20Transfer()`, to withdraw ERC20 tokens to any address.
- function `emergencyERC721Transfer()`, to withdraw ERC721 tokens to any address.
- function `emergencyPunksTransfer()`, to withdraw Punk to any address.
- function `emergencyEtherTransfer()`, to withdraw BNB to any address.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

In the contract `BToken`, the role `lendPool` has authority over the following functions:

- function `burn()`, to burn the bToken and withdraw underlying asset.
- function `mint()`, to mint the bToken.
- function `mintToTreasury()`, to mint the bToken for the `treasury` address.
- function `transferUnderlyingTo()`, to transfer the underlying asset to target address.

Any compromise to the `lendPool` account may allow a hacker to take advantage of this authority.

In the contract `DebtToken`, the role `lendPool` has authority over the following functions:

- function `burn()`, to burn the debt token.
- function `mint()`, to mint the debt token for user.

Any compromise to the `lendPool` account may allow a hacker to take advantage of this authority.

In the contract `LendPoolLoan`, the role `lendPool` has authority over the following functions:

- function `initNft()`, to grant approval of NFT asset for the `bNftAddress`.
- function `createLoan()`, to create a loan object.
- function `updateLoan()`, to update the given loan with some parameters.
- function `repayLoan()`, to repay the given loan.
- function `auctionLoan()`, to auction the given loan.
- function `liquidateLoan()`, to liquidate the given loan.

Any compromise to the `lendPool` account may allow a hacker to take advantage of this authority.

In the contract `LendPool`, the role `lendPoolConfigurator` has authority over the following functions:

- function `setPause()`, to pause/unpause the pool.
- function `setMaxNumberOfReserves()`, to set the maximum number of reserves.
- function `setMaxNumberOfNfts()`, to set the maximum number of NFTs.

- function `initReserve()`, to initialize a reserve, activating it, assigning an bToken and NFT loan and an interest rate strategy.
- function `initNft()`, to initialize a NFT, activating it, assigning NFT loan and an interest rate strategy.
- function `setReserveInterestRateAddress()`, to update the address of the interest rate strategy contract.
- function `setReserveConfiguration()`, to set the configuration bitmap of the reserve as a whole.
- function `setNftConfiguration()`, to set the configuration bitmap of the NFT as a whole.

Any compromise to the `lendPoolConfigurator` account may allow a hacker to take advantage of this authority.

In the contract `Vault`, the role `owner` has authority over the following functions:

- function `approve()`, to approve ERC20 token in the `Vault` for any account.
- function `transfer()`, to transfer ERC20 token in the `Vault` to any account.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

The team acknowledged this issue and they stated the following:

- BTokensAndBNFTsHelper is used just to simplify initial deployment and will be discarded after that.
- EmergencyTokenRecovery methods are used only to return tokens to users who transfer tokens to contracts by mistake.

In summary:

1. The team will use multi-signature wallets as owners at deployment.
2. The team will use a timelock controller at deployment.
3. The team will use multi-signature elections in their DAO.

CKP-02 | Redundant Statements

Category	Severity	Location	Status
Volatile Code	Informational	bend-incentive/contracts/vote/VeBend.sol: 103 bend-protocol/contracts/protocol/LendPoolLoan.sol: 40~43 bend-protocol/contracts/protocol/LendPool.sol: 73~76 bnft-protocol/contracts/misc/AirdropFlashLoanReceiver.sol: 29~33	Resolved

Description

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation

We advise that they are removed to better prepare the code for production environments.

Alleviation

The team heeded our advice and resolved this issue in commits:

- https://github.com/BendDAO/bend-protocol/commit/68d2c11b720bf2b83e34bd4f8d58a9590e8bd864
- https://github.com/BendDAO/bend-incentive/commit/95bdb11b6dabef2310bc40b673e5c27a4ea8062d

CKP-03 | Variables That Could Be Declared As Immutable

Category	Severity	Location	Status
Gas Optimization	Informational	bend-incentive/contracts/vote/VeBend.sol: 73, 87 bnft-protocol/contracts/misc/AirdropFlashLoanReceiver.sol: 16 bend-protocol/contracts/protocol/PunkGateway.sol: 26	Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable.

Alleviation

The team heeded our advice and resolved this issue in commits:

- https://github.com/BendDAO/bend-protocol/commit/68d2c11b720bf2b83e34bd4f8d58a9590e8bd864
- https://github.com/BendDAO/bend-incentive/commit/95bdb11b6dabef2310bc40b673e5c27a4ea8062d

CKP-04 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	bend-incentive/contracts/vote/VeBend.sol: 752~754 bnft-protocol/contracts/protocol/BNFTRegistry.sol: 126~132 bend-protocol/contracts/protocol/EmergencyTokenRecovery.sol: 65~68 bend-protocol/contracts/protocol/NFTOracle.sol: 41~43	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

The team heeded our advice and resolved this issue in commits:

- https://github.com/BendDAO/bend-protocol/commit/68d2c11b720bf2b83e34bd4f8d58a9590e8bd864
- https://github.com/BendDAO/bend-incentive/commit/95bdb11b6dabef2310bc40b673e5c27a4ea8062d
- https://github.com/BoundNFT/boundnft-protocol/commit/98c0e4958ce7e0d8c987f0fd7257c79d86e362b9

CON-01 | Incorrect Borrower Validation

Category	Severity	Location	Status
Logical Issue	Major	bend-protocol/contracts/protocol/LendPool.sol: 220~231 bend-protocol/contracts/libraries/logic/ValidationLogic.sol: 105~111	Resolved

Description

The borrower who has deposited NFT can update the loan to borrow the underlying asset. The function `LendPool.borrow()` needs to ensure the `msg.sender` is the `loanData.borrower`. However, the function calls the function `ValidationLogic.validateBorrow()` and pass the parameter `onBehalfOf` for the validation.

```
220     ValidationLogic.validateBorrow(
221       onBehalfOf,
222       asset,
223       amount,
224       reserveData,
225       nftAsset,
226       nftData,
227       vars.loanAddress,
228       vars.loanId,
229       vars.reserveOracle,
230       vars.nftOracle
231     );
```

As the result, the validation on L110 `require(user == loanData.borrower, Errors.VL_SPECIFIED_LOAN_NOT_BORROWED_BY_USER)` can not ensure that the user receives the underlying asset is the borrower, because the parameter `onBehalfOf` can be set by the caller to bypass the validation.

```
105     if (loanId != 0) {
106       DataTypes.LoanData memory loanData =
ILendPoolLoan(loanAddress).getLoan(loanId);
107
108       require(loanData.state == DataTypes.LoanState.Active,
Errors.LPL_INVALID_LOAN_STATE);
109       require(reserveAsset == loanData.reserveAsset,
Errors.VL_SPECIFIED_RESERVE_NOT_BORROWED_BY_USER);
110       require(user == loanData.borrower,
Errors.VL_SPECIFIED_LOAN_NOT_BORROWED_BY_USER);
111     }
```

Recommendation

We recommend adding logic of validation to ensure the account receives the underlying asset is the borrower via calling LendPool directly or through the gateway.

Alleviation

The team fixed this issue in the following commits.

- Added the allowance check in the function DebtToken.mint() in commit `https://github.com/BendDAO/bend-protocol/commit/7d15cfc204cf40a0cd80676f2fcc5f46fa355032`.
- Added the whitelist to the gateways in commit `https://github.com/BendDAO/bend-protocol/commit/56bd30c0dc6275a1bc6d33ac580b11072077733e`, to ensure the `msg.sender` and the `onBehalftOf` must be the same or the `msg.sender` is in the whitelist. The whitelist is maintained by the gateway owner.

CON-02 | Declaration Naming Convention

Category	Severity	Location	Status
Coding Style	Informational	bend-protocol/contracts/interfaces/ILendPool.sol: 429, 431 bend-protocol/contracts/libraries/math/WadRayMath.sol: 14, 17	Resolved

Description

The linked declarations do not conform to the Solidity style guide with regards to its naming convention.

Particularly:

- `camelCase`: Should be applied to `MAX_NUMBER_NFTS`, `MAX_NUMBER_RESERVES` function names.
- `UPPER_CASE`: Should be applied to `halfRAY`, `halfWAD` variables.

Recommendation

We recommend adjusting those variable and function names to properly conform to Solidity's naming convention.

Alleviation

The team heeded our advice and resolved this issue in commit https://github.com/BendDAO/bend-protocol/commit/68d2c11b720bf2b83e34bd4f8d58a9590e8bd864.

COR-01 | Functions With `_` As Name Prefix Are Not `private` Or `internal`

Category	Severity	Location	Status
Coding Style	Informational	bend-incentive/contracts/libs/ERC20Detailed.sol: 8~16 bend-incentive/contracts/incentives/DistributionManager.sol: 46~52	Resolved

Description

Functions with names starting with `_` should be declared as `private` or `internal`.

Recommendation

Consider changing function visibility to private or internal, or removing `_` from the start of the function name.

Alleviation

The team heeded our advice and resolved this issue in commit https://github.com/BendDAO/bend-incentive/commit/95bdb11b6dabef2310bc40b673e5c27a4ea8062d.

COT-01 | Any User Can Claim Airdrop Tokens

Category	Severity	Location	Status
Control Flow	Medium	bnft-protocol/contracts/misc/AirdropFlashLoanReceiver.sol bnft-protocol/contracts/protocol/BNFT.sol: 153~156	Resolved

Description

If the input `nftTokenIds` for the function `flashLoan()` is empty, the following loop will not start since the length is zero, hence the checks will be bypassed as well.

```
156     // only token owner can do flashloan
157     for (i = 0; i < nftTokenIds.length; i++) {
158       require(ownerOf(nftTokenIds[i]) == _msgSender(), "BNFT: caller is not owner");
159     }
```

The `AirdropFlashLoanReceiver` contract does not restrict users from obtaining airdrop tokens. Therefore, the users without any BNFT can receive airdrop tokens in the `AirdropFlashLoanReceiver` contract.

In addition, once any airdrop token is available, the user can claim the airdrop token multiple times.

Recommendation

We recommend adding appropriate restrictions to the airdrop.

Alleviation

The team heeded our advice and resolved this issue in commits:

- https://github.com/BendDAO/bend-protocol/commit/68d2c11b720bf2b83e34bd4f8d58a9590e8bd864
- https://github.com/BoundNFT/boundnft-protocol/commit/98c0e4958ce7e0d8c987f0fd7257c79d86e362b9

LPL-01 | Auction Could Fail Due To Price Fluctuations

Category	Severity	Location	Status
Control Flow	Major	bend-protocol/contracts/protocol/LendPoolLiquidator.sol: 113	Acknowledged

Description

The `auction()` function has the following restriction: the bid price cannot be smaller than the `borrowAmount`.

```
112     // bid price must greater than borrow debt
113     require(bidPrice >= vars.borrowAmount, Errors.LPL_BID_PRICE_LESS_THAN_BORROW);
```

In case the escrowed NFT price drops below the `borrowAmount`, the auction will not succeed. In the worst case, the auction never succeeds, and the escrowed NFT will be locked forever.

Recommendation

We recommend using a reasonable restriction for the `bidPrice`.

Alleviation

The team acknowledged this issue and they stated the following:

The team does not want depositors to lose their principal and must limit the auction price to be higher than the debt, so that after the NFT price returns to normal, it can still be sold.

If the NFT cannot be liquidated in auction for a long time, through the community vote, the DAO vault will contribute funds to participate in the liquidation to buy the NFT as an asset held by the vault, and after the NFT price rises above the debt, the NFT can be sold to obtain funds.

LPL-02 | Lack Of Validation In The Functions `redeem()/liquidate()`

Category	Severity	Location	Status
Logical Issue	Informational	bend-protocol/contracts/protocol/LendPoolLiquidator.sol: 203	Acknowledged

Description

In the functions `redeem()/liquidate()`, the caller is not checked.

As a result, the function `redeem()` cannot prevent being called by persons other than the borrower, and the function `liquidate()` cannot prevent being called by persons other than the bidder.

Recommendation

We recommend adding the validation for the callers of the functions `redeem()/liquidate()`.

Alleviation

The team acknowledged this issue and they stated the following:

They intentionally do not verify the caller for better architecture composition, such as WETH/Punk gateway.

NFT-01 | NFT Asset Price Not Based On Token ID

Category	Severity	Location	Status
Control Flow	Informational	bend-protocol/contracts/protocol/NFTOracle.sol: 86~87	Acknowledged

Description

The function `setAssetData()` will set the price for an NFT asset, but not based on the token ID.

```
  86      NFTPriceData memory data = NFTPriceData({price: _price, timestamp: _timestamp,
roundId: _roundId});
  87      nftPriceFeedMap[_nftContract].nftPriceData.push(data);
```

It is possible that different token IDs of an NFT asset have different prices. The "Bend" protocol does not support such NFTs.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

The team acknowledged this issue and they stated the following:

1. They only support floor price for NFT collection, not price for different NFT token.
2. They will recreate new oracle contract for different NFT token in the future.

PRT-01 | Potential Risks In The `DebtToken.mint()`

Category	Severity	Location	Status
Logical Issue	Major	bend-protocol/contracts/protocol/DebtToken.sol bend-protocol/contracts/protocol/LendPool.sol bend-protocol/contracts/protocol/PunkGateway.sol bend-protocol/contracts/protocol/WETHGateway.sol	Resolved

Description

According to the fix in the commit `https://github.com/BendDAO/bend-protocol/commit/7d15cfc204cf40a0cd80676f2fcc5f46fa355032`, the fix added the variable `allowance` in the DebtToken contract to check whether the delegator has authorized the corresponding borrowing permission to the delegatee.

```
79   function mint(
80     address initiator,
81     address onBehalfOf,
82     uint256 amount,
83     uint256 index
84   ) external override onlyLendPool returns (bool) {
85     if (initiator != onBehalfOf) {
86       _decreaseBorrowAllowance(onBehalfOf, initiator, amount);
87     }
88
89     uint256 previousBalance = super.balanceOf(onBehalfOf);
90     // index is expressed in Ray, so:
91     // amount.wadToRay().rayDiv(index).rayToWad() => amount.rayDiv(index)
92     uint256 amountScaled = amount.rayDiv(index);
93     require(amountScaled != 0, Errors.CT_INVALID_MINT_AMOUNT);
94
95     _mint(onBehalfOf, amountScaled);
96
97     emit Transfer(address(0), onBehalfOf, amount);
98     emit Mint(onBehalfOf, amount, index);
99
100    return previousBalance == 0;
101  }
```

The function `LendPool.borrow()` can be called by the delegatee directly or through the gateway.

When the delegator approves a delegatee to borrow the underlying asset, for example: `allowance[delegator][delegatee] = amount`. The delegatee can call the function `LendPool.borrow()` to

borrow the asset directly. However, the delegatee cannot call the gateway contract to borrow, because the `allowance[delegator][gateway]` is 0 and the function `LendPool.borrow()` will run into failure.

If the delegator wants to authorize borrowing permission to the delegatee and the delegate needs to borrow through the gateway, the `allowance[delegator][gateway]` should be the approved amount. However, if the gateway got the approval to borrow the asset, the hacker can attack the protocol by front-running.

Recommendation

We recommend ensuring the safety of the borrowing allowance.

Alleviation

The team heeded our advice and fixed this issue in commit `https://github.com/BendDAO/bend-protocol/commit/56bd30c0dc6275a1bc6d33ac580b11072077733e`.

The team also stated the following:

1. There's a whitelist in the gateway to ensure the `msg.sender` and the `onBehalftOf` must be the same or the `msg.sender` is in the whitelist.
2. The whitelist is maintained by the gateway owner.
3. The team will use multi-signature wallet to govern the WETHGateway.
4. Only contracts like PunkGateway which was created and deployed by Bend team will be added to the whitelist.

PRT-02 | Centralization Risks In Control Flow

Category	Severity	Location	Status
Centralization / Privilege	Major	bend-protocol/contracts/protocol/DebtToken.sol bend-protocol/contracts/protocol/LendPool.sol bend-protocol/contracts/protocol/PunkGateway.sol bend-protocol/contracts/protocol/WETHGateway.sol	Acknowledged

Description

According to the fixs in the commits `7d15cfc204cf40a0cd80676f2fcc5f46fa355032` and `56bd30c0dc6275a1bc6d33ac580b11072077733e`, the fixes added the variable `allowance` in the DebtToken contract to check whether the delegator has authorized the corresponding borrowing permission to the delegatee, and added the whitelist check whether the caller is in the whitelist or is the role `onBehalfOf`.

As per the team's response on the designs of the above-mentioned commit code:

1. The delegator can call the functions in the LendPool directly(including borrow, etc.)
2. The delegator can authorize borrowing permission to the delegatee, and the delegatee calls the functions in the LendPool directly on behalf of the delegator(including borrow, etc.)
3. The delegator authorizes borrowing permission to the gateway and calls the function in the LendPool through the gateway(including borrow, etc.)
4. The delegator authorizes borrowing permission to the WETHGateway, and calls the function in the LendPool by calling the functions of WETHGateway through the function of PunkGateway (including borrow, etc.) (The PunkGateway will be added into the whitelist in the WETHGateway to pass the whitelist check in the WETHGateway)

The below concerns are raised:

- The scope of the audit treats the calls of authorization function `approveDelegation()` during actual operations as black boxes and assumes the actual operations' correctness. These actual operations are not in the scope of the audit.
- In the contracts `PunkGateway/WETHGateway`, the role `owner` has authority over the function `authorizeCallerWhitelist()` to add addresses to the whitelist. For the above scenarios 3 and 4, the addresses added into the whitelist have the authority to attack the protocol by front-running.

Recommendation

We understand that the business logic of the Bend protocol requires the authorization function `approveDelegation()`. We encourage the team to continuously monitor user authorization actions and provide the correct and effective authorization invocation path, such as on-chain smart contracts and web3 applications to mitigate the side effects when unexpected activities are observed.

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Noted: Recommend considering the long-term solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

The team acknowledged this issue and they stated the following:

1. The team will use a multi-signature wallet as the owner at deployment.
2. The team will use a timelock controller at deployment.
3. The team will use multi-signature elections in their DAO.

VBC-01 | Usage Of Require Instead Of Assert

Category	Severity	Location	Status
Language Specific	Informational	bend-incentive/contracts/vote/VeBend.sol: 355, 427, 486, 600, 719	Resolved

Description

The `assert` function should only be used to test for internal errors, and to check invariants. The `require` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require()` function, along with a custom error message when the condition fails, instead of the `assert()` function.

Alleviation

The team heeded our advice and resolved this issue in commit https://github.com/BendDAO/bend-incentive/commit/95bdb11b6dabef2310bc40b673e5c27a4ea8062d.

VBC-02 | Missing Error Messages

Category	Severity	Location	Status
Coding Style	Informational	bend-incentive/contracts/vote/VeBend.sol: 103, 355, 427, 486, 600, 719	Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

The team heeded our advice and resolved this issue in commit https://github.com/BendDAO/bend-incentive/commit/95bdb11b6dabef2310bc40b673e5c27a4ea8062d.

WBP-01 | Locked Ether

Category	Severity	Location	Status
Language Specific	Medium	bend-protocol/contracts/misc/WalletBalanceProvider.sol: 34~37	Resolved

Description

The contract has the `receive()` payable function, but does not have a function to withdraw the fund.

```
34    receive() external payable {
35      //only contracts can send ETH to the core
36      require(msg.sender.isContract(), "22");
37    }
```

Recommendation

We recommend removing the `receive()` function or adding a withdrawal function.

Alleviation

The team heeded our advice and resolved this issue in commit `95bdb11b6dabef2310bc40b673e5c27a4ea8062d`.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.