BitLen Smart Contract

May 2024

SMART CONTRACT AUDIT REPORT

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by BitLen to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

Medium risks are mainly risks caused by centralized roles and price updates.

Low risk mainly refers to the potential risks that may exist when the contract logic is executed.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- Likelihood: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- Impact: measures the technical loss and business damage of a successful attack.
- Severity: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into for: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly, Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood \ IMPACT	Informational	Low	Medium	High
High	Informational	Medium	High	Critical
Medium	Informational	Low	Medium	High
Low	Informational	Low	Low	Medium

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive

assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Code and business security testing: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft Fail Attack
	Hard Fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control
	System API Call Analysis

Category	Assessment Item
	Contract Deployment Consistency Check
Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

2.1 Project Info And Contract Address

Project Name: BitLen

Audit Time: April 17nd, 2024 – May 10th, 2024

Language: Solidity

File Name	Link
BitLen	https://github.com/Bitlen-Protocol/bitlen-contracts/commit/c0899f93054dc90510a62912bd9517f40647f85c

2.2 Summary

Severity	Found	
Critical	0	
High	0	
Medium	2	
Low	4	
Informational	5	

2.3 Key Findings

Medium risks are mainly risks caused by centralized roles and price updates.

Low risk mainly refers to the potential risks that may exist when the contract logic is executed.

ID	Severity	Findings Title	Status	Confirm
NVE-001	Medium	Centralized role	Ignored	Confirmed
NVE-002	Medium	Price acquisition	Ignored	Confirmed
NVE-003	Low	There are no handling fees for flash loans	Ignored	Confirmed
NVE-004	Low	mintTo() and burnTo() may be called by any user to obtain contract funds	Ignored	Confirmed
NVE-005	Low	Interest still accrues during suspension	Ignored	Confirmed
NVE-006	Low	withdraw helper does not determine the whitelist	Fixed	Confirmed
NVE-007	Informational	Possible bad debts	Ignored	Confirmed
NVE-008	Informational	Unused variable	Fixed	Confirmed
NVE-009	Informational	There are no event records for mintto and burnto	Ignored	Confirmed
NVE-0010	Informational	There is no determination of whether the pool is non-repetitive	Ignored	Confirmed
NVE-011	Informational	Fewer participants will lead to higher platform liquidity and lending risks	Ignored	Confirmed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Centralized role

ID:	NVE-001	**Location:**	Config.sol,InitCore.sol, LiqIncentiveCalculator.sol, PosManager.sol,LendingPool.sol, RiskManager.sol,BitLenB2Oracle.sol, BitLenOracle.sol,BitLenPythOracle.sol
Severity:	Medium	**Category:**	Privileged role
Likelihood:	Low	**Impact:**	High

- Config.sol

Description:

There are two centralized permissions in the Config, InitCore, and LendingPool contracts, namely the governor and guardian roles modified by onlyGovernor and onlyGuardian.

The two roles in the Config contract can be used to set pool configuration related factors, set the maximum health value after liquidation and the status of the mode.

The two roles in the InitCore contract can set config, oracle and other important variables.

The two roles in the LendingPool contract can set the vault address, interest rate model, and reserve coefficient; they can also adjust the reserve coefficient and interest rate model.

There is a centralized authority in the LiqIncentiveCalculator, BitLenB2Oracle, BitLenOracle, and BitLenPythOracle contracts, which is the governor role modified by onlyGovernor.

The governor role in the LiqIncentiveCalculator contract can set incentive multipliers in different modes and tokens. It is also possible to set the maximum and minimum multipliers for liquidation incentives.

BitLenB2Oracle, BitLenOracle, BitLenPythOracle contract governor can directly set the price of a specific token

Set up the data source and SupraOracle instance.

There is a centralized authority in the PosManager and RiskManager contracts, which is the guardian role modified by onlyGuardian.

The guardian role in the PosManager contract can set the maximum amount of collateral.

The guardian role in the RiskManager contract can set the debt limit.

Many key values in the contract have centralized role settings. If the centralized authority is stolen or lost, it will have an impact on the project.

```
88      /// @inheritdoc IConfig
89      function setPoolConfig(address _pool, PoolConfig calldata _config) external onlyGuardian {
90          __poolConfigs[_pool] = _config;
91          emit SetPoolConfig(_pool, _config);
92      }
93
94      /// @inheritdoc IConfig
95      function setCollFactors_e18(uint16 _mode, address[] calldata _pools, uint128[] calldata _factors_e18)
96          external
97          onlyGovernor
98      {
99          _require(_mode != 0, Errors.INVALID_MODE);
100         _require(_pools.length == _factors_e18.length, Errors.ARRAY_LENGTH_MISMATCHED);
101         EnumerableSet.AddressSet storage collTokens = __modeConfigs[_mode].collTokens;
102         for (uint256 i; i < _pools.length; i = i.uinc()) {
103             _require(_factors_e18[i] <= ONE_E18, Errors.INVALID_FACTOR);
104             collTokens.add(_pools[i]);
105             __modeConfigs[_mode].factors[_pools[i]].collFactor_e18 = _factors_e18[i];
106         }
107         emit SetCollFactors_e18(_mode, _pools, _factors_e18);
108     }
109
110     /// @inheritdoc IConfig
111     function setBorrFactors_e18(uint16 _mode, address[] calldata _pools, uint128[] calldata _factors_e18)
112         external
113         onlyGovernor
114     {
115         _require(_mode != 0, Errors.INVALID_MODE);
116         _require(_pools.length == _factors_e18.length, Errors.ARRAY_LENGTH_MISMATCHED);
117         EnumerableSet.AddressSet storage borrTokens = __modeConfigs[_mode].borrTokens;
118         for (uint256 i; i < _pools.length; i = i.uinc()) {
119             borrTokens.add(_pools[i]);
120             _require(_factors_e18[i] >= ONE_E18, Errors.INVALID_FACTOR);
121             __modeConfigs[_mode].factors[_pools[i]].borrFactor_e18 = _factors_e18[i];
122         }
123         emit SetBorrFactors_e18(_mode, _pools, _factors_e18);
124     }
125
126     /// @inheritdoc IConfig
127     function setModeStatus(uint16 _mode, ModeStatus calldata _status) external onlyGuardian {
128         _require(_mode != 0, Errors.INVALID_MODE);
129         __modeConfigs[_mode].status = _status;
130         emit SetModeStatus(_mode, _status);
131     }
132
133     /// @inheritdoc IConfig
134     function setMaxHealthAfterLiq_e18(uint16 _mode, uint64 _maxHealthAfterLiq_e18) external onlyGuardian {
135         _require(_mode != 0, Errors.INVALID_MODE);
136         _require(_maxHealthAfterLiq_e18 > ONE_E18, Errors.INPUT_TOO_LOW);
137         __modeConfigs[_mode].maxHealthAfterLiq_e18 = _maxHealthAfterLiq_e18;
138         emit SetMaxHealthAfterLiq_e18(_mode, _maxHealthAfterLiq_e18);
139     }
140 }
```

Figure 3.1.1 set function

Recommendations:

Exvul Web3 Security recommends centralized roles are managed using multi-signatures.

Result: Confirmed

Fix Result:

BitLen confirms that it currently operates with independent addresses.

3.2 Price acquisition

ID:	NVE-002	Location:	BitLenB2Oracle.sol, BitLenOracle.sol,BitLenPythOracle.sol
Severity:	Medium	Category:	Business Issues
Likelihood:	Low	Impact:	High

- BitLenB2Oracle.sol

Description:

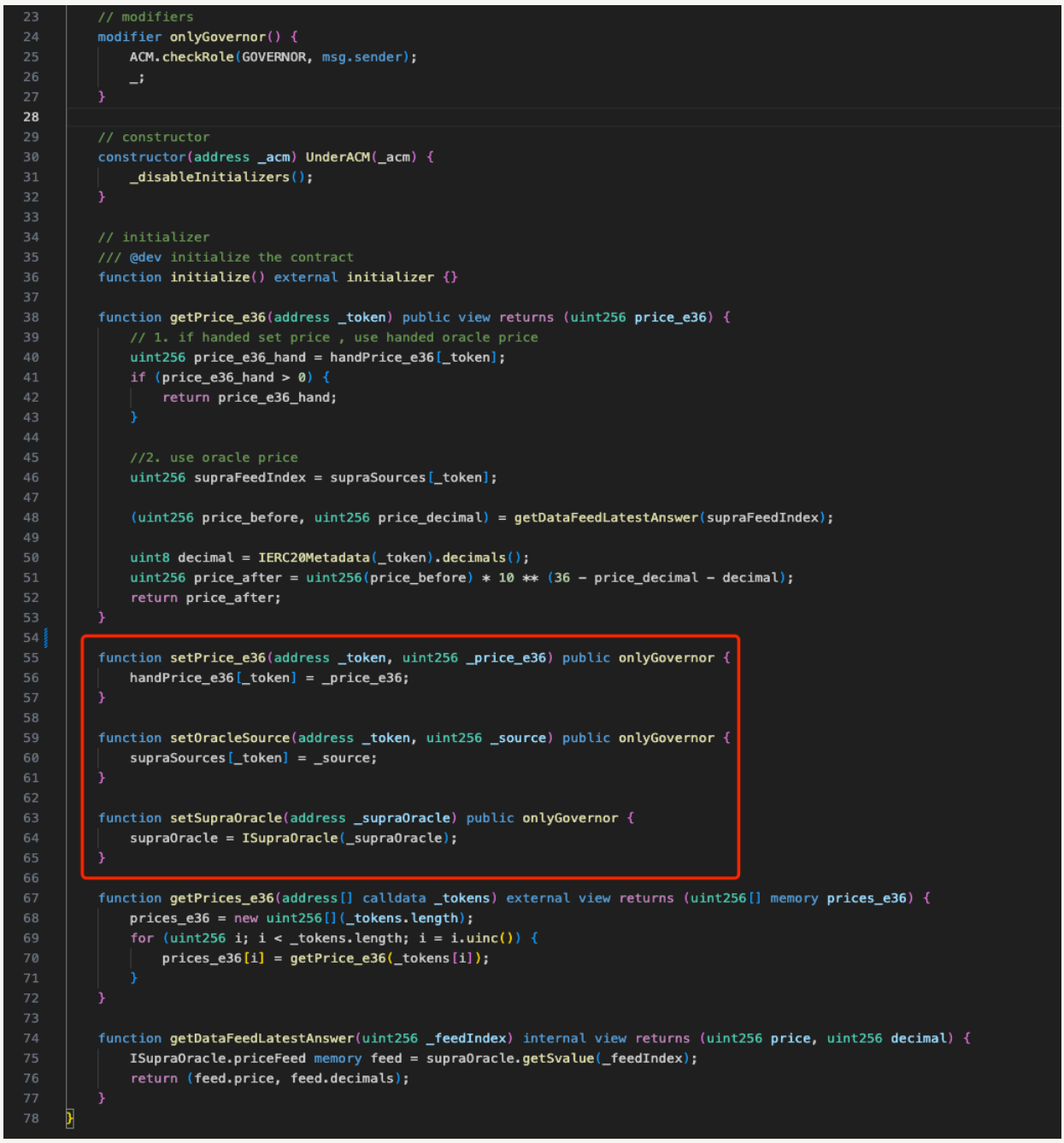
BitLenB2Oracle, BitLenOracle, and BitLenPythOracle contracts are all used to obtain prices. The Governor role in the contract can directly set the price of a specific token, set the data source, and SupraOracle instance.

Price acquisition: There are currently two ways to obtain prices. The first is to set a fixed price for a certain Token by a privileged role, and the second is to obtain the price through the set price source.

There are two potential risks here

First: If the fixed price of a certain Token is significantly different from the price of other platforms, or the price of the Token itself fluctuates greatly, then there will be a risk of fund theft caused by the large price difference when the price is used in the contract.

Second: If there is a problem with the set price source, resulting in an incorrect price being reported, there may also be a risk of fund theft.



```
23    // modifiers
24    modifier onlyGovernor() {
25        ACM.checkRole(GOVERNOR, msg.sender);
26        _;
27    }
28
29    // constructor
30    constructor(address _acm) UnderACM(_acm) {
31        _disableInitializers();
32    }
33
34    // initializer
35    /// @dev initialize the contract
36    function initialize() external initializer {}
37
38    function getPrice_e36(address _token) public view returns (uint256 price_e36) {
39        // 1. if handed set price , use handed oracle price
40        uint256 price_e36_hand = handPrice_e36[_token];
41        if (price_e36_hand > 0) {
42            return price_e36_hand;
43        }
44
45        //2. use oracle price
46        uint256 supraFeedIndex = supraSources[_token];
47
48        (uint256 price_before, uint256 price_decimal) = getDataFeedLatestAnswer(supraFeedIndex);
49
50        uint8 decimal = IERC20Metadata(_token).decimals();
51        uint256 price_after = uint256(price_before) * 10 ** (36 - price_decimal - decimal);
52        return price_after;
53    }
54
55    function setPrice_e36(address _token, uint256 _price_e36) public onlyGovernor {
56        handPrice_e36[_token] = _price_e36;
57    }
58
59    function setOracleSource(address _token, uint256 _source) public onlyGovernor {
60        supraSources[_token] = _source;
61    }
62
63    function setSupraOracle(address _supraOracle) public onlyGovernor {
64        supraOracle = ISupraOracle(_supraOracle);
65    }
66
67    function getPrices_e36(address[] calldata _tokens) external view returns (uint256[] memory prices_e36) {
68        prices_e36 = new uint256[](_tokens.length);
69        for (uint256 i; i < _tokens.length; i = i.uinc()) {
70            prices_e36[i] = getPrice_e36(_tokens[i]);
71        }
72    }
73
74    function getDataFeedLatestAnswer(uint256 _feedIndex) internal view returns (uint256 price, uint256 decimal) {
75        ISupraOracle.priceFeed memory feed = supraOracle.getSvalue(_feedIndex);
76        return (feed.price, feed.decimals);
77    }
78}
```

Figure 3.2.1 setPrice function

Recommendations:

Exvul Web3 Security recommends that centralized roles are managed using multi-signatures; caution is required when setting a specific Token price, and the correct price needs to be updated in a timely manner when there is a large deviation in the price source.

Result: Confirmed

Fix Result:
BitLen confirmed to use a third-party oracle first.

3.3 There are no handling fees for flash loans

ID:	NVE-003	Location:	InitCore.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

- InitCore.sol

Description:

The flash method allows users to perform flash loan operations, but this method does not include any clear logic for charging fees. No handling fee will be charged. Users can lend large amounts of assets in the capital pool for market manipulation and arbitrage.

```
273     /// @inheritdoc IInitCore
274     function flash(address[] calldata _pools, uint256[] calldata _amts, bytes calldata _data)
275         public
276         virtual
277         nonReentrant
278     {
279         // validate _pools and _amts length & validate _pools contain distinct addresses to avoid paying less flash fees
280         _require(_validateFlash(_pools, _amts), Errors.INVALID_FLASHLOAN);
281         // check that is not multicall tx
282         _require(!isMulticallTx, Errors.LOCKED_MULTICALL);
283         uint256[] memory balanceBefores = new uint256[](_pools.length);
284         address[] memory tokens = new address[](_pools.length);
285         IConfig _config = IConfig(config);
286         for (uint256 i; i < _pools.length; i = i.uinc()) {
287             PoolConfig memory poolConfig = _config.getPoolConfig(_pools[i]);
288             // check that flash is enabled
289             _require(poolConfig.canFlash, Errors.FLASH_PAUSED);
290             address token = ILendingPool(_pools[i]).underlyingToken();
291             tokens[i] = token;
292             // calculate return amt
293             balanceBefores[i] = IERC20(token).balanceOf(_pools[i]);
294             // take _amts[i] of _pools[i] to msg.sender
295             IERC20(token).safeTransferFrom(_pools[i], msg.sender, _amts[i]);
296         }
297         // execute callback
298         IFlashReceiver(msg.sender).flashCallback(_pools, _amts, _data);
299         // check pool balance after callback
300         for (uint256 i; i < _pools.length; i = i.uinc()) {
301             _require(IERC20(tokens[i]).balanceOf(_pools[i]) >= balanceBefores[i], Errors.INVALID_AMOUNT_TO_REPAY);
302         }
303     }
```

Figure 3.3.1 flash function

Recommendations:

Exvul Web3 Security recommends charging a small fee for flash loans to avoid abuse of the flash loan function.

Result: Confirmed

Fix Result:
BitLen confirmed that the flash loan function will be false initially, and will consider setting the flash loan fee later.

3.4 mintTo() and burnTo() may be called by any user to obtain contract funds

ID:	NVE-004	Location:	InitCore.sol,LendingPool.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

- InitCore.sol

Description:

The mintTo and burnTo methods are used to perform mint and burn operations respectively, but these two methods are called separately and can be called from all addresses. If a user transfers the token, a malicious attacker can directly call this method to obtain contract funds.

```
98      // functions
99      /// @inheritdoc IInitCore
100     function mintTo(address _pool, address _to) public virtual nonReentrant returns (uint256 shares) {
101         // check pool status
102         PoolConfig memory poolConfig = IConfig(config).getPoolConfig(_pool);
103         _require(poolConfig.canMint, Errors.MINT_PAUSED);
104         // call mint at pool using _to
105         shares = ILendingPool(_pool).mint(_to);
106         // check supply cap after mint
107         _require(ILendingPool(_pool).totalAssets() <= poolConfig.supplyCap, Errors.SUPPLY_CAP_REACHED);
108     }
109
110     /// @inheritdoc IInitCore
111     function burnTo(address _pool, address _to) public virtual nonReentrant returns (uint256 amt) {
112         // check pool status
113         PoolConfig memory poolConfig = IConfig(config).getPoolConfig(_pool);
114         _require(poolConfig.canBurn, Errors.REDEEM_PAUSED);
115         // call burn at pool using _to
116         amt = ILendingPool(_pool).burn(_to);
117     }
```

Figure 3.4.1 mintTo and burnTo function

```
100     /// @inheritdoc ILendingPool
101     function mint(address _receiver) external onlyCore accrue returns (uint256 shares) {
102         uint256 _cash = cash;
103         uint256 newCash = IERC20(underlyingToken).balanceOf(address(this));
104
105         uint256 amt = newCash - _cash;
106         // amount =  10e18
107         shares = _toShares(amt, _cash + totalDebt, totalSupply());
108         _require(shares != 0, Errors.ZERO_VALUE);
109         _mint(_receiver, shares);
110         cash = newCash;
111     }
112
113     function _toShares(uint256 _amt, uint256 _totalAssets, uint256 _totalShares)
114         internal
115         pure
116         returns (uint256 shares)
117     {
118         return _amt.mulDiv(_totalShares + VIRTUAL_SHARES, _totalAssets + VIRTUAL_ASSETS);
119     }
120
121     /// @inheritdoc ILendingPool
122     function toShares(uint256 _amt) public view returns (uint256 shares) {
123         shares = _toShares(_amt, totalAssets(), totalSupply());
124     }
125
126     /// @inheritdoc ILendingPool
127     function burn(address _receiver) external onlyCore accrue returns (uint256 amt) {
128         uint256 sharesToBurn = balanceOf(address(this));
129         uint256 _cash = cash;
130         _require(sharesToBurn != 0, Errors.ZERO_VALUE);
131         amt = _toAmt(sharesToBurn, _cash + totalDebt, totalSupply());
132         _require(amt <= _cash, Errors.NOT_ENOUGH_CASH);
133         unchecked {
134             cash = _cash - amt;
135         }
136         _burn(address(this), sharesToBurn);
137         IERC20(underlyingToken).safeTransfer(_receiver, amt);
138     }
```

Figure 3.4.2 mint and burn function

Recommendations:

Exvul Web3 Security recommends writing the mintTo and burnTo methods directly into the operation logic for calls to avoid damage to contract funds caused by calls from other addresses.

Result: Confirmed

Fix Result:
BitLen confirmed that the mintTo() and burnTo() methods will be expanded to include other hook calls in the future, so these two methods cannot be completely restricted.

3.5 Interest still accrues during suspension

ID:	NVE-005	Location:	InitCore.sol,LendingPool.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

- InitCore.sol

Description:

The _repay method is used to repay the loan. The administrator can suspend repayment, but interest will still accumulate after the suspension.

In addition to the interest rate model, interest accumulation is mainly calculated by time interval. In the absence of repayment, the accumulated interest will cause the total debt to continue to increase, which may lead to a decline in system health.

```
430     /// @dev repay borrowed tokens
431     /// @param _config config
432     /// @param _mode position mode
433     /// @param _posId position id
434     /// @param _pool pool address to repay
435     /// @param _shares amount of shares to repay
436     /// @return tokenToRepay token address to repay
437     ///         amt          amt of token to repay
438     function _repay(IConfig _config, uint16 _mode, uint256 _posId, address _pool, uint256 _shares)
439         internal
440         returns (address tokenToRepay, uint256 amt)
441     {
442         // check status
443         _require(_config.getPoolConfig(_pool).canRepay && _config.getModeStatus(_mode).canRepay, Errors.REPAY_PAUSED);
444         // get position debt share
445         uint256 positionDebtShares = IPosManager(POS_MANAGER).getPosDebtShares(_posId, _pool);
446         uint256 sharesToRepay = _shares < positionDebtShares ? _shares : positionDebtShares;
447         // get amtToRepay (accrue interest)
448         uint256 amtToRepay = ILendingPool(_pool).debtShareToAmtCurrent(sharesToRepay);
449         // take token from msg.sender to pool
450         tokenToRepay = ILendingPool(_pool).underlyingToken();
451         IERC20(tokenToRepay).safeTransferFrom(msg.sender, _pool, amtToRepay);
452         // update debt on the position
453         IPosManager(POS_MANAGER).updatePosDebtShares(_posId, _pool, -sharesToRepay.toInt256());
454         // call repay on the pool
455         amt = ILendingPool(_pool).repay(sharesToRepay);
456         // update debt on mode
457         IRiskManager(riskManager).updateModeDebtShares(_mode, _pool, -sharesToRepay.toInt256());
458         emit Repay(_pool, _posId, msg.sender, _shares, amt);
459     }
```

Figure 3.5.1 _repay function

Recommendations:

Exvul Web3 Security recommends that after suspending repayment, the accumulation of interest should also be suspended.

Result: Confirmed

Fix Result:
BitLen confirmed that the current logic is as designed and that suspensions are rare occurrences.

3.6 withdraw helper does not determine the whitelist

ID:	NVE-006	Location:	MoneyMarketHook.sol
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

- MoneyMarketHook.sol

Description:

The main function of the _handleWithdraw method is to withdraw funds from the loan agreement. However, this method does not whitelist the helper address. If the helper is an arbitrary address, when a malicious helper contract is added to the transaction process, the attacker may exploit unknown contract logic or vulnerabilities to perform unexpected operations and profit from it.

```
172     /// @dev generate withdraw data for multicall
173     /// @param _offset offset of data
174     /// @param _data multicall data
175     /// @param _initPosId init position id (nft id)
176     /// @param _params withdraw params
177     /// @return offset new offset
178     /// @return data new data
179     function _handleWithdraw(
180         uint256 _offset,
181         bytes[] memory _data,
182         uint256 _initPosId,
183         WithdrawParams[] calldata _params,
184         bool _returnNative
185     ) internal view returns (uint256, bytes[] memory) {
186         for (uint256 i; i < _params.length; i = i.uinc()) {
187             // decollateralize to pool
188             _data[_offset] = abi.encodeWithSelector(
189                 IInitCore.decollateralize.selector, _initPosId, _params[i].pool, _params[i].shares, _params[i].pool
190             );
191             _offset = _offset.uinc();
192             // burn collateral to underlying token
193             address helper = _params[i].rebaseHelperParams.helper;
194             address uToken = ILendingPool(_params[i].pool).underlyingToken();
195             address uTokenReceiver = _params[i].to;
196             if (uToken == WNATIVE && _returnNative) uTokenReceiver = address(this);
197             // if need to unwrap to rebase token
198             if (helper != address(0)) {
199                 // check if the helper is whitelisted
200                 _require(
201                     _params[i].rebaseHelperParams.tokenIn == uToken
202                         && IRebaseHelper(helper).YIELD_BEARING_TOKEN() == uToken,
203                     Errors.INVALID_TOKEN_IN
204                 );
205                 uTokenReceiver = helper;
206             }
207             _data[_offset] = abi.encodeWithSelector(IInitCore.burnTo.selector, _params[i].pool, uTokenReceiver);
208             _offset = _offset.uinc();
209         }
210         return (_offset, _data);
211     }
```

Figure 3.6.1 _handleWithdraw function

Recommendations:

Exvul Web3 Security recommends whitelisting helper addresses to ensure that only verified and approved helper addresses can be used.

Result: Confirmed

Fix Result:
Fixed.

BitLen confirmed that it has added a restriction for helper to whitelist addresses in the _handleWithdraw() method.

Fixed version: 0d75b55ae7b984527ad0f0be8416cb63e1b7dc98

3.7 Possible bad debts

ID:	NVE-007	**Location:**	InitCore.sol,LendingPool.sol
Severity:	Informational	**Category:**	Business Issues
Likelihood:	Informational	**Impact:**	Informational

Description:

If market conditions are extreme or the price of mortgage assets fluctuates sharply, there may be a situation where liquidation cannot be carried out in a timely manner or the proceeds from liquidation are insufficient to cover the loan, resulting in bad debts.

Recommendations:

Exvul Web3 Security recommends only pledging high-quality assets to avoid drastic price fluctuations. It also recommends adding liquidation monitoring to avoid a large number of bad debts.

Result: Confirmed

Fix Result:
Ignored.

3.8 Unused variable

ID:	NVE-008	Location:	PosManager.sol
Severity:	Informational	Category:	Business Issues
Likelihood:	Informational	Impact:	Informational

- PosManager.sol

Description:

The isCollateralized mapping is defined in the PosManager contract but is not used.

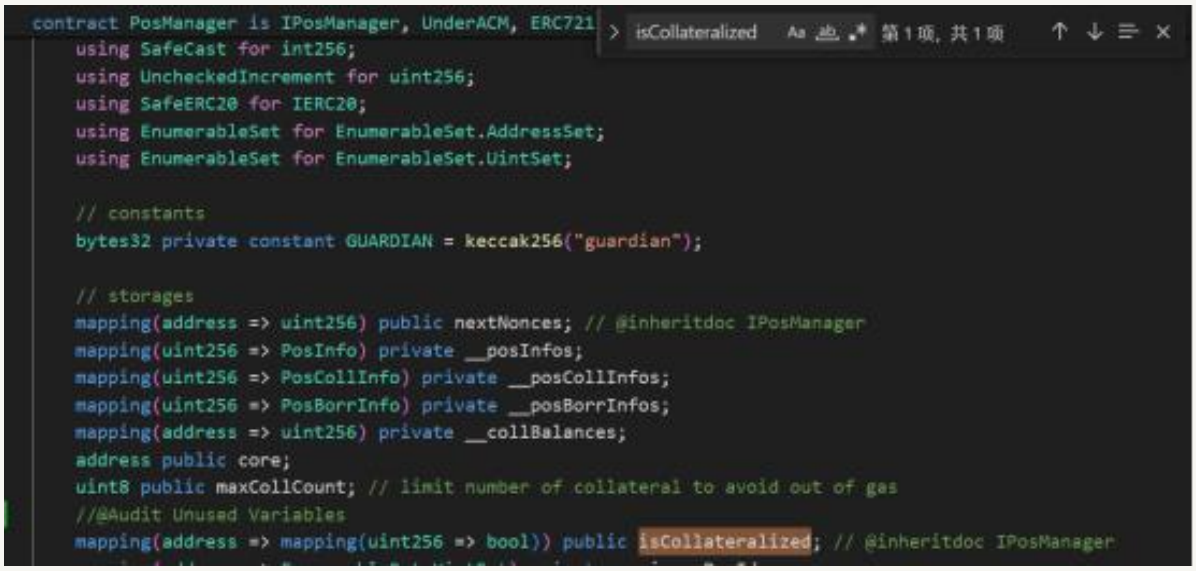


Figure 3.8.1 PosManager contract

Recommendations:

Exvul Web3 Security recommends Remove redundant code.

Result: Confirmed

Fix Result:
Fixed.

BitLen confirmed that the redundant code has been removed.

Fixed version: 0d75b55ae7b984527ad0f0be8416cb63e1b7dc98

3.9 There are no event records for mintto and burnto

ID:	NVE-009	Location:	InitCore.sol
Severity:	Informational	Category:	Business Issues
Likelihood:	Informational	Impact:	Informational

- InitCore.sol

Description:

The mintTo() and burnTo() methods can be called through the MoneyMarketHook.execute() method, or by the user independently. However, the methods do not record events, and it is not possible to track the operation and status changes of the contract through events.

```
98      // functions
99      /// @inheritdoc IInitCore
100     function mintTo(address _pool, address _to) public virtual nonReentrant returns (uint256 shares) {
101         // check pool status
102         PoolConfig memory poolConfig = IConfig(config).getPoolConfig(_pool);
103         _require(poolConfig.canMint, Errors.MINT_PAUSED);
104         // call mint at pool using _to
105         shares = ILendingPool(_pool).mint(_to);
106         // check supply cap after mint
107         _require(ILendingPool(_pool).totalAssets() <= poolConfig.supplyCap, Errors.SUPPLY_CAP_REACHED);
108     }
109
110     /// @inheritdoc IInitCore
111     function burnTo(address _pool, address _to) public virtual nonReentrant returns (uint256 amt) {
112         // check pool status
113         PoolConfig memory poolConfig = IConfig(config).getPoolConfig(_pool);
114         _require(poolConfig.canBurn, Errors.REDEEM_PAUSED);
115         // call burn at pool using _to
116         amt = ILendingPool(_pool).burn(_to);
117     }
```

Figure 3.9.1 mintTo and burnTo function

Recommendations:

Exvul Web3 Security recommends adding event logging.

Result: Confirmed

Fix Result:
Ignored.

3.10 There is no determination of whether the pool is non-repetitive

ID:	NVE-010	Location:	config.sol
Severity:	Informational	Category:	Business Issues
Likelihood:	Informational	Impact:	Informational

- config.sol

Description:

In the setBorrFactors_e18() and setCollFactors_e18() methods of the Config contract, there is no check for duplicate addresses in the passed _pools array, which may introduce duplicate data writes, causing the contract to update collFactor_e18 or borrFactor_e18 multiple times for the same address. Since the last write will overwrite the previous write, this may cause unnecessary calculations and state changes, wasting gas fees.

```
94      /// @inheritdoc IConfig
95      function setCollFactors_e18(uint16 _mode, address[] calldata _pools, uint128[] calldata _factors_e18)
96          external
97          onlyGovernor
98      {
99          _require(_mode != 0, Errors.INVALID_MODE);
100         _require(_pools.length == _factors_e18.length, Errors.ARRAY_LENGTH_MISMATCHED);
101         EnumerableSet.AddressSet storage collTokens = __modeConfigs[_mode].collTokens;
102         for (uint256 i; i < _pools.length; i = i.uinc()) {
103             _require(_factors_e18[i] <= ONE_E18, Errors.INVALID_FACTOR);
104             collTokens.add(_pools[i]);
105             __modeConfigs[_mode].factors[_pools[i]].collFactor_e18 = _factors_e18[i];
106         }
107         emit SetCollFactors_e18(_mode, _pools, _factors_e18);
108     }
109
110     /// @inheritdoc IConfig
111     function setBorrFactors_e18(uint16 _mode, address[] calldata _pools, uint128[] calldata _factors_e18)
112         external
113         onlyGovernor
114     {
115         _require(_mode != 0, Errors.INVALID_MODE);
116         _require(_pools.length == _factors_e18.length, Errors.ARRAY_LENGTH_MISMATCHED);
117         EnumerableSet.AddressSet storage borrTokens = __modeConfigs[_mode].borrTokens;
118         for (uint256 i; i < _pools.length; i = i.uinc()) {
119             borrTokens.add(_pools[i]);
120             _require(_factors_e18[i] >= ONE_E18, Errors.INVALID_FACTOR);
121             __modeConfigs[_mode].factors[_pools[i]].borrFactor_e18 = _factors_e18[i];
122         }
123         emit SetBorrFactors_e18(_mode, _pools, _factors_e18);
124     }
```

Figure 3.10.1 setCollFactors_e18 and setBorrFactors_e18 function

Recommendations:

Exvul Web3 Security recommends that in the setBorrFactors_e18() and setCollFactors_e18() methods, the addresses passed into the _pools array should not be repeated.

Result: Confirmed

Fix Result:
Ignored.

3.11 Fewer participants will lead to higher platform liquidity and lending risks

ID:	NVE-011	Location:	InitCore.sol
Severity:	Informational	Category:	Business Issues
Likelihood:	Informational	Impact:	Informational

- MoneyMarketHook.sol

Description:

The platform is a lending project. If the platform has a small number of users, it may lead to liquidity problems and higher lending risks. If the borrower group is small, then liquidation, bad debts and other factors may have a greater impact on the entire platform.

Result: Confirmed

Fix Result:
Ignored.

4. CONCLUSION

In this audit, we thoroughly analyzed **BitLen** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be **PASSED**. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

- Description: The security of apply verification
- Result: Not found
- Severity: Critical

5.1.2 Authorization Access Control

- Description: Permission checks for external integral functions
- Result: Not found
- Severity: Critical

5.1.3 Forged Transfer Vulnerability

- Description: Assess whether there is a forged transfer notification vulnerability in the contract
- Result: Not found
- Severity: Critical

5.1.4 Transaction Rollback Attack

- Description: Assess whether there is transaction rollback attack vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.5 Transaction Block Stuffing Attack

- Description: Assess whether there is transaction blocking attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.6 Soft Fail Attack Assessment

- Description: Assess whether there is soft fail attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.7 Hard Fail Attack Assessment

- Description: Examine for hard fail attack vulnerability
- Result: Not found
- Severity: Critical

5.1.8 Abnormal Memo Assessment

- Description: Assess whether there is abnormal memo vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.9 Abnormal Resource Consumption

- Description: Examine whether abnormal resource consumption in contract processing.
- Result: Not found
- Severity: Critical

5.1.10 Random Number Security

- Description: Examine whether the code uses insecure random number.
- Result: Not found
- Severity: Critical

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

- Description: Examine for weakness in cryptograph implementation.
- Results: Not Found
- Severity: High

5.2.2 Account Permission Control

- Description: Examine permission control issue in the contract
- Results: Not Found
- Severity: Medium

5.2.3 Malicious Code Behavior

- Description: Examine whether sensitive behavior present in the code
- Results: Not found
- Severity: Medium

5.2.4 Sensitive Information Disclosure

- Description: Examine whether sensitive information disclosure issue present in the code.
- Result: Not found
- Severity: Medium

5.2.5 System API

- Description: Examine whether system API application issue present in the code
- Results: Not found
- Severity: Low

6. 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 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 without ExVul's prior written consent.

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 ExVul 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. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul'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.

7. REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound).

https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error.

https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption.

https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation.

https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure.

https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features.

https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems.

https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors.

https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors.

https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology.

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

www.exvul.com

contact@exvul.com

@EXVULSEC

github.com/EXVUL-Sec

ExVul