BLOCKSEC

Security Audit
Report for Perp Contracts

Date: July 15, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	HertzFlow
Target	Perp Contracts

Version History

Version	Date	Description
1.0	July 15, 2026	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-08-13 13:28:03+0000

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of Perp Contracts of HertzFlow.

The Perp Contracts of HertzFlow is a perpetual exchange, which is forked from GMX Synthetics v2.3-branch. This version introduces a credit token program where the project-issued credit token can be used in two ways. First, the credit tokens can be spent to offset trading fees. Second, the credit tokens can serve as collateral to open positions via the corresponding market, where positive PnL is settled in USDT through a profit claim vault. In addition, a Loss Rebate mechanism is introduced to incentivize weak-side participation. When a position is opened on the weaker side of an imbalanced market, a per-position USD rebate accrues and later offsets realized losses during position decreases. Furthermore, the referral system is rewritten from single-level to a three-level (L0/L1/L2) model where traders receive fee discounts and affiliates earn multi-tier rewards with a pro-rata cap ensuring total rebates never exceed the position fee amount. Moreover, the project also introduces a new module named HFBank. Specifically, it is an asset wrapping layer, which supports real asset custody, wrapped token minting and redemption, flash minting, authorized wrapping, and reserve borrow and repay flows. It also updates liquidation order construction and relay signing so that delegated order creation is bound to the selected position type.

Note this audit only focuses on the smart contracts in the following directories/files. Code prior to and including the baseline version 0, where applicable, is outside the scope of this audit and assumes to be reliable and secure.

- contracts/config/Config.sol
- contracts/config/ConfigUtils.sol
- contracts/credit/*
- contracts/data/Keys.sol
- contracts/data/Keys2.sol
- contracts/fee/FeeHandler.sol
- contracts/order/DecreaseOrderUtils.sol
- contracts/order/OrderUtils.sol
- contracts/position/DecreaseCreditPositionCollateralUtils.sol
- contracts/position/DecreaseCreditPositionUtils.sol
- contracts/position/DecreasePositionCollateralUtils.sol
- contracts/position/DecreasePositionUtils.sol

[1] https://github.com/HertzFlow/contracts-v2-bsc-audit

- contracts/position/DecreaseZFPPositionCollateralUtils.sol
- contracts/position/DecreaseZFPPositionUtils.sol
- contracts/position/IncreasePositionUtils.sol
- contracts/position/Position.sol
- contracts/position/PositionEventUtils.sol
- contracts/position/PositionStoreUtils.sol
- contracts/position/PositionUtils.sol
- contracts/reader/*
- contracts/referral/*
- contracts/role/Role.sol
- contracts/role/RoleModule.sol
- contracts/xp/XPV1.sol
- contracts/exchange/DepositHandler.sol
- contracts/error/Errors.sol
- contracts/pricing/PositionPricingUtils.sol
- contracts/lego/interfaces
- contracts/lego/HFBank.sol
- contracts/lego/HFBankFactory.sol
- contracts/lego/WrappedToken.sol
- contracts/liquidation/EarlyForceLiquidationUtils.sol
- contracts/liquidation/LiquidationUtils.sol
- contracts/router/relay/RelayUtils.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
contracts-v2-bsc-audit	Version 0	860811ca56a51529cd969c46b62e283376cbfd6e
	Version 1	7ce2eed90f53aa323020720a2f90edcdb4a8dd10
	Version 2	90c56ed755345bdd752988c0d6257a1294e58dc7

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any

decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism
* Economic and incentive impact

1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

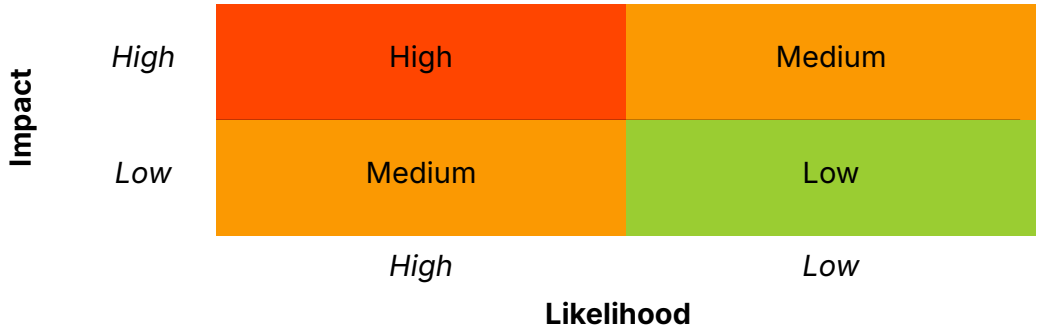
Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification



Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **eight** potential security issues. Besides, we have **five** recommendations and **seven** notes.

- High Risk: 2
- Medium Risk: 5
- Low Risk: 1
- Recommendation: 5
- Note: 7

ID	Severity	Description	Category	Status
1	High	Improper loss rebate design due to the dependence of the collateral amount	Security Issue	Fixed
2	High	Incorrect check in function `_validateBorrowReserve()`	Security Issue	Fixed
3	Medium	Circumvention of self-referral restriction via the function `transferCode()`	Security Issue	Confirmed
4	Medium	Loss rebate extraction using hedging positions	Security Issue	Confirmed
5	Medium	Improper claimable accounting due to malicious transfers	Security Issue	Confirmed
6	Medium	Incorrect bank revocation logic	Security Issue	Fixed
7	Medium	Griefing attack via cross-bank redemption	Security Issue	Fixed
8	Low	Lack of mapping between `WrappedToken` and underlying asset	Security Issue	Fixed
9	-	Add a decimal check for the input `_rewardToken`	Recommendation	Confirmed
10	-	Add non-zero checks in the function `batchSetXP()`	Recommendation	Confirmed
11	-	Add validations in function `_validateMarketTokens()`	Recommendation	Fixed
12	-	Correct error name for receiver balance check	Recommendation	Fixed
13	-	Remove redundant code	Recommendation	Fixed
14	-	Potential centralization risks	Note	-
15	-	Ensure sufficient USDT reserves in the contract `CreditProfitClaimVault`	Note	-
16	-	Exclude the credit token market from HLV creations	Note	-
17	-	Configuration assumptions	Note	-
18	-	Weird ERC20 tokens	Note	-

19	-	Assumptions of the borrowing mechanism	Note	-
20	-	Proper adaptations between the credit and lego modules	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Improper loss rebate design due to the dependence of the collateral amount

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `IncreasePositionUtils`, the function `_calculateLossRebate()` computes the potential loss rebate based on users' collateral and the OI difference. However, when a user creates a position with a large collateral amount and a small position size, the calculated loss rebate is potentially larger than or equal to the position value. As a result, the user may obtain a position without taking any risks of loss.

```
360   function _calculateLossRebate(
361       PositionUtils.UpdatePositionParams memory params,
362       Price.Props memory collateralTokenPrice,
363       uint256 collateralIncrementAmount,
364       uint256 lossRebateFactor
365   ) internal view {
366       // read OI before it gets updated (updateOpenInterest is called later)
367       // reuse weakOI/strongOI to minimize local variables (avoid stack too deep)
368       uint256 weakOI;
369       uint256 strongOI;
370       {
371           uint256 longOI = MarketUtils.getOpenInterest(params.contracts.dataStore, params.market,
                  true);
372           uint256 shortOI = MarketUtils.getOpenInterest(params.contracts.dataStore, params.market,
                   false);
373           weakOI = params.position.isLong() ? longOI : shortOI;
374           strongOI = params.position.isLong() ? shortOI : longOI;
375       }
376
377       // user must be on the weak side
378       if (weakOI >= strongOI) {
379           return;
380       }
381
382       uint256 oiDiff = strongOI - weakOI;
383
384       // check threshold: sizeDeltaUsd must not cause worse imbalance after potential flip
385       // threshold = (strongOI + weakOI) * (strongOI - weakOI) / weakOI
```

```
386       // factored form of (strongOI² - weakOI²) / weakOI to avoid overflow
387       // if weakOI == 0, threshold = infinity, any size qualifies
388       if (weakOI > 0 && params.order.sizeDeltaUsd() > Precision.mulDiv(strongOI + weakOI, oiDiff,
              weakOI)) {
389           return;
390       }
391
392       // eligible collateral is capped by OI difference (only the balancing portion counts)
393       // lrUsdDelta = collateralUsd * min(sizeDelta, oiDiff) / sizeDelta * lossRebateFactor /
              FLOAT_PRECISION
394       uint256 lrUsdDelta = Precision.applyFactor(
395           Precision.mulDiv(
396               collateralIncrementAmount * collateralTokenPrice.min,
397               params.order.sizeDeltaUsd() < oiDiff ? params.order.sizeDeltaUsd() : oiDiff,
398               params.order.sizeDeltaUsd()
399           ),
400           lossRebateFactor
401       );
402
403       params.position.setPendingLossRebateUsd(
404           params.position.pendingLossRebateUsd() + lrUsdDelta
405       );
406   }
```

Listing 2.1: contracts-v2-bsc-audit/contracts/position/IncreasePositionUtils.sol

Impact Users can obtain positions without taking any risks of loss.

Suggestion Revise the code accordingly.

2.1.2 Incorrect check in function `_validateBorrowReserve()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `HFBank`, function `_validateBorrowReserve()` computes the reserve ratio as `reserve / wrappedToken.totalSupply()` and enforces each bank's own `minReserveFactor`. Because `wrappedToken` is designed to be shared across multiple `HFBank` instances (line 19), function `totalSupply()` reflects the total supply minted by all bank instances, not just the current bank performing the check. As a result, when another bank redeems the same wrapped tokens, the shared total supply decreases. This artificially loosens the reserve ratio check in the current bank, even though its own reserves did not improve. Conversely, minting in another bank can artificially tighten borrowing in the current bank.

```
18 *      The two WrappedToken contracts are passed in externally
19 *      at initialize time, allowing multiple banks to share the same WrappedToken instance.
```

Listing 2.2: contracts-v2-bsc-audit/contracts/lego/HFBank.sol

```
651   function _validateBorrowReserve(IERC20 token, WrappedToken wrappedToken, uint256 amount)
          private view {
```

```
652       uint256 reserve = token.balanceOf(address(this));
653       if (reserve < amount) revert InsufficientBankReserve(address(token), reserve, amount);
654
655       BankStorage storage store = _getBankStorage();
656       if (store.minReserveFactor == 0) {
657           return;
658       }
659
660       uint256 supply = wrappedToken.totalSupply();
661       if (supply == 0) {
```

Listing 2.3: contracts-v2-bsc-audit/contracts/lego/HFBank.sol

Impact Mints and redemptions in a bank affect the reserve ratio check in other banks that share the same wrapped token.

Suggestion Revise the code logic accordingly.

2.1.3 Circumvention of self-referral restriction via the function `transferCode()`

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In the contract `ReferralStorage`, the function `_bindReferrer()` prevents direct self-referral when a trader binds a referral code. However, the function `transferCode()` does not check whether the new owner has already bound to the same referral code. Thus, after a user binds to another account's code, the code can later be transferred to that same user, causing the user to become their own referrer. As a result, a trader can circumvent the intended self-referral restriction and receive both L1 and L2 referral benefits from their own trades.

```
118   function transferCode(bytes32 _code, address _newOwner) external override {
119       require(_code != bytes32(0), "ReferralStorage: invalid _code");
120       require(_newOwner != address(0), "ReferralStorage: invalid _newOwner");
121       require(codeOwners[_code] == msg.sender, "ReferralStorage: forbidden");
122
123       codeOwners[_code] = _newOwner;
124       emit TransferCode(msg.sender, _newOwner, _code);
125   }
```

Listing 2.4: contracts-v2-bsc-audit/contracts/mock/ReferralStorage.sol

```
140   function _bindReferrer(address _account, bytes32 _code) private {
141       if (_code == bytes32(0)) { return; }
142
143       address referrer = codeOwners[_code];
144       require(referrer != address(0), "ReferralStorage: code not found");
145       require(referrer != _account, "ReferralStorage: self-referral");
146
147       traderReferralCodes[_account] = _code;
148
149       // auto-assign tier 1 if user has no tier
```

```
150         if (referrerTiers[_account] == 0) {
151             referrerTiers[_account] = 1;
152         }
153     }
```

Listing 2.5: contracts-v2-bsc-audit/contracts/mock/ReferralStorage.sol

Impact A trader can circumvent self-referral restriction and benefit from L1 and L2 affiliate rewards.

Suggestion In the function `transferCode()`, reject transfers where `_newOwner` has already bound to `_code`.

Feedback from the project The project stated that the risk is acceptable.

2.1.4 Loss rebate extraction using hedging positions

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In the contract `IncreasePositionUtils`, the function `_calculateLossRebate()` grants a pending loss rebate when a user increases a position on the weak OI side. The pending loss rebate remains when the weak side flips. However, this design is potentially vulnerable to the loss rebate extraction using hedging positions. Specifically, a user could open a position on the weak side to enforce the flip and subsequently open a hedging position on the new weak side. As a result, both positions could obtain corresponding loss rebates for potential losses and the user could gain profits, which is theoretically from the loss rebate, using hedging positions.

```
360     function _calculateLossRebate(
361         PositionUtils.UpdatePositionParams memory params,
362         Price.Props memory collateralTokenPrice,
363         uint256 collateralIncrementAmount,
364         uint256 lossRebateFactor
365     ) internal view {
366         // read OI before it gets updated (updateOpenInterest is called later)
367         // reuse weakOI/strongOI to minimize local variables (avoid stack too deep)
368         uint256 weakOI;
369         uint256 strongOI;
370         {
371             uint256 longOI = MarketUtils.getOpenInterest(params.contracts.dataStore, params.market,
                    true);
372             uint256 shortOI = MarketUtils.getOpenInterest(params.contracts.dataStore, params.market,
                     false);
373             weakOI = params.position.isLong() ? longOI : shortOI;
374             strongOI = params.position.isLong() ? shortOI : longOI;
375         }
376
377         // user must be on the weak side
378         if (weakOI >= strongOI) {
379             return;
380         }
```

```
381
382       uint256 oiDiff = strongOI - weakOI;
383
384       // check threshold: sizeDeltaUsd must not cause worse imbalance after potential flip
385       // threshold = (strongOI + weakOI) * (strongOI - weakOI) / weakOI
386       // factored form of (strongOI² - weakOI²) / weakOI to avoid overflow
387       // if weakOI == 0, threshold = infinity, any size qualifies
388       if (weakOI > 0 && params.order.sizeDeltaUsd() > Precision.mulDiv(strongOI + weakOI, oiDiff,
              weakOI)) {
389           return;
390       }
391
392       // eligible collateral is capped by OI difference (only the balancing portion counts)
393       // lrUsdDelta = collateralUsd * min(sizeDelta, oiDiff) / sizeDelta * lossRebateFactor /
              FLOAT_PRECISION
394       uint256 lrUsdDelta = Precision.applyFactor(
395           Precision.mulDiv(
396               collateralIncrementAmount * collateralTokenPrice.min,
397               params.order.sizeDeltaUsd() < oiDiff ? params.order.sizeDeltaUsd() : oiDiff,
398               params.order.sizeDeltaUsd()
399           ),
400           lossRebateFactor
401       );
402
403       params.position.setPendingLossRebateUsd(
404           params.position.pendingLossRebateUsd() + lrUsdDelta
405       );
406   }
```

Listing 2.6: contracts-v2-bsc-audit/contracts/position/IncreasePositionUtils.sol

Impact A user could extract the loss rebate as profits by using hedging positions.

Suggestion Revise the logic accordingly.

Feedback from the project The project stated that the risk is acceptable and they will mitigate the potential risk by decreasing the LR rate.

2.1.5 Improper claimable accounting due to malicious transfers

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In the contract `CreditUtils`, the function `applyCreditFeeOffset()` computes the available credit fee, which is based on the difference between a user's actual credit token balance (i.e., the variable `holding`) and their existing claimable accounting (i.e., the variable `existingClaimable`). Then, the function applies a credit fee offset by updating the user's claimable states and transferring the corresponding collateral to the contract `CreditFeeClaimVault`. However, this design may lead to the accounting manipulation due to malicious credit token transfers. Specifically, a user could transfer credit tokens to other accounts to inflate the accounts'

credit fee offset. As a result, the account's claimable state may be inflated, disrupting the accounting.

```
74      uint256 holding = IERC20(creditToken).balanceOf(account);
75      if (holding == 0) {
76          return 0;
77      }
78
79      bytes32 claimableKey = Keys.creditClaimableAmountKey(account);
80      uint256 existingClaimable = dataStore.getUint(claimableKey);
81
82      // amount of credit-backed value the user has not yet recorded as claimable
83      uint256 availableCredit = holding > existingClaimable
84          ? holding - existingClaimable
85          : 0;
86      if (availableCredit == 0) {
87          return 0;
88      }
89
90      creditOffset = feeAmountForPool < availableCredit
91          ? feeAmountForPool
92          : availableCredit;
93
94      dataStore.incrementUint(claimableKey, creditOffset);
```

Listing 2.7: contracts-v2-bsc-audit/contracts/credit/CreditUtils.sol

```
96      address feeClaimVault = dataStore.getAddress(Keys.creditFeeClaimVaultKey());
97      if (feeClaimVault == address(0)) {
98          revert Errors.CreditFeeClaimVaultNotSet();
99      }
100
101     // physically transfer the offset amount of payout token (collateralToken == USDT)
102     // from the MarketToken to the CreditFeeClaimVault so the vault has matching
103     // reserves when the user later swaps their Credit Token.
104     MarketToken(payable(market)).transferOut(collateralToken, feeClaimVault, creditOffset);
```

Listing 2.8: contracts-v2-bsc-audit/contracts/credit/CreditUtils.sol

```
43      bytes32 claimableKey = Keys.creditClaimableAmountKey(msg.sender);
44      uint256 claimable = dataStore.getUint(claimableKey);
45      if (amount > claimable) {
46          revert Errors.CreditClaimableExceeded(amount, claimable);
47      }
48      dataStore.setUint(claimableKey, claimable - amount);
49
50      address creditToken = dataStore.getAddress(Keys.creditTokenKey());
51      address payoutToken = dataStore.getAddress(Keys.creditPayoutTokenKey());
52      if (payoutToken == address(0)) {
53          revert Errors.CreditPayoutTokenNotSet();
54      }
55
56      // pull Credit Token from the user; the vault holds these as a sink and
57      // admins can sweep / burn them later via adminWithdraw
```

```
58        IERC20(creditToken).safeTransferFrom(msg.sender, address(this), amount);
59
60        // send payout token (USDT) to the user
61        _transferOut(payoutToken, msg.sender, amount);
62
63        CreditEventUtils.emitCreditSwapped(eventEmitter, msg.sender, amount, creditToken,
              payoutToken);
64    }
```

Listing 2.9: contracts-v2-bsc-audit/contracts/credit/CreditFeeClaimVault.sol

```
1 // SPDX-License-Identifier: BUSL-1.1
2
3 pragma solidity ^0.8.0;
4
5 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
6
7 import "../role/RoleModule.sol";
8
9 // @title CreditToken
10 // @dev ERC20 representing Hertzflow  (Credit Token).
11 //     Held by users who can use it to offset one-time fees during trading,
12 //     and exchange it 1:1 for the configured payout token (USDT) at the CreditFeeClaimVault.
13 //
14 //     The mint / burn functions are gated by the CREDIT_KEEPER role rather than
15 //     CONTROLLER so that issuance authority is independent from the protocol's
16 //     core CONTROLLER set.
17 contract CreditToken is ERC20, RoleModule {
18     uint8 private immutable _decimals;
19
20     constructor(
21         RoleStore _roleStore,
22         string memory name_,
23         string memory symbol_,
24         uint8 decimals_
25     ) ERC20(name_, symbol_) RoleModule(_roleStore) {
26         _decimals = decimals_;
27     }
28
29     function decimals() public view virtual override returns (uint8) {
30         return _decimals;
31     }
32
33     // @dev mint Credit Token to an account
34     function mint(address account, uint256 amount) external onlyCreditKeeper {
35         _mint(account, amount);
36     }
37
38     // @dev burn Credit Token from an account
39     function burn(address account, uint256 amount) external onlyCreditKeeper {
40         _burn(account, amount);
41     }
42 }
```

Listing 2.10: contracts-v2-bsc-audit/contracts/credit/CreditToken.sol

```
413   function getPositionFeesAndApplyCreditOffset(
414       GetPositionFeesParams memory params,
415       EventEmitter eventEmitter
416   ) external returns (PositionFees memory) {
417       PositionFees memory fees = getPositionFees(params);
418       uint256 creditOffset = CreditUtils.applyCreditFeeOffset(
419           params.dataStore,
420           eventEmitter,
421           params.position.account(),
422           params.position.market(),
423           params.position.collateralToken(),
424           fees.positionFeeAmountForPool
425       );
426       fees.feeAmountForPool -= creditOffset;
427       fees.positionFeeAmountForPool -= creditOffset;
428       return fees;
```

Listing 2.11: contracts-v2-bsc-audit/contracts/pricing/PositionPricingUtils.sol

Impact The account's claimable state may be inflated due to malicious transfers.

Suggestion Revise the logic accordingly.

Feedback from the project The project stated that there is no loss of funds, only incorrect accounting. They consider the risk acceptable and plan to fix it in the future.

2.1.6 Incorrect bank revocation logic

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `HFBankFactory`, the function `revokeBankMinter()` removes a bank from the `wrappedToken.minter`. However, it does not check whether the bank still holds underlying reserves or has outstanding debt before revocation. Since function `burn()` of the wrapped token is restricted to minters, revoking a bank may block redemption paths that require burning wrapped tokens (e.g., `withdrawQuote()` and `withdrawBase()`).

Additionally, the function `linkBankToMarket()` does not allow reconfiguring an existing market-to-bank mapping. Once a bank is disabled, affected markets may be unable to migrate to a new bank.

```
329   /**
330    * @notice Revoke a bank's minter role from a wrapped token.
331    */
332   function revokeBankMinter(address bank, WrappedToken wrappedToken) external onlyAdmin {
333       if (!isBank[bank]) revert BankNotFromFactory();
334       wrappedToken.removeMinter(bank);
335       emit BankMinterRevoked(bank, address(wrappedToken));
```

```
336     }
```

Listing 2.12: contracts-v2-bsc-audit/contracts/lego/HFBankFactory.sol

```
493     function withdrawBase(uint256 amount) external nonReentrant whenNotPaused onlyAdmin {
494         if (amount == 0) revert ZeroAmount();
495
496         BankStorage storage store = _getBankStorage();
497         store.wrappedBaseToken.burn(msg.sender, amount);
498         _validateAvailableReserve(store.baseToken, amount);
499         store.baseToken.safeTransfer(msg.sender, amount);
500
501         emit WithdrawBase(msg.sender, amount);
502     }
503
504     /**
505      * @notice Withdraw quote tokens by burning wrapped quote tokens (admin only)
506      * @param amount Amount to withdraw
507      */
508     function withdrawQuote(uint256 amount) external nonReentrant whenNotPaused onlyAdmin {
509         if (amount == 0) revert ZeroAmount();
510
511         BankStorage storage store = _getBankStorage();
512         store.wrappedQuoteToken.burn(msg.sender, amount);
513         _validateAvailableReserve(store.quoteToken, amount);
514         store.quoteToken.safeTransfer(msg.sender, amount);
515
516         emit WithdrawQuote(msg.sender, amount);
517     }
```

Listing 2.13: contracts-v2-bsc-audit/contracts/lego/HFBank.sol

```
156     function burn(address from, uint256 amount) external onlyMinter {
157         _burn(from, amount);
158     }
```

Listing 2.14: contracts-v2-bsc-audit/contracts/lego/WrappedToken.sol

```
220     function linkBankToMarket(address bank, address gmxMarket) external onlyAdmin {
221         if (gmxMarket == address(0)) revert ZeroAddress();
222         if (bankByMarket[gmxMarket] != address(0)) revert MarketAlreadyLinked();
223
224         // Verify bank is from this factory
225         if (!isBank[bank]) revert BankNotFromFactory();
226
227         _validateMarketTokens(bank, gmxMarket);
228
229         // Update mapping
230         bankByMarket[gmxMarket] = bank;
231
232         emit BankMarketSet(bank, gmxMarket);
233     }
```

Listing 2.15: contracts-v2-bsc-audit/contracts/lego/HFBankFactory.sol

Impact 1. The underlying assets held by the revoked bank may become inaccessible, leading to a potential fund lock. 2. The affected market may also remain linked to an unusable bank, preventing migration to another bank.

Suggestion Revise the code logic accordingly.

2.1.7 Griefing attack via cross-bank redemption

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `HFBank`, the function `flashMint()` pulls real tokens and mints `WrappedToken`. The function only requires minted wrapped tokens to be consumed during the callback.

If multiple banks share the same wrapped token, the callback can redeem the minted tokens via functions `redeemBase()` and `redeemQuote()` on a different bank. The current bank retains the pulled tokens, while the target bank pays out from its own reserves. Malicious actors can exploit this to deplete the reserve of a target bank, disrupting its redemptions, borrow reserve checks, and settlement or withdrawal flows.

```
356   function flashMint(
357       address receiver,
358       uint256 baseAmount,
359       uint256 quoteAmount,
360       bytes calldata data
361   ) external nonReentrant whenNotPaused {
362       if (baseAmount == 0 && quoteAmount == 0) revert ZeroAmount();
363       if (receiver == address(0)) revert ZeroAddress();
364
365       BankStorage storage store = _getBankStorage();
366       FlashBalanceSnapshot memory snapshot;
367       snapshot.wrappedBaseToken = store.wrappedBaseToken;
368       snapshot.wrappedQuoteToken = store.wrappedQuoteToken;
369       snapshot.useBase = baseAmount > 0;
370       snapshot.useQuote = quoteAmount > 0;
371       snapshot.diff = receiver != msg.sender;
372
373       // Snapshot pre-flash balances so admins (permanent whitelist holders) can use flashMint
374       snapshot.senderBaseBefore = snapshot.useBase ? snapshot.wrappedBaseToken.balanceOf(msg.
              sender) : 0;
375       snapshot.senderQuoteBefore = snapshot.useQuote ? snapshot.wrappedQuoteToken.balanceOf(msg.
              sender) : 0;
376       snapshot.receiverBaseBefore = (snapshot.useBase && snapshot.diff) ? snapshot.
              wrappedBaseToken.balanceOf(receiver) : 0;
377       snapshot.receiverQuoteBefore = (snapshot.useQuote && snapshot.diff) ? snapshot.
              wrappedQuoteToken.balanceOf(receiver) : 0;
378
379       _setFlashMintTemporaryHolders(snapshot, receiver, true);
380
381       // Pull real tokens, mint wrapped to caller
382       if (snapshot.useBase) {
```

```
383             store.baseToken.safeTransferFrom(msg.sender, address(this), baseAmount);
384             snapshot.wrappedBaseToken.mint(msg.sender, baseAmount);
385             emit DepositBase(msg.sender, baseAmount);
386         }
387         if (snapshot.useQuote) {
388             store.quoteToken.safeTransferFrom(msg.sender, address(this), quoteAmount);
389             snapshot.wrappedQuoteToken.mint(msg.sender, quoteAmount);
390             emit DepositQuote(msg.sender, quoteAmount);
391         }
392
393         if (IFlashMintReceiver(receiver).onFlashMint(baseAmount, quoteAmount, data) !=
                CALLBACK_SUCCESS) {
394             revert InvalidCallbackReturn();
395         }
396
397         _setFlashMintTemporaryHolders(snapshot, receiver, false);
398
399         // All wrapped must be consumed: balances back to pre-flash levels
400         if (snapshot.useBase && snapshot.wrappedBaseToken.balanceOf(msg.sender) != snapshot.
                senderBaseBefore) {
401             revert CallerStillHoldsWrappedTokens();
402         }
403         if (snapshot.useQuote && snapshot.wrappedQuoteToken.balanceOf(msg.sender) != snapshot.
                senderQuoteBefore) {
404             revert CallerStillHoldsWrappedTokens();
405         }
406         if (
407             snapshot.diff &&
408             snapshot.useBase &&
409             snapshot.wrappedBaseToken.balanceOf(receiver) != snapshot.receiverBaseBefore
410         ) {
411             revert CallerStillHoldsWrappedTokens();
412         }
413         if (
414             snapshot.diff &&
415             snapshot.useQuote &&
416             snapshot.wrappedQuoteToken.balanceOf(receiver) != snapshot.receiverQuoteBefore
417         ) {
418             revert CallerStillHoldsWrappedTokens();
419         }
420
421         emit FlashMint(msg.sender, receiver, baseAmount, quoteAmount);
422     }
```

Listing 2.16: contracts-v2-bsc-audit/contracts/lego/HFBank.sol

```
526     function redeemBase(uint256 amount, address receiver) external nonReentrant whenNotPaused {
527         if (amount == 0) revert ZeroAmount();
528         if (receiver == address(0)) revert ZeroAddress();
529
530         BankStorage storage store = _getBankStorage();
531         _validateAvailableReserve(store.baseToken, amount);
532         store.wrappedBaseToken.burn(msg.sender, amount);
```

```
533         store.baseToken.safeTransfer(receiver, amount);
534
535         emit RedeemBase(msg.sender, receiver, amount);
536     }
```

Listing 2.17: contracts-v2-bsc-audit/contracts/lego/HFBank.sol

Impact Depleting a bank's real token reserves can disrupt bank functionality.

Suggestion Revise the code logic accordingly.

2.1.8 Lack of mapping between `WrappedToken` and underlying asset

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `HFBankFactory`, function `createBank()` records the caller-supplied underlying asset and `WrappedToken` into the new bank via `HFBank.initialize()`. The annotation of contract `WrappedToken`, "ERC20 wrapped representation of an underlying asset", implies a one-to-one mapping between them. However, neither the factory nor the bank enforces this. Consequently, one wrapped token can represent multiple underlying assets, while one underlying asset can be represented by multiple wrapped tokens across different banks.

```
119     function createBank(
120         address baseToken,
121         address quoteToken,
122         WrappedToken wrappedBase,
123         WrappedToken wrappedQuote
124     ) external onlyAdmin returns (address bank) {
125         if (!allowedBaseTokens[baseToken]) revert BaseTokenNotAllowed(baseToken);
126         if (!allowedBaseTokens[quoteToken]) revert BaseTokenNotAllowed(quoteToken);
127
128         bytes memory initData = abi.encodeWithSelector(
129             HFBank.initialize.selector,
130             baseToken,
131             quoteToken,
132             wrappedBase,
133             wrappedQuote,
134             admin
135         );
```

Listing 2.18: contracts-v2-bsc-audit/contracts/lego/HFBankFactory.sol

```
136     function initialize(
137         address baseToken_,
138         address quoteToken_,
139         WrappedToken wrappedBase_,
140         WrappedToken wrappedQuote_,
141         address admin_
142     ) external initializer {
143         if (baseToken_ == address(0) || quoteToken_ == address(0) || admin_ == address(0)) {
```

```
144             revert ZeroAddress();
145         }
146         if (address(wrappedBase_) == address(0) || address(wrappedQuote_) == address(0)) {
147             revert ZeroAddress();
148         }
149         if (wrappedBase_.decimals() != IERC20Metadata(baseToken_).decimals()) revert
                DecimalsMismatch();
150         if (wrappedQuote_.decimals() != IERC20Metadata(quoteToken_).decimals()) revert
                DecimalsMismatch();
```

Listing 2.19: contracts-v2-bsc-audit/contracts/lego/HFBank.sol

```
8 * @notice ERC20 wrapped representation of an underlying asset, with role-based mint/burn and
      whitelist.
```

Listing 2.20: contracts-v2-bsc-audit/contracts/lego/WrappedToken.sol

Impact This breaks the intended one-to-one mapping between the wrapped token and its corresponding underlying asset.

Suggestion Revise the code logic accordingly.

2.2 Recommendation

2.2.1 Add a decimal check for the input `_rewardToken`

Status Confirmed

Introduced by `Version 1`

Description The documentation in the XP module specifies that users claim the reward token (i.e., USDT) at a 1:1 ratio with XP using 18-decimal precision. Therefore, it is recommended to add decimal checks for the input `_rewardToken` in the `constructor()` of the contract `XPV1` to prevent potential misconfigurations.

```
41  constructor(
42      address _rewardToken,
43      address _xpKeeper,
44      uint256 _campaignEndTime
45  ) {
46      if (_rewardToken == address(0)) revert ZeroAddress();
47      if (_xpKeeper == address(0)) revert ZeroAddress();
48
49      rewardToken = IERC20(_rewardToken);
50      xpKeeper = _xpKeeper;
51      campaignEndTime = _campaignEndTime;
52  }
```

Listing 2.21: contracts-v2-bsc-audit/contracts/xp/XPV1.sol

Suggestion Add decimal checks for the input `_rewardToken` in the `constructor()` of the contract `XPV1`.

2.2.2 Add non-zero checks in the function `batchSetXP()`

Status Confirmed

Introduced by `Version 1`

Description In the contract `XPV1`, the function `forceSetXP()` validates the variable `user` is not the zero address, while the function `batchSetXP()` does not apply such validation. It is recommended to add non-zero checks in the function `batchSetXP()` for code consistency.

```
 99   function batchSetXP(
100       address[] calldata users,
101       uint256[] calldata xpValues
102   ) external onlyXPKeeper whenNotPaused {
103       uint256 len = users.length;
104       if (len != xpValues.length) revert ArrayLengthMismatch(len, xpValues.length);
105
106       if (campaignEndTime != 0 && block.timestamp > campaignEndTime) {
107           revert CampaignEnded(campaignEndTime);
108       }
109
110       uint256 totalDelta;
111       for (uint256 i; i < len; ) {
112           address user = users[i];
113           uint256 newXP = xpValues[i];
114           uint256 oldXP = xp[user];
115
116           if (newXP != oldXP) {
117               if (newXP < oldXP) revert XPDecreased(user, oldXP, newXP);
118               xp[user] = newXP;
119               totalDelta += (newXP - oldXP);
120               emit XPUpdated(user, oldXP, newXP);
121           }
122
123           unchecked { ++i; }
124       }
125       totalXP += totalDelta;
126   }
```

Listing 2.22: contracts-v2-bsc-audit/contracts/xp/XPV1.sol

Suggestion Add non-zero checks in the function `batchSetXP()`.

2.2.3 Add validations in function `_validateMarketTokens()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `HFBankFactory`, the function `_validateMarketTokens()` returns directly when the variable `dataStore` is not configured. Consequently, a bank may be linked to a market without token validation.

Additionally, the function does not validate that `liquidationSettlementHook` is set or that the hook address can receive wrapped tokens. These configurations are critical to the liquidation flow.

It is recommended to validate these configurations before linking a bank to a market.

```
354   function _validateMarketTokens(address bank, address gmxMarket) private view {
355       if (address(dataStore) == address(0)) {
356           return;
357       }
```

Listing 2.23: contracts-v2-bsc-audit/contracts/lego/HFBankFactory.sol

```
72        cache.lastSrcChainId = dataStore.getUint(Keys.positionLastSrcChainId(cache.positionKey));
73        cache.receiver = dataStore.getAddress(Keys.liquidationSettlementHook(market, cache.
              collateralToken));
74        if (cache.receiver == address(0)) {
75            cache.receiver = account;
76            cache.callbackContract = CallbackUtils.getSavedCallbackContract(dataStore, account,
                  market);
77        } else {
78            cache.callbackContract = cache.receiver;
79        }
```

Listing 2.24: contracts-v2-bsc-audit/contracts/liquidation/EarlyForceLiquidationUtils.sol

Suggestion Revert when the variable `dataStore` is not configured. Validate the market tokens, the `liquidationSettlementHook`, and the hook address's wrapped token whitelist.

2.2.4 Correct error name for receiver balance check

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `HFBank`, the function `flashMint()` checks both sender and receiver wrapped-token balances. However, receiver-side failures revert with the `CallerStillHoldsWrappedTokens` error, which does not match the failed account.

```
406       if (
407           snapshot.diff &&
408           snapshot.useBase &&
409           snapshot.wrappedBaseToken.balanceOf(receiver) != snapshot.receiverBaseBefore
410       ) {
411           revert CallerStillHoldsWrappedTokens();
412       }
413       if (
414           snapshot.diff &&
415           snapshot.useQuote &&
416           snapshot.wrappedQuoteToken.balanceOf(receiver) != snapshot.receiverQuoteBefore
417       ) {
418           revert CallerStillHoldsWrappedTokens();
419       }
```

Listing 2.25: contracts-v2-bsc-audit/contracts/lego/HFBank.sol

Suggestion Add a receiver-specific wrapped balance error.

2.2.5 Remove redundant code

Status Fixed in Version 2

Introduced by Version 1

Description In the contract HFBank, function gmxMarket() returns the primary market recorded in the storage. However, the function is never invoked in this project. It is recommended to remove this function while preserving the existing storage layout.

```
191    function gmxMarket() external view returns (address) {
192        return _getBankStorage().gmxMarket;
193    }
```

Listing 2.26: contracts-v2-bsc-audit/contracts/lego/HFBank.sol

Suggestion Remove the redundant code.

2.3 Note

2.3.1 Potential centralization risks

Introduced by Version 1

Description In this project, several privileged roles, such as the HFBankFactory admin, HFBank admin, WrappedToken admin, DataStore controller, and liquidation hook configurator, can perform sensitive operations, which introduces potential centralization risks. For example, privileged accounts can create banks with selected real and wrapped token pairs, link banks to wrapped tokens, configure protocol parameters, authorize wrap callers and receivers, manage wrapped-token holder permissions, and route liquidation settlement through configured hooks. If the private keys of privileged accounts are lost, compromised, or used maliciously, the protocol may suffer from improper configuration, unauthorized fund movement, or service disruption. The review assumes these privileged configurations are set and maintained correctly as part of the protocol operation.

2.3.2 Ensure sufficient USDT reserves in the contract CreditProfitClaimVault

Introduced by Version 1

Description In the contract CreditProfitClaimVault, the payout tokens (i.e., USDT) is required to be pre-funded for profit settlements. The project must ensure the vault contains sufficient payout tokens for proper functionalities.

2.3.3 Exclude the credit token market from HLV creations

Introduced by Version 1

Description In the contract DepositHandler, the function createDeposit() only allows the role CREDIT_MARKET_KEEPER to add liquidity to the credit token market. However, if the project creates

a HLV token for the credit token market, the ACL design may be broken. The project must ensure that the credit token market is not involved in a HLV token creation to avoid potential ACL.

2.3.4 Configuration assumptions

Introduced by `Version 1`

Description Since the `WrappedToken` is used as the long token or short token of a market, related market flows depend on correct configuration and proper operator behavior. There are some examples as follows:

1. Liquidation settlement requires an operator to receive `WrappedToken` before redeeming it for the underlying asset. The operator must be configured in the protocol `DataStore` and allowed to receive the relevant `WrappedToken`.

2. When a position reduction order is created, the receiver is checked only as a non-zero address. If output tokens are returned as wrapped tokens, the transfer requires the receiver or the fallback `HOLDING_ADDRESS` to be allowed by the wrapped token whitelist. Therefore, the project must provide a designated operator for position reductions, and users must set this operator as the receiver.

3. The designated operators must implement the correct processing logic to ensure timely and correct settlement (e.g., liquidation), including retry logic for failed redemptions.

Feedback from the project The project states that each `WrappedToken` will have an operator that serves as an intermediary for all related flows. The project ensures the correct implementation of the logic for these operators.

2.3.5 Weird ERC20 tokens

Introduced by `Version 1`

Description In this project, it is important to note that weird `ERC20` tokens, such as fee-on-transfer or rebasing tokens, are not supported and may result in unexpected behavior. The real tokens (i.e., `baseToken`, `quoteToken`) used by contract `HFBank` are expected to behave like standard ERC20 tokens and preserve a 1:1 relationship between transferred amounts and wrapped token supply. If these tokens apply transfer fees or rebased balances, wrapping, redemption, and reserve accounting may deviate from the intended result.

2.3.6 Assumptions of the borrowing mechanism

Introduced by `Version 1`

Description This project allows authorized borrowers to borrow underlying assets from contract `HFBank` without providing collateral. These borrowers are trusted roles operated by the project. To limit how much they can borrow, contract `HFBank` enforces a `minReserveFactor` guard. However, this guard is only effective once it is explicitly set to a non-zero value. The project should authorize borrowers only after configuring this value appropriately, and raise it when needed.

Contract `HFBank` itself pays no yield to depositors. Instead, borrowers pay an extra surplus to liquidity providers in markets, not traders. Specifically, borrowers wrap operational surplus into wrapper tokens, then deposit them into the target market to raise LP share value. Borrowers should ensure that rewards are distributed fairly to liquidity providers in return for their participation in the market.

Even with the `minReserveFactor` guard in place, underlying reserves are not guaranteed to cover every cancellation, withdrawal, or settlement in the markets. The project must monitor reserve balance, outstanding debt, wrapped supply, and reserve ratio off-chain carefully, and trigger repayments promptly.

2.3.7 Proper adaptations between the credit and lego modules

Introduced by `Version 1`

Description The lego and credit modules are both newly introduced during this audit. Compatibility between the two is outside the scope of this audit. The project stated that they will implement a proper adaptation for two modules in the future.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS