Felix

Smart Contract Security Assessment

December 18, 2024

ABSTRACT

Dedaub was commissioned to perform a security audit of the Felix protocol, a Liquity v2 fork, which is going to be deployed on Hyperliquid's blockchain.

BACKGROUND

The Felix protocol introduces the following modifications to Liquity v2:

Transparent Proxy Design
Unlike the immutable design of the Liquity v2 protocol, the Felix protocol employs a Transparent Proxy design for each contract. This approach allows for greater flexibility by enabling contract upgrades while preserving existing state and functionality.

Mutable Critical Parameters
The Felix protocol introduces mutability for certain critical parameters via setter functions. These parameters include:

- MCR: The minimum ratio of collateral to debt required to maintain a trove.
- CCR: The ratio at which the system enters recovery mode.
- MaxCapDebt: A new parameter introduced in this fork to prevent overexposure to specific collateral types. It accounts for the total accrued debt (including interest) and enforces the cap only when users attempt to mint additional debt. Debt accumulation from interest is not restricted by this cap and continues as normal.
- Interest Router: Allows updates to the routing mechanism for interest accrual.
- Stability Pool Yield Percentage: Configurable yield percentage for Stability Pool participants.

AdminController for Centralized Governance

Administrative actions are centralized under the AdminController contract, which serves as the owner of all system contracts. Key features include:

- Proposal/Apply Methodology:
  - Changes to critical parameters follow a proposal and apply mechanism.
  - An enforced timelock period between proposal and application ensures users have time to respond to protocol changes.

- Upgradeability:
  - The proxy admin for all contract proxies is owned by the AdminController, which can upgrade contract implementations if necessary.

Removal of Gas Compensation Mechanism

The Felix protocol removes the gas compensation mechanism originally implemented in Liquity v2. In the original protocol, users were rewarded with gas tokens upon liquidating a trove to incentivize liquidators during periods of high gas costs. This mechanism is deemed unnecessary in Felix, as it is designed for chains with minimal gas constraints.

- ETH_GAS_COMPENSATION: This constant has been set to 0. Certain versions of WETH contracts (such as this one: Etherscan link) support allowance, transfer, and deposit operations with a 0 amount, ensuring compatibility with the updated protocol.

- Note: ETH_GAS_COMPENSATION is immutable once set, ensuring consistency for troves already opened under this configuration.

SETTING & CAVEATS

This audit report mainly covers the contracts of the at-the-time private repository https://github.com/threesigmaxyz/bold of the Felix Protocol at commit `f005f2b4ba4b646852fedef66ed2c5236d4b355a` (branch `feat/AddCollateral`). The audit primarily focused on the changes introduced by the Felix team to the original Liquity v2 codebase.

Two auditors worked on the codebase for 6 days on the following contracts:

```
src/
├── ActivePool.sol
├── AddressesRegistry.sol
├── AdminController.sol
├── BoldToken.sol
├── BorrowerOperations.sol
├── CollateralRegistry.sol
├── CollSurplusPool.sol
├── DefaultPool.sol
├── Dependencies/
│   ├── AddRemoveManagers.sol
│   ├── Constants.sol
│   └── LiquityBase.sol
├── GasPool.sol
├── HintHelpers.sol
├── Libraries/
│   ├── BorrowerOperationsInit.sol
│   ├── LiquityBaseInit.sol
│   └── TroveManagerInit.sol
├── MultiTroveGetter.sol
├── PriceFeeds/
│   └── HLPriceFeed.sol
├── SortedTroves.sol
├── StabilityPool.sol
├── TroveManager.sol
├── TroveNFT.sol
└── Zappers/
    ├── BaseZapper.sol
```

```
├── GasCompZapper.sol
├── Interfaces/*
├── LeftoversSweep.sol
├── LeverageLSTZapper.sol
├── LeverageWETHZapper.sol
├── Modules/*
└── WETHZapper.sol
```

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than the regular use of the protocol. Functional correctness (i.e. issues in "regular use") is a secondary consideration. Typically it can only be covered if we are provided with unambiguous (i.e. full-detail) specifications of what is the expected, correct behavior. In terms of functional correctness, we often trusted the code's calculations and interactions, in the absence of any other specification. Functional correctness relative to low-level calculations (including units, scaling and quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues affecting the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third-party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third-party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: • User or system funds can be lost when third-party systems misbehave. • DoS, under specific conditions. • Part of the functionality becomes unusable due to a programming error.
LOW	Examples: • Breaking important system invariants but without apparent consequences. • Buggy functionality for trusted users where a workaround exists. • Security issues which may manifest when the system evolves.

Issue resolution includes "dismissed" or "acknowledged" but no action taken, by the client, or "resolved", per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

ID	Description	STATUS
H1	Critical Liquity v2 upgrades have not been integrated	**RESOLVED**

Several critical upgrades to the Liquity v2 codebase have not been integrated into the Felix codebase.

Below is a partial list of missing commits:

- commit `d5bd31b` (October 23rd) "fix: Apply SP error fix to scale updates"
- commit `87198e9` (October 31st) "fix: SP previous error should be applied to newProductFactor, not P"
- commit `71026e6` (Nov 1st) "fix: Make sure a fully redeemed trove ends up with zero shares in a batch"
- commit `b89ef84` (Nov 1st) "fix: Revert if trying to apply debt to empty trove"
- commit `4dfe520` (November 18th) "feat: Add Zappers Hybrid exchange helpers"
- commit `afc2464` (Nov 25th) "fix: Receiver usage should be constrained to remove manager"
- commit `a689378` (December 4th) "fix: Make sure safeTransfer is used for collateral tokens"

MEDIUM SEVERITY:

ID	Description	STATUS
M1	Users cannot access the protocol's main functions if the oracle fails, exposing them to significant risks	**RESOLVED**
The `HLPriceFeed::fetchPrice` function reverts if the oracle call returns an erroneous `0` value as the price of the asset. This behavior differs from the Liquity v2 implementation, which would automatically shut down the market and use the last known good price to prevent disruption. In the current setup, however, the market can only be shut down manually, adding a layer of reliance on human intervention. As a result, if oracle calls fail during periods of significant market volatility, protocol users may be unable to react in time, potentially resulting in substantial losses.		
M2	The absence of redemption-specific oracle prices might leave the protocol vulnerable to redemption arbitrage	**DISMISSED**
The `HLPriceFeed` contract does not define a `fetchRedemptionPrice` function, unlike the standard Liquity v2 oracles. As a result, the price calculation does not differ between redemptions and other protocol operations. Although the asset prices provided by the HyperEVM system contract are sourced from major CEXs, the lack of a redemption-specific price may leave Felix vulnerable to unwanted redemption arbitrage if someone is able to manipulate the oracles.		

LOW SEVERITY:

ID	Description	STATUS
L1	Mask computation shifts 1 by 20 bits instead of 160	**RESOLVED**

In the assembly block of `LiquityBaseInit::initialize`, the `mask` computation is incorrect. The expression `shl(20, 1)` shifts `1` to the left by `20` bits, whereas the intention is to shift it by `20` bytes (`160` bits).

`LiquityBaseInit::initialize:29-40`

```
assembly {
  /// @dev Storage variables are written from right to left
  /// Thus the active pool address is located at 2 bytes offset
  /// from the right in the slot
  let ptr := sload(ACTIVE_POOL_SLOT)

  // Dedaub:
  // computed mask:
  // 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff00000ffff
  // intended mask:
  // 0xffffffffffffffffffff0000000000000000000000000000000000000000ffff
  let mask := not(shl(mul(ACTIVE_POOL_OFFSET, 8), sub(shl(20, 1), 1)))
  ptr := and(ptr, mask)
  ptr := or(ptr, shl(mul(ACTIVE_POOL_OFFSET, 8), activePool))

  sstore(ACTIVE_POOL_SLOT, ptr)
  sstore(DEFAULT_POOL_SLOT, defaultPool)
  sstore(PRICE_FEED_SLOT, priceFeed)
}
```

Fortunately, the final `ptr` value remains unaffected by the incorrect mask, assuming the 20 slot bytes to be masked are set to `0` during contract initialization.

L2	AdminController::batchAddAddressRegistry will fail for batches sizes greater than 1	**RESOLVED**

The function `AdminController::batchAddAddressRegistry` loops over the `_addressRegistries` array and adds each registry to the `addressesRegistries`

storage array after checking that `_doesIndexMatch(_addressRegistries[i], index += i)` is true.

`AdminController::initialize:385-397`

```
function batchAddAddressRegistry
  IAddressesRegistry[] memory _addressRegistries
) external onlyOwner {
  if(address(collateralRegistry) == address(0)) revert
    AdminController__CollateralRegistryNotSet();
  uint256 index = addressesRegistries.length;

  for(uint256 i = 0; i < _addressRegistries.length; i++) {
    // Dedaub: should be index + i instead of index += i
    if(!_doesIndexMatch(_addressRegistries[i], index += i))
      revert AdminController__IndexMismatch();

    addressesRegistries.push(_addressRegistries[i]);
    emit AddressRegistryAdded(address(_addressRegistries[i]));
  }
}
```

The second argument of `_doesIndexMatch` is used as an index for the `tokens` array of the `CollateralRegistry` contract. Therefore, its starting value should equal `addressesRegistries.length` to account for the existing tokens and address registries, and it should increment by 1 with each subsequent loop iteration. However, the current implementation `index += i` accumulates `i` to `index` on every iteration, whereas `index + i` would achieve the intended behavior.

L3	AdminController::proposeNewCollateral lacks decimal verification	**RESOLVED**

The `AdminController::proposeNewCollateral` function does not verify the number of decimals for the `_newCollateral` token, even though the current protocol version

supports only tokens with `18` decimals. This is a known limitation of the current version; however, a token with a different number of decimals could still be added by mistake.		
L4	Incorrect event name	**RESOLVED**
The event `NewCMRSet`, which is defined in the `TroveManager` contract, should be renamed to `NewMCRSet`.		
L5	No verification to ensure that CCR > MCR in AdminController	**RESOLVED**
When applying an `AdminController` proposal to change the `MCR` or `CCR` parameters, it should be checked to ensure that they do not break a critical invariant of the protocol, i.e., the system's critical collateralization ratio is not set to a value smaller than the maintenance collateralization ratio of individual troves.		
L6	No verification to ensure that CCR > SCR in AdminController	**RESOLVED**
When applying an `AdminController` proposal to change the `CCR` parameter, it should be checked to ensure that `CCR` (critical collateralization ratio) is greater than the `SCR` (shutdown collateralization ratio).		

CENTRALIZATION ISSUES:

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. Even if the owner is reputable, users are more likely to engage with a protocol that guarantees no catastrophic failure even in the case the owner gets hacked/compromised. We list issues of this kind below. (These issues should be considered in the context of usage/deployment, as they are not uncommon. Several high-profile, high-value protocols have significant centralization threats.)

ID	Description	STATUS
N1	Upgradeable contracts	**INFO**
The Felix protocol, unlike Liquity v2, permits contract upgrades while maintaining the existing state through the use of transparent proxies. This design enables essential upgrades to the contracts' logic but also introduces risks. Malicious actors, whether protocol owners or adversaries compromising the owners' accounts, could exploit this feature and introduce malicious code. Additionally, protocol owners might unintentionally introduce buggy functionality during upgrades. The AdminController contract, which facilitates updates, enforces a timelock period (currently set to 3 days) for each contract upgrade. While this provides users with a reasonable window to react, it could still be easily overlooked by a significant number of users.		
N2	Mutable critical parameters	**INFO**
The Felix protocol introduces mutability for the following critical branch parameters: • MCR • CCR • SPYield • InterestRouter • PriceFeed		

- MaxDebtCap

Incorrect or malicious parameter settings can negatively affect the health of a branch or the entire system due to the accrual of excessive bad debt, particularly in inefficient markets. Protocol owners should set these parameters defensively to ensure the stability, security, and resilience of the protocol.

The owners of the protocol can add new collaterals/branches. We believe that choosing collaterals requires careful consideration, as the collapse of a branch (whether due to collateral collapses or branch oracle fails) could create bad debt that is shared across all branches. This, in turn, could lead to bank runs, as described in the known Liquity v2 issues guide.

OTHER / ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	AdminController::applyNewCollateral optimization	**INFO**
In the `AdminController::applyNewCollateral` function, the `_troveManager` parameter is passed directly, though it could be retrieved from the `_addressRegistry` contract to reduce the risk of errors.		
A2	Returning address(0) instead of reverting	**RESOLVED**
The `getToken` and `getTroveManager` functions of `CollateralRegistry` return `address(0)` when the `index` argument is greater than or equal to the number of tokens and trove managers, respectively. We believe that reverting with a clear error in such cases would be a more robust and fail-safe approach.		
A3	The error AdminController__NotZeroValue is never used	**RESOLVED**
The `AdminController__NotZeroValue` error defined in `AdminController` is never used.		
A4	Contracts that won't be deployed should be marked as abstract	**RESOLVED**
Dependencies such as `LiquityBase`, `AddRemoveManageres`, and `Ownable` should be marked as `abstract`.		
A5	Inconsistent CCR and MCR bounds between contracts	**RESOLVED**

The `AddressesRegistry` contract defines bounds for CCR and MCR (lower = `1e18`, higher = `2e18`) that are not consistent with those defined by AdminController (lower = `1.1e18`, higher = `4.5e18`). This difference might be reasonable if we consider that the AdminController bounds will be applied after initialization and can therefore be more relaxed.		
A6	Missing validations	**RESOLVED**
The functions `addAddressRegistry` and `batchAddAddressRegistry` in the `AdminController` contract do not verify that the provided `_addressRegistry` and `_addressRegistries[i]` are not equal to `address(0)`.		
A7	Missing oracle constants	**INFO**
Oracle constants are undefined for assets other than ETH, likely because the potential collateral types have yet to be finalized.		
A8	Incorrect comment	**RESOLVED**
The `@notice` code comment for `AdminController::setCollateralRegistry` is incorrect, as there is a single `CollateralRegistry` contract, not one per branch.		
A9	Compiler bugs	**INFO**
The code is compiled with Solidity `0.8.24`. Version `0.8.24`, in particular, has no known bugs..		

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Security Suite.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.